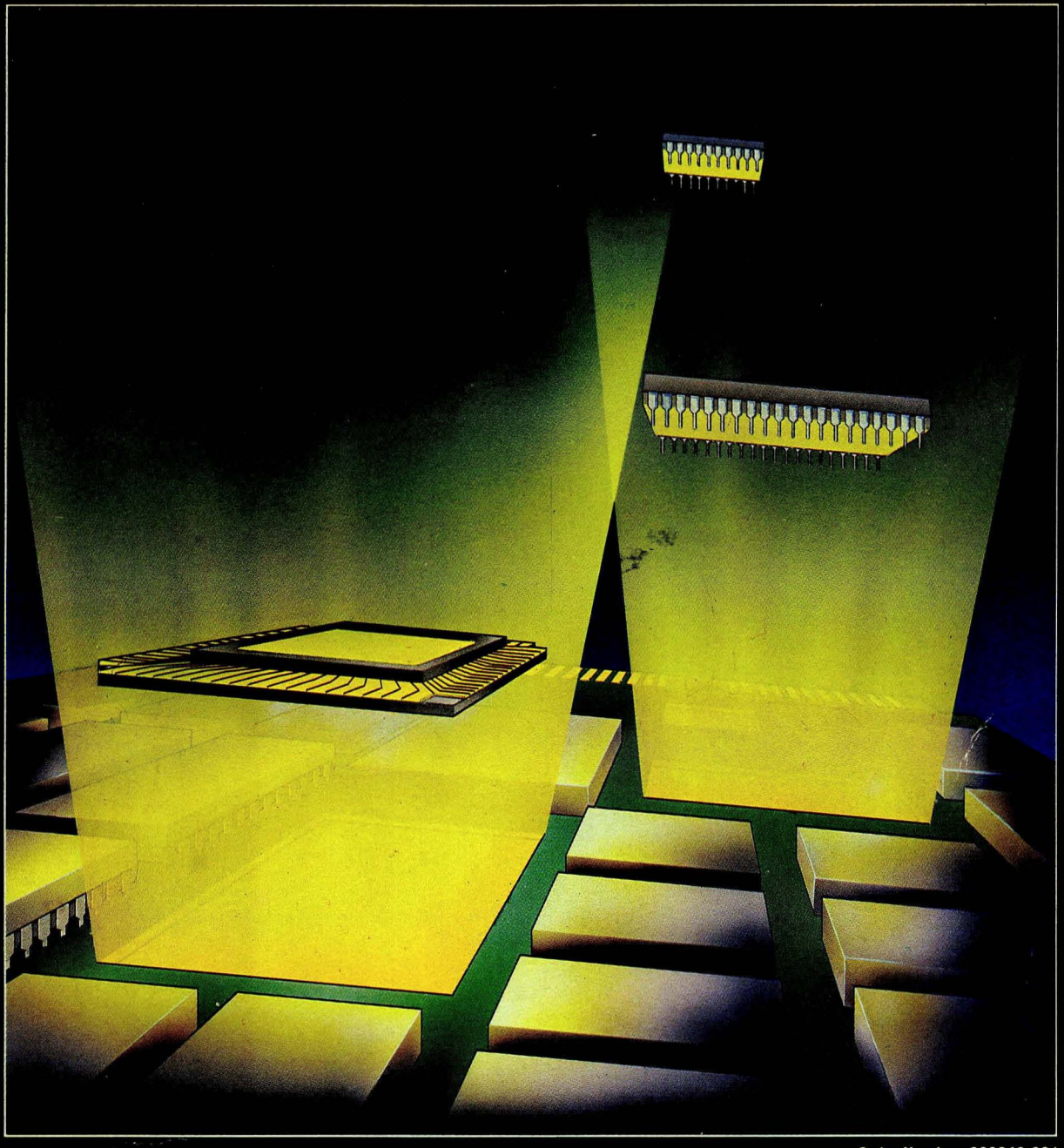
intel

Microsystem Components Handbook Volume I



Order Number: 230843-001

LITERATURE

In addition to the product line Handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846) provides an overview of Intel's complete product line and customer services.

Consult the INTEL LITERATURE GUIDE for a complete listing of Intel literature. TO ORDER literature in the United States, write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

1984 HANDBOOKS	U.S. PRICE*
Memory Components Handbook (Order No. 210830) Contains all application notes, article reprints, data sheets, and other design information on RAMs, DRAMs, EPROMs, E^2PROMs, Bubble Memories.	**$15.00**
Telecommunication Products Handbook (Order No. 230730) Contains all application notes, article reprints, and data sheets for telecommunication products.	**7.50**
Microcontroller Handbook (Order No. 210918) Contains all application notes, article reprints, data sheets, and design information for the MCS-48, MCS-51 and MCS-96 families.	**15.00**
Microsystem Components Handbook (Order No. 230843) Contains application notes, article reprints, data sheets, technical papers for microprocessors and peripherals. (2 Volumes) (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc. Consult the Literature Guide for prices and order numbers.)	**20.00**
Military Handbook (Order No. 210461) Contains complete data sheets for all military products. Information on Leadless Chip Carriers and on Quality Assurance is also included.	**10.00**
Development Systems Handbook (Order No. 210940) Contains data sheets on development systems and software, support options, and design kits.	**10.00**
OEM Systems Handbook (Order No. 210941) Contains all data sheets, application notes, and article reprints for OEM boards and systems.	**15.00**
Software Handbook (Order No. 230786) Contains all data sheets, applications notes, and article reprints available directly from Intel, as well as 3rd Party software.	**10.00**

*** Prices are for the U.S. only.**

intel®

MICROSYSTEM COMPONENTS HANDBOOK

VOLUME 1

1984

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BITBUS, COMMputer, CREDIT, Data Pipeline, GENIUS, i, ↑, ICE, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMMX, Insite, Intel, int_el, int_elBOS, Intelevision, $int_eligent$ Identifier, $int_eligent$ Programming, Intellec, Intellink, iOSP, iPDS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, Plug-A-Bubble, PROMPT, Promware, QUEST, QUEX, Ripplemode, RMX/80, RUPI, Seamless, SOLO, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

*CP/M-86 is a Trademark of Digital Research, Inc.

CHAPTER 5

iAPX 432 MICROMAINFRAME™

DATA SHEETS

CHAPTER 6

MEMORY CONTROLLERS

APPLICATION NOTES

ARTICLE REPRINTS

TECHNICAL PAPERS

DATA SHEETS

USERS MANUAL

— VOLUME 2 —

SUPPORT PERIPHERALS

APPLICATION NOTES

DATA SHEETS

FLOPPY DISK CONTROLLERS

APPLICATION NOTES

DATA SHEETS

HARD DISK CONTROLLERS

DATA SHEETS

Overview 1

INTRODUCTION

Intel microprocessors and peripherals provide a complete solution in increasingly complex application environments. Quite often, a single peripheral device will replace anywhere from 20 to 100 TTL devices (and the associated design time that goes with them).

Built-in functions and a standard Intel microprocessor/peripheral interface deliver very real *time* and *performance* advantages to the designer of microprocessor-based systems.

REDUCED TIME TO MARKET

When you can purchase an off-the-shelf solution that replaces a number of discrete devices, you're also replacing all the design, testing, and debug *time* that goes with them.

INCREASED RELIABILITY

At Intel, the rate of failure for devices is carefully tracked. Reliability is a tangible goal, and today we're measuring field failures in terms of *parts per million!* That translates to higher reliability for your product, reduced downtime, and reduced repair costs. And as more and more functions are integrated on a single VLSI device, the resulting system requires less power, produces less heat, and requires fewer mechanical connections—again resulting in greater system reliability.

LOWER PRODUCT COST

By minimizing design time, increasing reliability, and replacing numerous parts, microprocessor and peripheral solutions can contribute dramatically to a lower product cost.

HIGHER SYSTEM PERFORMANCE

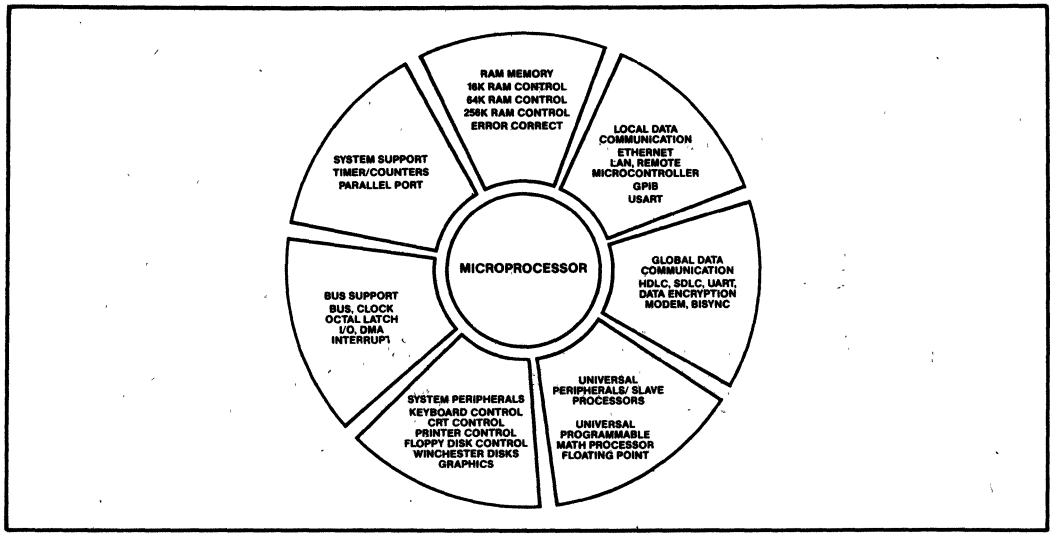
Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tommorrow's) microprocessor-based applications. For example, the iAPX 286 CPU, with its on-chip memory management and protection, offers the highest performance for multitasking, multiuser systems.

HOW TO USE THE GUIDE

The following application guide illustrates the range of microprocessors and peripherals that can be used for the applications in the vertical column on the left. The peripherals are grouped by the I/O function they control: CRT, datacommunication, universal (user programmable), mass storage, dynamic RAM's, and CPU/bus support.

An "X" in a horizontal application row indicates a potential peripheral or CPU, depending upon the features desired. For example, a conversational terminal could use either of the three display controllers, depending upon features like the number of characters per row or font capability. A "Y" indicates a likely candidate, for example, the 8272A Floppy Disk Controller in a small business computer.

The Intel microprocessor and peripherals family provides a broad range of time-saving, high performance solutions.



POTENTIAL CANDIDATE X—TYPICAL CANDIDATE Y

APPLICATION	μPROCESSOR					DISPLAY			DATACOMM									UPI	DISKS		DRAM CONTROL				SUPPORT		
	8088	8086	188	186	286	8275/76	82720	82730/731	8251A	8256	8273	8274	8291A/92/93	82530	82538	82586/501	8044/8744	8042/8742	8272A	82062	8203	8206	8207	8208	8254	8255	8231A
PERIPHERALS																											
Printers	X		X				X	X	X	X			X				X	X							X	X	
Plotters	X	X	X	X					X				X				X	X								X	
Keyboards									X								X	X								X	
MASS STORAGE																											
Hard Disk	X	X	Y																	X						X	
Mini Winchester	X		Y																	Y							
Tape			X														X	X								X	
Cassette																		X								X	
Floppy/Mini																			Y								
COMMUNICATIONS																											
PBX			X	X	Y					X	X	X		X			X				X	X	X	X	X	X	
LANS	X	X	X	X								X		X	X	Y	X								X		
Modems									X			X		X			X	X									
Bisync									X			X			X										X		
SDLC/HDLC											X	X		X			X								X		
Serial Back Plane											X	X		X		X	X										
Central Office		X		X	Y					X	X	X		X			X					X	X	X	X		
Network Control		Y		X	Y							X		X	X	X	X								X		
OFFICE/BUS																											
Copier/FAX	X		X							X			X			X	X	X									
Word Processor	X	X	Y	Y	Y		X	Y	X	X							X		Y		X	X	X	X	Y	Y	
Typewriter			X														X	X								Y	
Elect. Mail		X	X	X			X	Y				X		X	X	X											
Transaction System		Y		X	X				X	X							X								Y	X	
Data Entry	X	X	X	X		X	X	X	X	X							X	X						X	Y		
COMPUTERS																											
SM Bus Computer		X	Y	X	X	X	Y	Y	X	X		X		X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	X
PC	Y	X	Y	X	X	X	Y	Y	X	X		X	X	X	X	X	X	X	Y	Y	X	X	X	X	Y	Y	
Portable PC			X	X					X	X								X	Y		Y			X	Y	Y	
Home Computer	X	X	X	X	X		X		X	X								X	Y		X			X	Y	Y	

POTENTIAL CANDIDATE X—TYPICAL CANDIDATE Y (CONTINUED)

APPLICATION	μPROCESSOR					DISPLAY			DATACOMM								UPI	DISKS		DRAM CONTROL				SUPPORT			
	8088	8086	188	186	286	8275/76	82720	82730/731	8251A	8256	8273	8274	8291A/92/93	82530	82538	82586/501	8044/8744	8042/8742	8272A	82062	8203	8206	8207	8208	8254	8255	8231A	
TERMINALS																												
Conversational			X			X	X	X	X	X	X																	
Graphics CRT		Y		Y	Y		Y	Y	X	X		X		X		X			X	X	X	X	X	X	Y	Y		
Editing	X	X	X	X	X		Y	Y	X	Y		X		X		X			X	X								
Intelligent	X	X	Y	Y				Y	X	X		X		X	X	X			X	X			X	X	Y	Y	X	
Videotex	X	X	X	X			X	X	X																			
Printing, Laser, Impact	X	X	X	X			X	X	X							X	X	X							Y	Y		
Portable	X	X	Y				X	X	X												X							
INDUSTRIAL AUTO																												
Robotics			Y	Y	X							X		X		Y	X											
Network	X	X	X	X	X						X	X		X	X	X	Y	X										
Num Control		X	X	X	X		Y				X	X		X		X	X											
Process Control	X	X	X	X	X		Y				X	X		X		X	Y	X			X			X	X	X		
Instrumentation	X	X	X	X			Y						X					X										
Aviation/Navig		X	X	X	X													X							X	X	X	
INDUST/DATA ACQ																												
Laboratory Instr	X	X	X	X	X								X				Y	X										
Source Data	X		X														Y									X		
Auto Test	X	X	X	X	Y																							
Medical	X	X	X	X	X		Y									X	Y	X								X	X	
Test Instr	X	X	X	X	X							X		X				X										
Security		X	X	X											X		Y	X								X		
COMMERCIAL DATA PROCESSING																												
POS Terminal		X	X	X		X	X	X	X	X		X		X		X	X	X										
Financial Transfer		X	X	X		X	X	X	X	X					X		Y							X		X		
Automatic Teller		X	X	X		X	X	X	X	X							Y	X										
Document Processing	X	X	X	X	X	X	X	X	X	X							Y							X				
WORKSTATIONS																												
Office	X	X	X	X	X		Y	Y	X	X		X		X	X	X	X	X	Y	X	X							
Engineering	X	X	X	X	X		Y	Y	X	X		X		X	X	X		X	Y	Y	X	X	X	X	X	Y	X	
CAD		X	X	X	Y		Y	Y	X			X		X		X		X	X	X	X							
MINI MAINFRAME																												
Processor & Control Store		X		Y	Y	Y						X		X										X				
Database Subsys		X		Y	X	X									X							X	X					
I/O Subsystem			Y	Y	Y							X		X			X											
Comm. Subsystem		X		Y	Y				X			X		X	X		X						X					

MCS®-80/85 Microprocessors

2

Microprocessors Section

8080A/8080A-1/8080A-2
8-BIT N-CHANNEL MICROPROCESSOR

- **TTL Drive Capability**
- **2 μs (– 1:1.3 μs, – 2:1.5 μs) Instruction Cycle**
- **Powerful Problem Solving Instruction Set**
- **6 General Purpose Registers and an Accumulator**
- **16-Bit Program Counter for Directly Addressing up to 64K Bytes of Memory**
- **16-Bit Stack Pointer and Stack Manipulation Instructions for Rapid Switching of the Program Environment**
- **Decimal, Binary, and Double Precision Arithmetic**
- **Ability to Provide Priority Vectored Interrupts**
- **512 Directly Addressed I/O Ports**
- **Available in EXPRESS**
  - **Standard Temperature Range**

The Intel® 8080A is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers the user a high performance solution to control and processing applications.

The 8080A contains 6 8-bit general purpose working registers and an accumulator. The 6 general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset 4 testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, program counter, and all of the 6 general purpose registers. The 16-bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bidirectional data busses are used to facilitate easy interface to memory and I/O. Signals to control the interface to memory and I/O are provided directly by the 8080A. Ultimate control of the address and data busses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data busses into a high impedance state. This permits OR-tying these busses with other controlling devices for (DMA) direct memory access or multi-processor operation.

NOTE:
The 8080A is functionally and electrically compatible with the Intel® 8080.

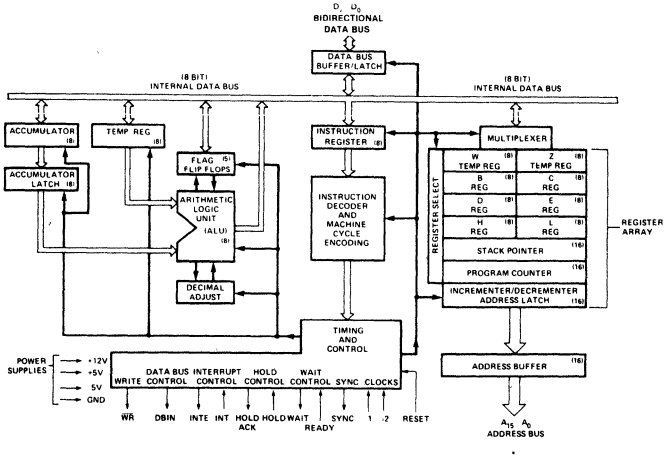


Figure 1. Block Diagram

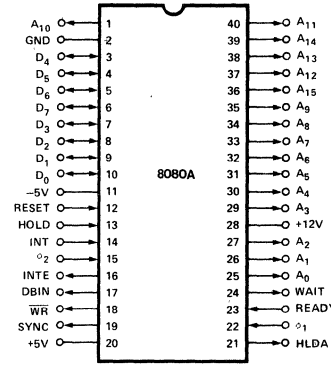


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A_{15}-A_0	O	**Address Bus:** The address bus provides the address to memory (up to 64K 8-bit words) or denotes the I/O device number for up to 256 input and 256 output devices. A_0 is the least significant address bit.
D_7-D_0	I/O	**Data Bus:** The data bus provides bi-directional communication betweeen the CPU, memory, and I/O devices for instructions and data transfers. Also, during the first clock cycle of each machine cycle, the 8080A outputs a status word on the data bus that describes the current machine cycle. D_0 is the least significant bit.
SYNC	O	**Synchronizing Signal:** The SYNC pin provides a signal to indicate the beginning of each machine cycle.
DBIN	O	**Data Bus In:** The DBIN signal indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the 8080A data bus from memory or I/O.
READY	I	**Ready:** The READY signal indicates to the 8080A that valid memory or input data is available on the 8080A data bus. This signal is used to synchronize the CPU with slower memory or I/O devices. If after sending an address out the 8080A does not receive a READY input, the 8080A will enter a WAIT state for as long as the READY line is low. READY can also be used to single step the CPU
WAIT	O	**Wait:** The WAIT signal acknowledges that the CPU is in a WAIT state.
$\overline{WR}$	O	**Write:** The $\overline{WR}$ signal is used for memory WRITE or I/O output control. The data on the data bus is stable while the $\overline{WR}$ signal is active low ($\overline{WR}$ = 0).
HOLD	I	**Hold:** The HOLD signal requests the CPU to enter the HOLD state. The HOLD state allows an external device to gain control of the 8080A address and data bus as soon as the 8080A has completed its use of these busses for the current machine cycle. It is recognized under the following conditions: • the CPU is in the HALT state. • the CPU is in the T2 or TW state and the READY signal is active. As a result of entering the HOLD state the CPU ADDRESS BUS (A_{15}-A_0) and DATA BUS (D_7-D_0) will be in their high impedance state. The CPU acknowledges its state with the HOLD ACKNOWLEDGE (HLDA) pin.
HLDA	O	**Hold Acknowledge:** The HLDA signal appears in response to the HOLD signal and indicates that the data and address bus will go to the high impedance state. The HLDA signal begins at: • T3 for READ memory or input. • The Clock Period following T3 for WRITE memory or OUTPUT operation. In either case, the HLDA signal appears after the rising edge of ϕ_2.
INTE	O	**Interrupt Enable:** Indicates the content of the internal interrupt enable flip/flop. This flip/flop may be set or reset by the Enable and Disable Interrupt instructions and inhibits interrupts from being accepted by the CPU when it is reset. It is automatically reset (disabling further interrupts) at time T1 of the instruction fetch cycle (M1) when an interrupt is accepted and is also reset by the RESET signal.
INT	I	**Interrupt Request:** The CPU recognizes an interrupt request on this line at the end of the current instruction or while halted. If the CPU is in the HOLD state or if the Interrupt Enable flip/flop is reset it will not honor the request.
RESET[1]	I	**Reset:** While the RESET signal is activated, the content of the program counter is cleared. After RESET, the program will start at location 0 in memory. The INTE and HLDA flip/flops are also reset. Note that the flags, accumulator, stack pointer, and registers are not cleared.
V_{SS}		**Ground:** Reference.
V_{DD}		**Power:** +12 ±5% Volts.
V_{CC}		**Power:** +5 ±5% Volts
V_{BB}		**Power:** −5 ±5% Volts.
ϕ_1, ϕ_2		**Clock Phases:** 2 externally supplied clock phases. (non TTL compatible)

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to +70° C
Storage Temperature . . . . . -65°C to +150°C
All Input or Output Voltages
With Respect to V_{BB} . . . . . -0.3V to +20V
V_{CC}, V_{DD} and V_{SS} With Respect to V_{BB} -0.3V to +20V
Power Dissipation 1.5W

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

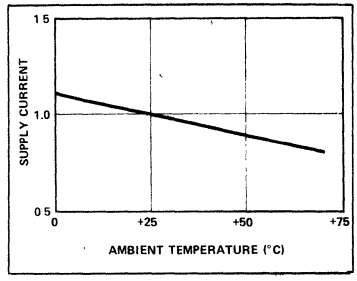
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} =0V; unless otherwise noted)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IHC}	Clock Input High Voltage	9.0		$V_{DD}+1$	V	
V_{IL}	Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
V_{IH}	Input High Voltage	3.3		$V_{CC}+1$	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.9mA on all outputs,
V_{OH}	Output High Voltage	3.7			V	I_{OH} = −150μA.
$I_{DD\,(AV)}$	Avg. Power Supply Current (V_{DD})		40	70	mA	Operation
$I_{CC\,(AV)}$	Avg. Power Supply Current (V_{CC})		60	80	mA	T_{CY} = .48 μsec
$I_{BB\,(AV)}$	Avg. Power Supply Current (V_{BB})		.01	1	mA	
I_{IL}	Input Leakage			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{CL}	Clock Leakage			±10	μA	$V_{SS} \leq V_{CLOCK} \leq V_{DD}$
I_{DL} [2]	Data Bus Leakage in Input Mode			-100 -2.0	μA mA	$V_{SS} \leq V_{IN} \leq V_{SS}+0.8V$ $V_{SS}+0.8V \leq V_{IN} \leq V_{CC}$
I_{FL}	Address and Data Bus Leakage During HOLD			+10 -100	μA	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS} + 0.45V$

CAPACITANCE (T_A = 25°C, V_{CC} = V_{DD} = V_{SS} = 0V, V_{BB} = −5V)

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
C_ϕ	Clock Capacitance	17	25	pf	f_c = 1 MHz
C_{IN}	Input Capacitance	6	10	pf	Unmeasured Pins
C_{OUT}	Output Capacitance	10	20	pf	Returned to V_{SS}

NOTES:
1. The RESET signal must be active for a minimum of 3 clock cycles.
2. ΔI supply / ΔT_A = -0.45%/°C.



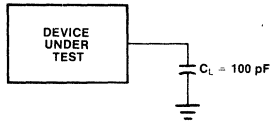
Typical Supply Current vs. Temperature, Normalized[3]

A.C. CHARACTERISTICS (8080A)

(T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} = 0V; unless otherwise noted)

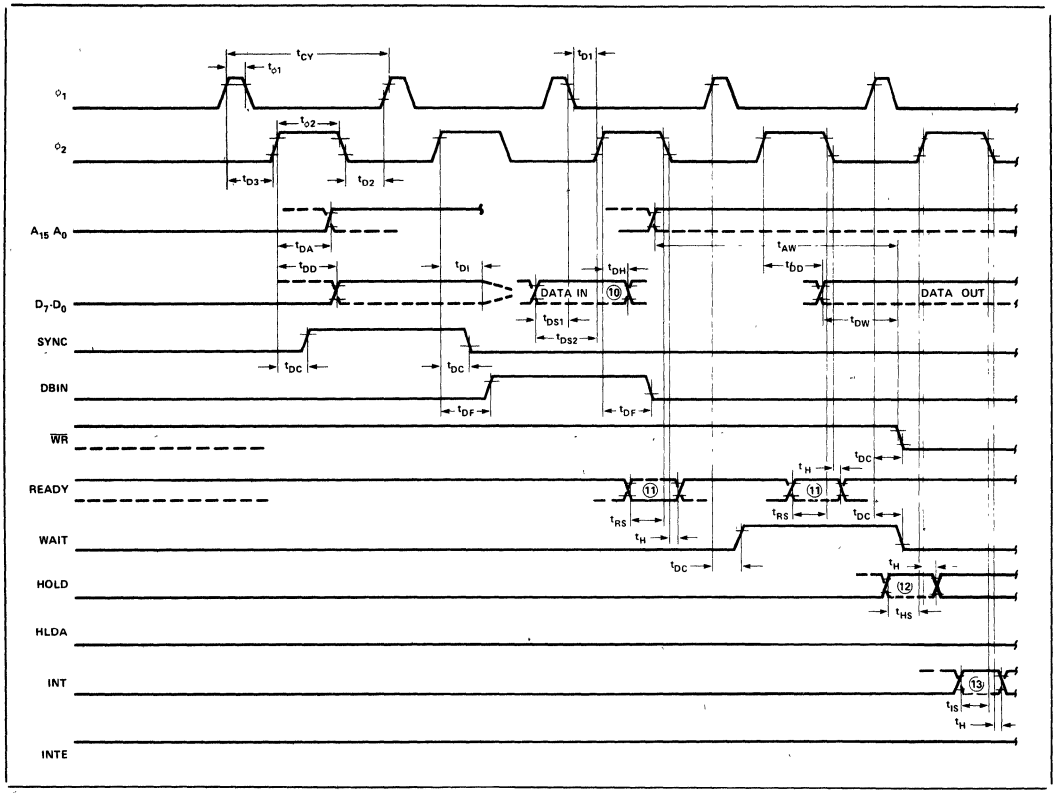
Symbol	Parameter	Min.	Max.	-1 Min.	-1 Max.	-2 Min.	-2 Max.	Unit	Test Condition
t_{CY}[3]	Clock Period	0.48	2.0	0.32	2.0	0.38	2.0	μsec	
t_r, t_f	Clock Rise and Fall Time	0	50	0	25	0	50	nsec	
$t_{\phi 1}$	ϕ_1 Pulse Width	60		50		60		nsec	
$t_{\phi 2}$	ϕ_2 Pulse Width	220		145		175		nsec	
t_{D1}	Delay ϕ_1 to ϕ_2	0		0		0		nsec	
t_{D2}	Delay ϕ_2 to ϕ_1	70		60		70		nsec	
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	80		60		70		nsec	
t_{DA}	Address Output Delay From ϕ_2		200		150		175	nsec	C_L = 100 pF
t_{DD}	Data Output Delay From ϕ_2		220		180		200	nsec	C_L = 100 pF
t_{DC}	Signal Output Delay From ϕ_1 or ϕ_2 (SYNC, WR, WAIT, HLDA)		120		110		120	nsec	C_L = 50 pF
t_{DF}	DBIN Delay From ϕ_2	25	140	25	130	25	140	nsec	C_L = 50 pF
t_{DI}[1]	Delay for Input Bus to Enter Input Mode		t_{DF}		t_{DF}		t_{DF}	nsec	
t_{DS1}	Data Setup Time During ϕ_1 and DBIN	30		10		20		nsec	
t_{DS2}	Data Setup Time to ϕ_2 During DBIN	150		120		130		nsec	
t_{DH}[1]	Data Holt time From ϕ_2 During DBIN	[1]		[1]		[1]		nsec	
t_{IE}	INTE Output Delay From ϕ_2		200		200		200	nsec	C_L = 50 pF
t_{RS}	READY Setup Time During ϕ_2	120		90		90		nsec	
t_{HS}	HOLD Setup Time to ϕ_2	140		120		120		nsec	
t_{IS}	INT Setup Time During ϕ_2	120		100		100		nsec	
t_H	Hold Time From ϕ_2 (READY, INT, HOLD)	0		0		0		nsec	
t_{FD}	Delay to Float During Hold (Address and Data Bus)		120		120		120	nsec	
t_{AW}	Address Stable Prior to WR	[5]		[5]		[5]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{DW}	Output Data Stable Prior to WR	[6]		[6]		[6]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{WD}	Output Data Stable From WR	[7]		[7]		[7]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{WA}	Address Stable From WR	[7]		[7]		[7]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{HF}	HLDA to Float Delay	[8]		[8]		[8]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{WF}	WR to Float Delay	[9]		[9]		[9]		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN
t_{AH}	Address Hold Time After DBIN During HLDA	−20		−20		−20		nsec	C_L = 100 pF: Address, Data C_L = 50 pF: $\overline{WR}$,HLDA,DBIN

A.C. TESTING LOAD CIRCUIT



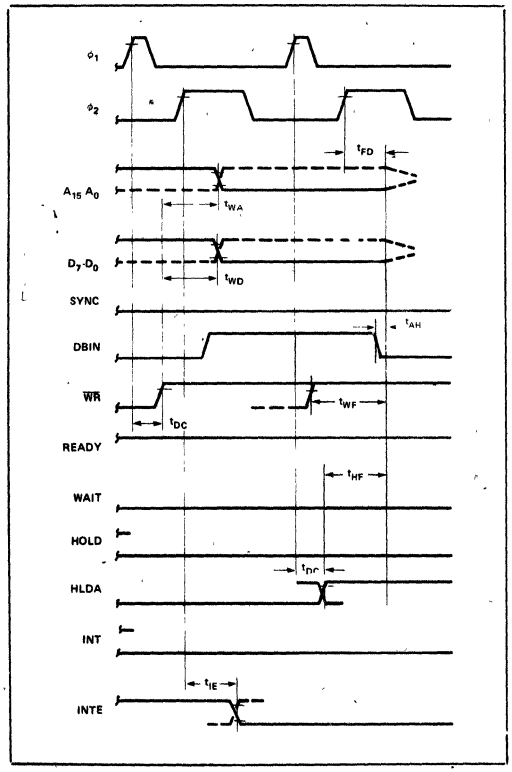
C_L 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS



NOTE:
Timing measurements are made at the following reference voltages: CLOCK "1" = 8.0V, "0" = 1.0V; INPUTS "1" = 3.3V, "0" = 0.8V; OUTPUTS "1" = 2.0V, "0" = 0.8V.

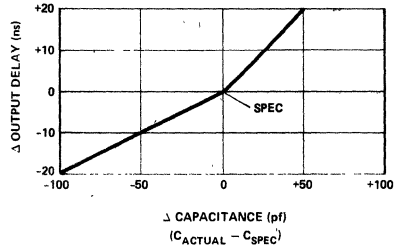
WAVEFORMS (Continued)



NOTES: (Parenthesis gives -1, -2 specifications, respectively)

1. Data input should be enabled with DBIN status. No bus conflict can then occur and data hold time is assured. t_{DH} = 50 ns or t_{DF}, whichever is less.
2. $t_{CY} = t_{D3} + t_{r\phi 2} + t_{\phi 2} + t_{f\phi 2} + t_{D2} + t_{r\phi 1} \geq 480$ ns (− 1:320 ns, − 2:380 ns).

TYPICAL Δ OUTPUT DELAY VS. Δ CAPACITANCE



3. The following are relevant when interfacing the 8080A to devices having V_{IH} = 3.3V:
   a) Maximum output rise time from .8V to 3.3V = 100ns @ C_L = SPEC.
   b) Output delay when measured to 3.0V = SPEC +60ns @ C_L = SPEC.
   c) If C_L = SPEC, add .6ns/pF if $C_L > C_{SPEC}$, subtract .3ns/pF (from modified delay) if $C_L < C_{SPEC}$.
4. $t_{AW} = 2t_{CY} - t_{D3} - t_{r\phi 2} - 140$ ns (− 1:110 ns, − 2:130 ns).
5. $t_{DW} = t_{CY} - t_{D3} - t_{r\phi 2} - 170$ ns (− 1:150 ns, − 2:170 ns).
6. If not HLDA, $t_{WD} = t_{WA} = t_{D3} + t_{r\phi 2} + 10$ ns. If HLDA, $t_{WD} = t_{WA} = t_{WF}$.
7. $t_{HF} = t_{D3} + t_{r\phi 2} - 50$ ns).
8. $t_{WF} = t_{D3} + t_{r\phi 2} - 10$ns.
9. Data in must be stable for this period during DBIN T_3. Both t_{DS1} and t_{DS2} must be satisfied.
10. Ready signal must be stable for this period during T_2 or T_W. (Must be externally synchronized.)
11. Hold signal must be stable for this period during T_2 or T_W when entering hold mode, and during T_3, T_4, T_5 and T_{WH} when in hold mode. (External synchronization is not required.)
12. Interrupt signal must be stable during this period of the last clock cycle of any instruction in order to be recognized on the following instruction. (External synchronization is not required.)
13. This timing diagram shows timing relationships only; it does not represent any specific machine cycle.

INSTRUCTION SET

The accumulator group instructions include arithmetic and logical operators with direct, indirect, and immediate addressing modes.

Move, load, and store instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using direct, indirect, and immediate addressing modes.

The ability to branch to different portions of the program is provided with jump, jump conditional, and computed jumps. Also the ability to call to and return from subroutines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as stack manipulation and double add instructions extend both the arithmetic and interrupt handling capability of the 8080A. The ability to increment and decrement memory, the six general registers and the accumulator is provided as well as extended increment and decrement instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to rotate the accumulator left or right through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080A instruction set.

The following special instruction group completes the 8080A instruction set: the NOP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

Data and Instruction Formats

Data in the 8080A is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

One Byte Instructions

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	OP CODE

TYPICAL INSTRUCTIONS

Register to register, memory reference, arithmetic or logical, rotate, return, push, pop, enable or disable Interrupt instructions

Two Byte Instructions

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	OP CODE
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	OPERAND

Immediate mode or I/O instructions

Three Byte Instructions

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	OP CODE
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	LOW ADDRESS OR OPERAND 1
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	HIGH ADDRESS OR OPERAND 2

Jump, call or direct load and store instructions

For the 8080A a logic "1" is defined as a high level and a logic "0" is defined as a low level.

Table 2. Instruction Set Summary

Mnemonic	Instruction Code [1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description	Clock Cycles [2]
MOVE, LOAD, AND STORE										
MOVr1,r2	0	1	D	D	D	S	S	S	Move register to register	5
MOV M,r	0	1	1	1	0	S	S	S	Move register to memory	7
MOV r,M	0	1	D	D	D	1	1	0	Move memory to register	7
MVI r	0	0	D	D	D	1	1	0	Move immediate register	7
MVI M	0	0	1	1	0	1	1	0	Move immediate memory	10
LXI B	0	0	0	0	0	0	0	1	Load immediate register Pair B & C	10
LXI D	0	0	0	1	0	0	0	1	Load immediate register Pair D & E	10
LXI H	0	0	1	0	0	0	0	1	Load immediate register Pair H & L	10
STAX B	0	0	0	0	0	0	1	0	Store A indirect	7
STAX D	0	0	0	1	0	0	1	0	Store A indirect	7
LDAX B	0	0	0	0	1	0	1	0	Load A indirect	7
LDAX D	0	0	0	1	1	0	1	0	Load A indirect	7
STA	0	0	1	1	0	0	1	0	Store A direct	13
LDA	0	0	1	1	1	0	1	0	Load A direct	13
SHLD	0	0	1	0	0	0	1	0	Store H & L direct	16
LHLD	0	0	1	0	1	0	1	0	Load H & L direct	16
XCHG	1	1	1	0	1	0	1	1	Exchange D & E, H & L Registers	4
STACK OPS										
PUSH B	1	1	0	0	0	1	0	1	Push register Pair B & C on stack	11
PUSH D	1	1	0	1	0	1	0	1	Push register Pair D & E on stack	11
PUSH H	1	1	1	0	0	1	0	1	Push register Pair H & L on stack	11
PUSH PSW	1	1	1	1	0	1	0	1	Push A and Flags on stack	11
POP B	1	1	0	0	0	0	0	1	Pop register Pair B & C off stack	10
POP D	1	1	0	1	0	0	0	1	Pop register Pair D & E off stack	10
POP H	1	1	1	0	0	0	0	1	Pop register Pair H & L off stack	10
POP PSW	1	1	1	1	0	0	0	1	Pop A and Flags off stack	10
XTHL	1	1	1	0	0	0	1	1	Exchange top of stack, H & L	18
SPHL	1	1	1	1	1	0	0	1	H & L to stack pointer	5
LXI SP	0	0	1	1	0	0	0	1	Load immediate stack pointer	10
INX SP	0	0	1	1	0	0	1	1	Increment stack pointer	5
DCX SP	0	0	1	1	1	0	1	1	Decrement stack pointer	5
JUMP										
JMP	1	1	0	0	0	0	1	1	Jump unconditional	10
JC	1	1	0	1	1	0	1	0	Jump on carry	10
JNC	1	1	0	1	0	0	1	0	Jump on no carry	10
JZ	1	1	0	0	1	0	1	0	Jump on zero	10
JNZ	1	1	0	0	0	0	1	0	Jump on no zero	10
JP	1	1	1	1	0	0	1	0	Jump on positive	10
JM	1	1	1	1	1	0	1	0	Jump on minus	10
JPE	1	1	1	0	1	0	1	0	Jump on parity even	10
JPO	1	1	1	0	0	0	1	0	Jump on parity odd	10
PCHL	1	1	1	0	1	0	0	1	H & L to program counter	5
CALL										
CALL	1	1	0	0	1	1	0	1	Call unconditional	17
CC	1	1	0	1	1	1	0	0	Call on carry	11/17
CNC	1	1	0	1	0	1	0	0	Call on no carry	11/17
CZ	1	1	0	0	1	1	0	0	Call on zero	11/17
CNZ	1	1	0	0	0	1	0	0	Call on no zero	11/17
CP	1	1	1	1	0	1	0	0	Call on positive	11/17
CM	1	1	1	1	1	1	0	0	Call on minus	11/17
CPE	1	1	1	0	1	1	0	0	Call on parity even	11/17
CPO	1	1	1	0	0	1	0	0	Call on parity odd	11/17
RETURN										
RET	1	1	0	0	1	0	0	1	Return	10
RC	1	1	0	1	1	0	0	0	Return on carry	5/11
RNC	1	1	0	1	0	0	0	0	Return on no carry	5/11
RZ	1	1	0	0	1	0	0	0	Return on zero	5/11
RNZ	1	1	0	0	0	0	0	0	Return on no zero	5/11
RP	1	1	1	1	0	0	0	0	Return on positive	5/11
RM	1	1	1	1	1	0	0	0	Return on minus	5/11
RPE	1	1	1	0	1	0	0	0	Return on parity even	5/11
RPO	1	1	1	0	0	0	0	0	Return on parity odd	5/11
RESTART										
RST	1	1	A	A	A	1	1	1	Restart	11
INCREMENT AND DECREMENT										
INR r	0	0	D	D	D	1	0	0	Increment register	5
DCR r	0	0	D	D	D	1	0	1	Decrement register	5
INR M	0	0	1	1	0	1	0	0	Increment memory	10
DCR M	0	0	1	1	0	1	0	1	Decrement memory	10
INX B	0	0	0	0	0	0	1	1	Increment B & C registers	5
INX D	0	0	0	1	0	0	1	1	Increment D & E registers	5
INX H	0	0	1	0	0	0	1	1	Increment H & L registers	5
DCX B	0	0	0	0	1	0	1	1	Decrement B & C	5
DCX D	0	0	0	1	1	0	1	1	Decrement D & E	5
DCX H	0	0	1	0	1	0	1	1	Decrement H & L	5
ADD										
ADD r	1	0	0	0	0	S	S	S	Add register to A	4
ADC r	1	0	0	0	1	S	S	S	Add register to A with carry	4
ADD M	1	0	0	0	0	1	1	0	Add memory to A	7
ADC M	1	0	0	0	1	1	1	0	Add memory to A with carry	7
ADI	1	1	0	0	0	1	1	0	Add immediate to A	7
ACI	1	1	0	0	1	1	1	0	Add immediate to A with carry	7
DAD B	0	0	0	0	1	0	0	1	Add B & C to H & L	10
DAD D	0	0	0	1	1	0	0	1	Add D & E to H & L	10
DAD H	0	0	1	0	1	0	0	1	Add H & L to H & L	10
DAD SP	0	0	1	1	1	0	0	1	Add stack pointer to H & L	10

Summary of Processor Instructions (Cont.)

Mnemonic	Instruction Code [1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description	Clock Cycles [2]
SUBTRACT										
SUB r	1	0	0	1	0	S	S	S	Subtract register from A	4
SBB r	1	0	0	1	1	S	S	S	Subtract register from A with borrow	4
SUB M	1	0	0	1	0	1	1	0	Subtract memory from A	7
SBB M	1	0	0	1	1	1	1	0	Subtract memory from A with borrow	7
SUI	1	1	0	1	0	1	1	0	Subtract immediate from A	7
SBI	1	1	0	1	1	1	1	0	Subtract immediate from A with borrow	7
LOGICAL										
ANA r	1	0	1	0	0	S	S	S	And register with A	4
XRA r	1	0	1	0	1	S	S	S	Exclusive Or register with A	4
ORA r	1	0	1	1	0	S	S	S	Or register with A	4
CMP r	1	0	1	1	1	S	S	S	Compare register with A	4
ANA M	1	0	1	0	0	1	1	0	And memory with A	7
XRA M	1	0	1	0	1	1	1	0	Exclusive Or memory with A	7
ORA M	1	0	1	1	0	1	1	0	Or memory with A	7
CMP M	1	0	1	1	1	1	1	0	Compare memory with A	7
ANI	1	1	1	0	0	1	1	0	And immediate with A	7
XRI	1	1	1	0	1	1	1	0	Exclusive Or immediate with A	7
ORI	1	1	1	1	0	1	1	0	Or immediate with A	7
CPI	1	1	1	1	1	1	1	0	Compare immediate with A	7

Mnemonic	Instruction Code [1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description	Clock Cycles [2]
ROTATE										
RLC	0	0	0	0	0	1	1	1	Rotate A left	4
RRC	0	0	0	0	1	1	1	1	Rotate A right	4
RAL	0	0	0	1	0	1	1	1	Rotate A left through carry	4
RAR	0	0	0	1	1	1	1	1	Rotate A right through carry	4
SPECIALS										
CMA	0	0	1	0	1	1	1	1	Complement A	4
STC	0	0	1	1	0	1	1	1	Set carry	4
CMC	0	0	1	1	1	1	1	1	Complement carry	4
DAA	0	0	1	0	0	1	1	1	Decimal adjust A	4
INPUT/OUTPUT										
IN	1	1	0	1	1	0	1	1	Input	10
OUT	1	1	0	1	0	0	1	1	Output	10
CONTROL										
EI	1	1	1	1	1	0	1	1	Enable Interrupts	4
DI	1	1	1	1	0	0	1	1	Disable Interrupt	4
NOP	0	0	0	0	0	0	0	0	No-operation	4
HLT	0	1	1	1	0	1	1	0	Halt	7

NOTES:

1. DDD or SSS: B=000, C=001, D=010, E=011, H=100, L=101, Memory=110, A=111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.

8085AH/8085AH-2/8085AH-1
8-BIT HMOS MICROPROCESSORS

- **Single +5V Power Supply with 10% Voltage Margins**
- **3 MHz, 5 MHz and 6 MHz Selections Available**
- **20% Lower Power Consumption than 8085A for 3 MHz and 5 MHz**
- **1.3 μs Instruction Cycle (8085AH); 0.8 μs (8085AH-2); 0.67 μs (8085AH-1)**
- **100% Compatible with 8085A**
- **100% Software Compatible with 8080A**
- **On-Chip Clock Generator (with External Crystal, LC or RC Network)**
- **On-Chip System Controller; Advanced Cycle Status Information Available for Large System Control**
- **Four Vectored Interrupt Inputs (One is Non-Maskable) Plus an 8080A-Compatible Interrupt**
- **Serial In/Serial Out Port**
- **Decimal, Binary and Double Precision Arithmetic**
- **Direct Addressing Capability to 64K Bytes of Memory**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8085AH is a complete 8 bit parallel Central Processing Unit (CPU) implemented in N-channel, depletion load, silicon gate technology (HMOS). Its instruction set is 100% software compatible with the 8080A microprocessor, and it is designed to improve the present 8080A's performance by higher system speed. Its high level of system integration allows a minimum system of three IC's [8085AH (CPU), 8156H (RAM/IO) and 8355/8755A (ROM/PROM/IO)] while maintaining total system expandability. The 8085AH-2 and 8085AH-1 are faster versions of the 8085AH.

The 8085AH incorporates all of the features that the 8224 (clock generator) and 8228 (system controller) provided for the 8080A, thereby offering a high level of system integration.

The 8085AH uses a multiplexed data bus. The address is split between the 8 bit address bus and the 8 bit data bus. The on-chip address latches of 8155H/8156H/8355/8755A memory products allow a direct interface with the 8085AH.

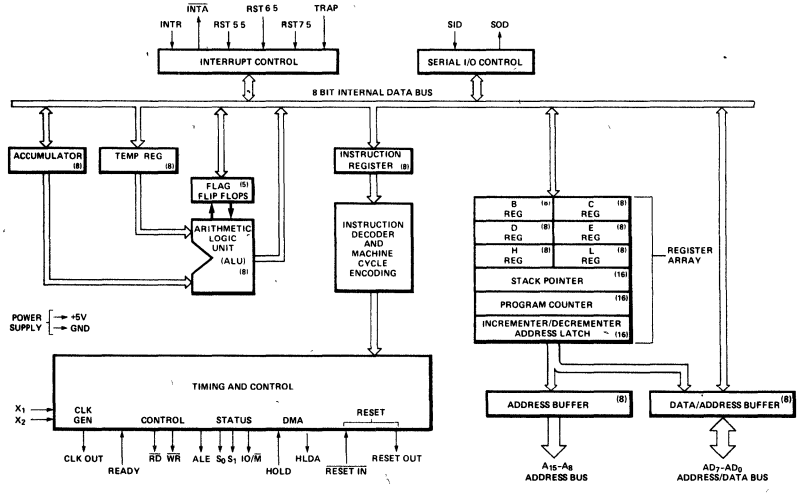


Figure 1. 8085AH CPU Functional Block Diagram

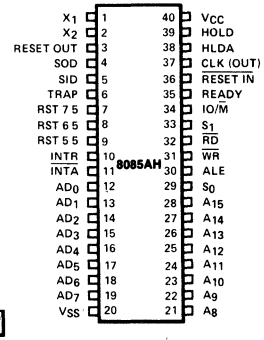


Figure 2. 8085AH Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A_8–A_{15}	O	**Address Bus:** The most significant 8 bits of the memory address or the 8 bits of the I/O address, 3-stated during Hold and Halt modes and during RESET.
AD_0–$_7$	I/O	**Multiplexed Address/Data Bus:** Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle (T state) of a machine cycle. It then becomes the data bus during the second and third clock cycles.
ALE	O	**Address Latch Enable:** It occurs during the first clock state of a machine cycle and enables the address to get latched into the on-chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. The falling edge of ALE can also be used to strobe the status information. ALE is never 3-stated.
S_0, S_1, and IO/$\overline{M}$	O	**Machine Cycle Status:** IO/$\overline{M}$ S_1 S_0 Status 0 0 1 Memory write 0 1 0 Memory read 1 0 1 I/O write 1 1 0 I/O read 0 1 1 Opcode fetch 1 1 1 Opcode fetch 1 1 1 Interrupt Acknowledge * 0 0 Halt * X X Hold * X X Reset * = 3-state (high impedance) X = unspecified S_1 can be used as an advanced R/$\overline{W}$ status. IO/$\overline{M}$, S_0 and S_1 become valid at the beginning of a machine cycle and remain stable throughout the cycle. The falling edge of ALE may be used to latch the state of these lines.
$\overline{RD}$	O	**Read Control:** A low level on $\overline{RD}$ indicates the selected memory or I/O device is to be read and that the Data Bus is available for the data transfer, 3-stated during Hold and Halt modes and during RESET.
$\overline{WR}$	O	**Write Control:** A low level on $\overline{WR}$ indicates the data on the Data Bus is to be written into the selected memory or I/O location. Data is set up at the trailing edge of $\overline{WR}$. 3-stated during Hold and Halt modes and during RESET.

Symbol	Type	Name and Function
READY	I	**Ready:** If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If READY is low, the cpu will wait an integral number of clock cycles for READY to go high before completing the read or write cycle. READY must conform to specified setup and hold times.
HOLD	I	**Hold:** Indicates that another master is requesting the use of the address and data buses. The cpu, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the Address, Data $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ lines are 3-stated.
HLDA	O	**Hold Acknowledge:** Indicates that the cpu has received the HOLD request and that it will relinquish the bus in the next clock cycle. HLDA goes low after the Hold request is removed. The cpu takes the bus one half clock cycle after HLDA goes low.
INTR	I	**Interrupt Request:** Is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of an instruction and during Hold and Halt states. If it is active, the Program Counter (PC) will be inhibited from incrementing and an $\overline{INTA}$ will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted
$\overline{INTA}$	O	**Interrupt Acknowledge:** Is used instead of (and has the same timing as) $\overline{RD}$ during the Instruction cycle after an INTR is accepted. It can be used to activate an 8259A Interrupt chip or some other interrupt port.
RST 5.5 RST 6.5 RST 7.5	I	**Restart Interrupts:** These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted The priority of these interrupts is ordered as shown in Table 2. These interrupts have a higher priority than INTR. In addition, they may be individually masked out using the SIM instruction.

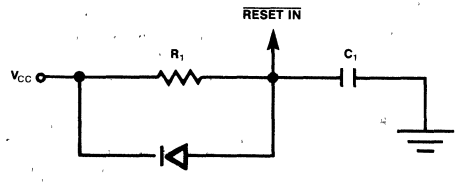
Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
TRAP	I	**Trap:** Trap interrupt is a non-maskable RESTART interrupt. It is recognized at the same time as INTR or RST 5.5-7.5. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt. (See Table 2)
$\overline{\text{RESET IN}}$	I	**Reset In:** Sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. The data and address buses and the control lines are 3-stated during RESET and because of the asynchronous nature of RESET, the processor's internal registers and flags may be altered by RESET with unpredictable results. $\overline{\text{RESET IN}}$ is a Schmitt-triggered input, allowing connection to an R-C network for power-on RESET delay (see Figure 3). Upon power-up, $\overline{\text{RESET IN}}$ must remain low for at least 10 ms after minimum V_{CC} has been reached. For proper reset operation after the power-up duration, $\overline{\text{RESET IN}}$ should be kept low a minimum of three clock periods. The CPU is held in the reset condition as long as $\overline{\text{RESET IN}}$ is applied.
RESET OUT	O	**Reset Out:** Reset Out indicates cpu is being reset. Can be used as a system reset. The signal is synchronized to the processor clock and lasts an integral number of clock periods.
X_1, X_2	I	**X_1 and X_2:** Are connected to a crystal, LC, or RC network to drive the internal clock generator. X_1 can also be an external clock input from a logic gate. The input frequency is divided by 2 to give the processor's internal operating frequency.
CLK	O	**Clock:** Clock output for use as a system clock. The period of CLK is twice the X_1, X_2 input period.
SID	I	**Serial Input Data Line:** The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.
SOD	O	**Serial Output Data Line:** The output SOD is set or reset as specified by the SIM instruction.
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Reference.

Table 2. Interrupt Priority, Restart Address, and Sensitivity

Name	Priority	Address Branched To (1) When Interrupt Occurs	Type Trigger
TRAP	1	24H	Rising edge AND high level until sampled.
RST 7.5	2	3CH	Rising edge (latched).
RST 6.5	3	34H	High level until sampled.
RST 5.5	4	2CH	High level until sampled.
INTR	5	See Note (2).	High level until sampled.

NOTES:
1. The processor pushes the PC on the stack before branching to the indicated address.
2. The address branched to depends on the instruction provided to the cpu when the interrupt is acknowledged.



TYPICAL POWER-ON RESET RC VALUES*
R_1 = 75 KΩ
C_1 = 1 μF

*VALUES MAY HAVE TO VARY DUE TO APPLIED POWER SUPPLY RAMP UP TIME.

Figure 3. Power-On Reset Circuit

FUNCTIONAL DESCRIPTION

The 8085AH is a complete 8-bit parallel central processor. It is designed with N-channel, depletion load, silicon gate technology (HMOS), and requires a single +5 volt supply. Its basic clock speed is 3 MHz (8085AH), 5 MHz (8085AH-2), or 6 MHz (8085AH-1), thus improving on the present 8080A's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU (8085AH), a RAM/IO (8156H), and a ROM or EPROM/IO chip (8355 or 8755A).

The 8085AH has twelve addressable 8-bit registers. Four of them can function only as two 16-bit register pairs. Six others can be used interchangeably as 8-bit registers or as 16-bit register pairs. The 8085AH register set is as follows:

Mnemonic	Register	Contents
ACC or A	Accumulator	8 bits
PC	Program Counter	16-bit address
BC,DE,HL	General-Purpose Registers; data pointer (HL)	8 bits x 6 or 16 bits x 3
SP	Stack Pointer	16-bit address
Flags or F	Flag Register	5 flags (8-bit space)

The 8085AH uses a multiplexed Data Bus. The address is split between the higher 8-bit Address Bus and the lower 8-bit Address/Data Bus. During the first T state (clock cycle) of a machine cycle the low order address is sent out on the Address/Data bus. These lower 8 bits may be latched externally by the Address Latch Enable signal (ALE). During the rest of the machine cycle the data bus is used for memory or I/O data.

The 8085AH provides $\overline{RD}$, $\overline{WR}$, S_0, S_1, and IO/$\overline{M}$ signals for bus control. An Interrupt Acknowledge signal ($\overline{INTA}$) is also provided. HOLD and all Interrupts are synchronized with the processor's internal clock. The 8085AH also provides Serial Input Data (SID) and Serial Output Data (SOD) lines for simple serial interface.

In addition to these features, the 8085AH has three maskable, vector interrupt pins, one nonmaskable TRAP interrupt, and a bus vectored interrupt, INTR.

INTERRUPT AND SERIAL I/O

The 8085AH has 5 interrupt inputs: INTR, RST 5.5, RST 6.5, RST 7.5, and TRAP. INTR is identical in function to the 8080A INT. Each of the three RESTART inputs, 5.5, 6.5, and 7.5, has a programmable mask. TRAP is also a RESTART interrupt but it is nonmaskable.

The three maskable interrupts cause the internal execution of RESTART (saving the program counter in the stack and branching to the RESTART address) if the interrupts are enabled and if the interrupt mask is not set. The nonmaskable TRAP causes the internal execution of a RESTART vector independent of the state of the interrupt enable or masks. (See Table 2.)

There are two different types of inputs in the restart interrupts. RST 5.5 and RST 6.5 are *high level-sensitive* like INTR (and INT on the 8080) and are recognized with the same timing as INTR. RST 7.5 is *rising edge-sensitive.*

For RST 7.5, only a pulse is required to set an internal flip-flop which generates the internal interrupt request (a normally high level signal with a low going pulse is recommended for highest system noise immunity). The RST 7.5 request flip-flop remains set until the request is serviced. Then it is reset automatically. This flip-flop may also be reset by using the SIM instruction or by issuing a $\overline{RESET\ IN}$ to the 8085AH. The RST 7.5 internal flip-flop will be set by a pulse on the RST 7.5 pin even when the RST 7.5 interrupt is masked out.

The status of the three RST interrupt masks can only be affected by the SIM instruction and $\overline{RESET\ IN}$. (See SIM, Chapter 5 of the MCS-80/85 User's Manual.)

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending as follows: TRAP—highest priority, RST 7.5, RST 6.5, RST 5.5, INTR—lowest priority. This priority scheme does not take into account the priority of a routine that was started by a higher priority interrupt. RST 5.5 can interrupt an RST 7.5 routine if the interrupts are re-enabled before the end of the RST 7.5 routine.

The TRAP interrupt is useful for catastrophic events such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the highest priority. It is not affected by any flag or mask. The TRAP input is both *edge and level sensitive*. The TRAP input must go high and remain high until it is acknowledged. It will not be recognized again until it goes low, then high again. This avoids any false triggering due to noise or logic glitches. Figure 4 illustrates the TRAP interrupt request circuitry within the 8085AH. Note that the servicing of any interrupt (TRAP, RST 7.5, RST 6.5, RST 5.5, INTR) disables all future interrupts (except TRAPs) until an EI instruction is executed.

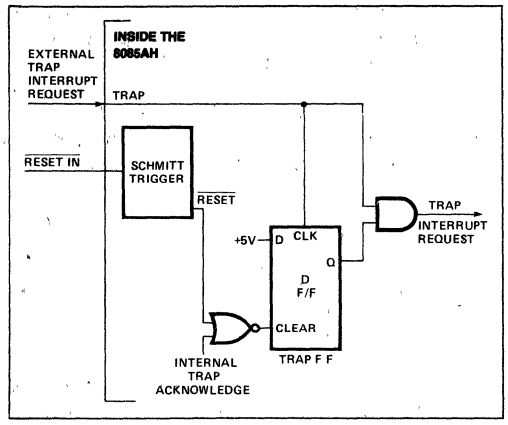


Figure 4. TRAP and RESET IN Circuit

The TRAP interrupt is special in that it disables interrupts, but preserves the previous interrupt enable status. Performing the first RIM instruction following a TRAP interrupt allows you to determine whether interrupts were enabled or disabled prior to the TRAP. All subsequent RIM instructions provide current interrupt enable status. Performing a RIM instruction following INTR, or RST 5.5–7.5 will provide current Interrupt Enable status, revealing that Interrupts are disabled. See the description of the RIM instruction in the MCS-80/85 Family User's Manual.

The serial I/O system is also controlled by the RIM and SIM instructions. SID is read by RIM, and SIM sets the SOD data.

DRIVING THE X_1 AND X_2 INPUTS

You may drive the clock inputs of the 8085AH, 8085AH-2, or 8085AH-1 with a crystal, an LC tuned circuit, an RC network, or an external clock source. The crystal frequency must be at least 1 MHz, and must be twice the desired internal clock frequency; hence, the 8085AH is operated with a 6 MHz crystal (for 3 MHz clock), the 8085AH-2 operated with a 10 MHz crystal (for 5 MHz clock), and the 8085AH-1 can be operated with a 12 MHz crystal (for 6 MHz clock). If a crystal is used, it must have the following characteristics:

Parallel resonance at twice the clock frequency desired
C_L (load capacitance) $\leq$ 30 pF
C_S (shunt capacitance) $\leq$ 7 pF
R_S (equivalent shunt resistance) $\leq$ 75 Ohms
Drive level: 10 mW
Frequency tolerance: $\pm$.005% (suggested)

Note the use of the 20 pF capacitor between X_2 and ground. This capacitor is required with crystal frequencies below 4 MHz to assure oscillator startup at the correct frequency. A parallel-resonant LC circuit may be used as the frequency-determining network for the 8085AH, providing that its frequency tolerance of approximately $\pm$10% is acceptable. The components are chosen from the formula:

$$f = \frac{1}{2\pi\sqrt{L(C_{ext} + C_{int})}}$$

To minimize variations in frequency, it is recommended that you choose a value for C_{ext} that is at least twice that of C_{int}, or 30 pF. The use of an LC circuit is not recommended for frequencies higher than approximately 5 MHz.

An RC circuit may be used as the frequency-determining network for the 8085AH if maintaining a precise clock frequency is of no importance. Variations in the on-chip timing generation can cause a wide variation in frequency when using the RC mode. Its advantage is its low component cost. The driving frequency generated by the circuit shown is approximately 3 MHz. It is not recommended that frequencies greatly higher or lower than this be attempted.

Figure 5 shows the recommended clock driver circuits. Note in D and E that pullup resistors are required to assure that the high level voltage of the input is at least 4V and maximum low level voltage of 0.8V.

For driving frequencies up to and including 6 MHz you may supply the driving signal to X_1 and leave X_2 open-circuited (Figure 5D). If the driving frequency is from 6 MHz to 12 MHz, stability of the clock generator will be improved by driving both X_1 and X_2 with a push-pull source (Figure 5E). To prevent self-oscillation of the 8085AH, be sure that X_2 is not coupled back to X_1 through the driving circuit.

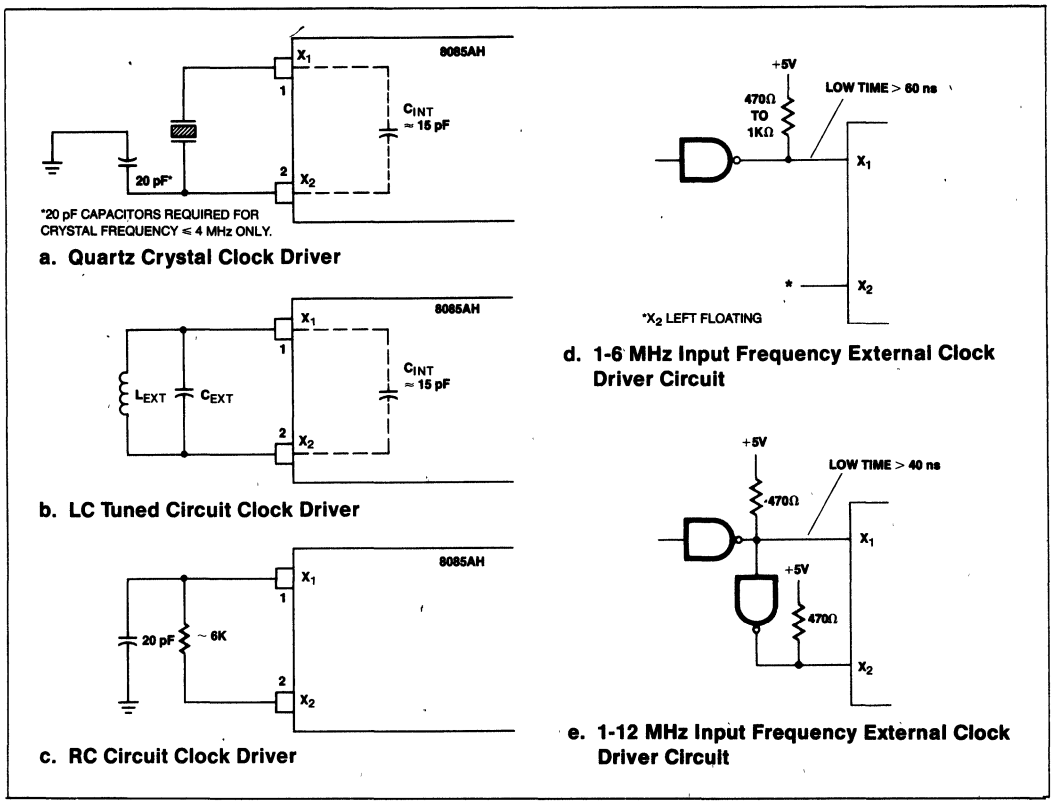


Figure 5. Clock Driver Circuits

GENERATING AN 8085AH WAIT STATE

If your system requirements are such that slow memories or peripheral devices are being used, the circuit shown in Figure 6 may be used to insert one WAIT state in each 8085AH machine cycle.

The D flip-flops should be chosen so that
- CLK is rising edge-triggered
- CLEAR is low-level active.

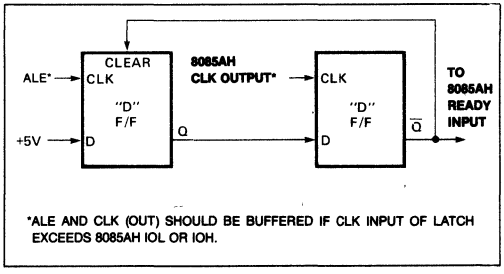


Figure 6. Generation of a Wait State for 8085AH CPU

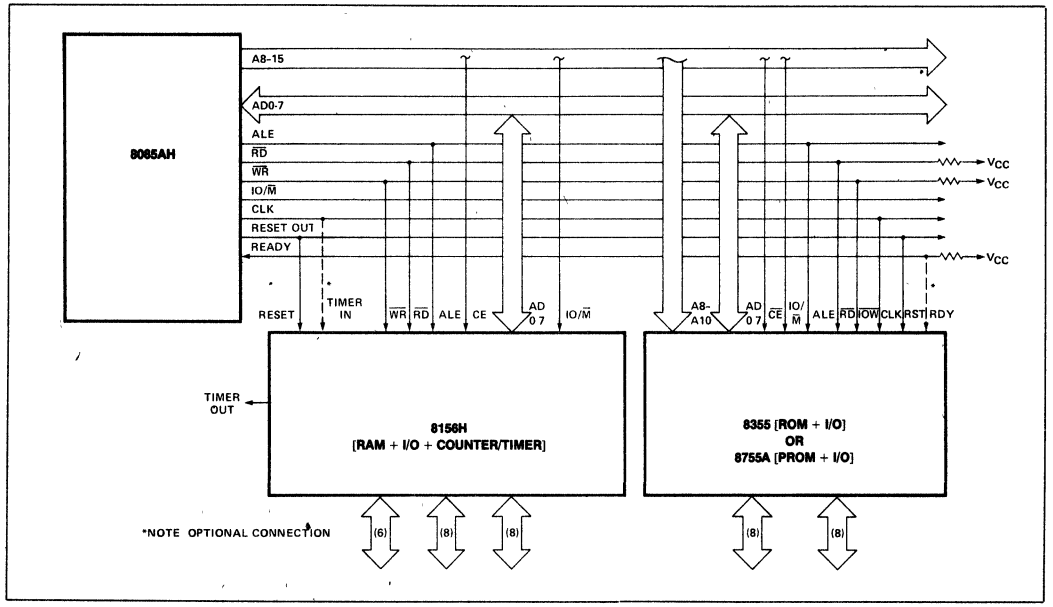


Figure 8. MCS-85® Minimum System (Memory Mapped I/O)

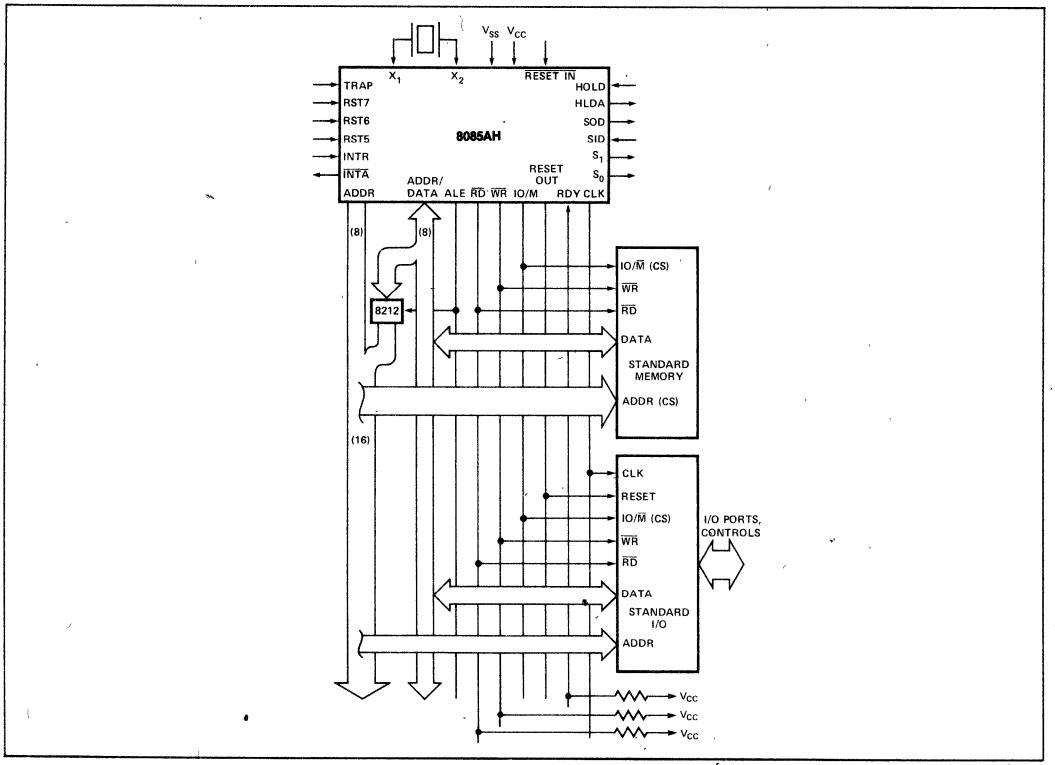


Figure 9. MCS-85® System (Using Standard Memories)

As in the 8080, the READY line is used to extend the read and write pulse lengths so that the 8085AH can be used with slow memory. HOLD causes the CPU to relinquish the bus when it is through with it by floating the Address and Data Buses.

SYSTEM INTERFACE

The 8085AH family includes memory components, which are directly compatible to the 8085AH CPU. For example, a system consisting of the three chips, 8085AH, 8156H, and 8355 will have the following features:

- 2K Bytes ROM
- 256 Bytes RAM
- 1 Timer/Counter
- 4 8-bit I/O Ports
- 1 6-bit I/O Port
- 4 Interrupt Levels
- Serial In/Serial Out Ports

This minimum system, using the standard I/O technique is as shown in Figure 7.

In addition to standard I/O, the memory mapped I/O offers an efficient I/O addressing technique. With this technique, an area of memory address space is assigned for I/O address, thereby, using the memory address for I/O manipulation. Figure 8 shows the system configuration of Memory Mapped I/O using 8085AH.

The 8085AH CPU can also interface with the standard memory that does *not* have the multiplexed address/data bus. It will require a simple 8212 (8-bit latch) as shown in Figure 9.

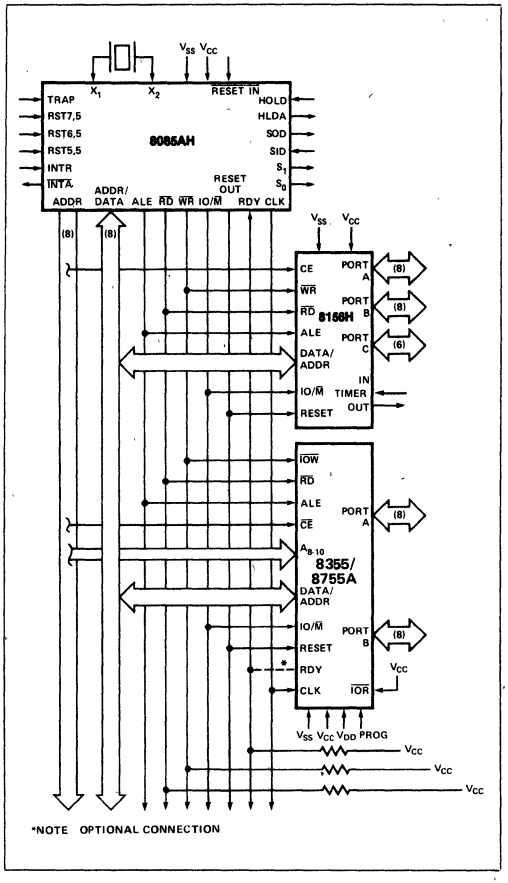


Figure 7. 8085AH Minimum System (Standard I/O Technique)

BASIC SYSTEM TIMING

The 8085AH has a multiplexed Data Bus. ALE is used as a strobe to sample the lower 8-bits of address on the Data Bus. Figure 10 shows an instruction fetch, memory read and I/O write cycle (as would occur during processing of the OUT instruction). Note that during the I/O write and read cycle that the I/O port address is copied on both the upper and lower half of the address.

There are seven possible types of machine cycles. Which of these seven takes place is defined by the status of the three status lines ($IO/\overline{M}$, S_1, S_0) and the three control signals ($\overline{RD}$, $\overline{WR}$, and $\overline{INTA}$). (See Table 3.) The status lines can be used as advanced controls (for device selection, for example), since they become active at the T_1 state, at the outset of each machine cycle. Control lines $\overline{RD}$ and $\overline{WR}$ become active later, at the time when the transfer of data is to take place, so are used as command lines.

A machine cycle normally consists of three T states, with the exception of OPCODE FETCH, which normally has either four or six T states (unless WAIT or HOLD states are forced by the receipt of $\overline{READY}$ or HOLD inputs). Any T state must be one of ten possible states, shown in Table 4.

Table 3. 8085AH Machine Cycle Chart

MACHINE CYCLE		STATUS			CONTROL		
		$IO/\overline{M}$	S1	S0*	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
OPCODE FETCH	(OF)	0	1	1	0	1	1
MEMORY READ	(MR)	0	1	0	0	1	1
MEMORY WRITE	(MW)	0	0	1	1	0	1
I/O READ	(IOR)	1	1	0	0	1	1
I/O WRITE	(IOW)	1	0	1	1	0	1
ACKNOWLEDGE OF INTR	(INA)	1	1	1	1	1	0
BUS IDLE	(BI) DAD	0	1	0	1	1	1
	ACK OF RST,TRAP	1	1	1	1	1	1
	HALT	TS	0	0	TS	TS	1

Table 4. 8085AH Machine State Chart

Machine State	Status & Buses				Control		
	S1,S0	$IO/\overline{M}$	A_8-A_{15}	AD_0-AD_7	$\overline{RD}$,$\overline{WR}$	$\overline{INTA}$	ALE
T_1	X	X	X	X	1	1	1*
T_2	X	X	X	X	X	X	0
T_{WAIT}	X	X	X	X	X	X	0
T_3	X	X	X	X	X	X	0
T_4	1	0†	X	TS	1	1	0
T_5	1	0†	X	TS	1	1	0
T_6	1	0†	X	TS	1	1	0
T_{RESET}	X	TS	TS	TS	TS	1	0
T_{HALT}	0	TS	TS	TS	TS	1	0
T_{HOLD}	X	TS	TS	TS	TS	1	0

0 = Logic "0" TS = High Impedance
1 = Logic "1" X = Unspecified

* ALE not generated during 2nd and 3rd machine cycles of DAD instruction

† IO/M = 1 during T_4-T_6 of INA machine cycle

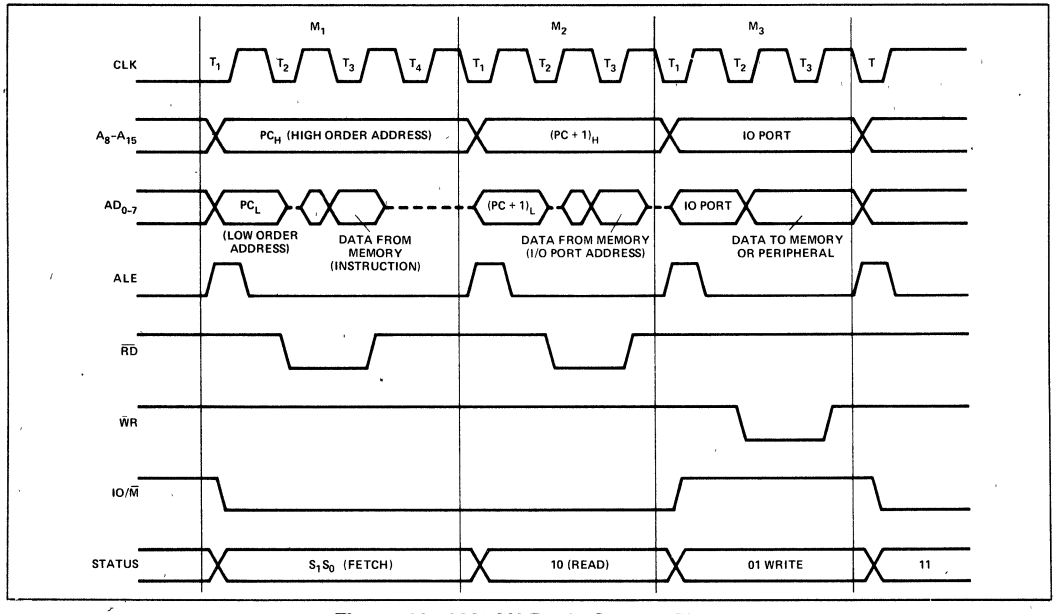


Figure 10. 8085AH Basic System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

8085AH, 8085AH-2: (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} =0V; unless otherwise specified)*
8085AH-1: (T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V; unless otherwise specified)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
I_{CC}	Power Supply Current		135	mA	8085AH, 8085AH-2
			200	mA	8085AH-1 (Preliminary)
I_{IL}	Input Leakage		±10	μA	$0 \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{ILR}	Input Low Level, RESET	−0.5	+0.8	V	
V_{IHR}	Input High Level, RESET	2.4	V_{CC} +0.5	V	
V_{HY}	Hysteresis, RESET	0.15		V	

A.C. CHARACTERISTICS

8085AH, 8085AH-2: (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = OV)*
8085AH-1: (T_A = 0°C to 70°C, V_{CC} = 5V ±5%, V_{SS} = 0V)

Symbol	Parameter	8085AH[2] (Final)		8085AH-2[2] (Final)		8085AH-1 (Preliminary)		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{CYC}	CLK Cycle Period	320	2000	200	2000	167	2000	ns
t_1	CLK Low Time (Standard CLK Loading)	80		40		20		ns
t_2	CLK High Time (Standard CLK Loading)	120		70		50		ns
t_r, t_f	CLK Rise and Fall Time		30		30		30	ns
t_{XKR}	X_1 Rising to CLK Rising	20	120	20	100	20	100	ns
t_{XKF}	X_1 Rising to CLK Falling	20	150	20	110	20	110	ns
t_{AC}	$A_{8\text{-}15}$ Valid to Leading Edge of Control[1]	270		115		70		ns
t_{ACL}	$A_{0\text{-}7}$ Valid to Leading Edge of Control	240		115		60		ns
t_{AD}	$A_{0\text{-}15}$ Valid to Valid Data In		575		350		225	ns
t_{AFR}	Address Float After Leading Edge of $\overline{READ}$ ($\overline{INTA}$)		0		0		0	ns
t_{AL}	$A_{8\text{-}15}$ Valid Before Trailing Edge of ALE [1]	115		50		25		ns

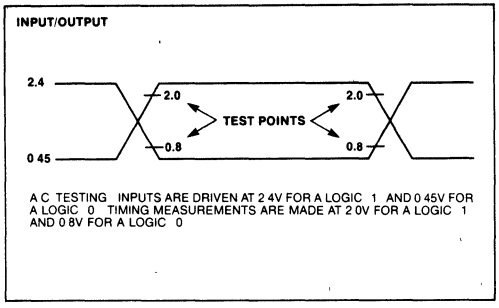
Note: For Extended Temperature EXPRESS use M8085AH Electricals Parameters.

A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8085AH[2] (Final)		8085AH-2[2] (Final)		8085AH-1 (Preliminary)		Units
		Min.	Max.	Min.	Max.	Min.	Max.	
t_{ALL}	A_{0-7} Valid Before Trailing Edge of ALE	90		50		25		ns
t_{ARY}	READY Valid from Address Valid		220		100		40	ns
t_{CA}	Address (A_{8-15}) Valid After Control	120		60		30		ns
t_{CC}	Width of Control Low ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$) Edge of ALE	400		230		150		ns
t_{CL}	Trailing Edge of Control to Leading Edge of ALE	50		25		0		ns
t_{DW}	Data Valid to Trailing Edge of $\overline{WRITE}$	420		230		140		ns
t_{HABE}	HLDA to Bus Enable		210		150		150	ns
t_{HABF}	Bus Float After HLDA		210		150		150	ns
t_{HACK}	HLDA Valid to Trailing Edge of CLK	110		40		0		ns
t_{HDH}	HOLD Hold Time	0		0		0		ns
t_{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		120		120		ns
t_{INH}	INTR Hold Time	0		0		0		ns
t_{INS}	INTR, RST, and TRAP Setup Time to Falling Edge of CLK	160		150		150		ns
t_{LA}	Address Hold Time After ALE	100		50		20		ns
t_{LC}	Trailing Edge of ALE to Leading Edge of Control	130		60		25		ns
t_{LCK}	ALE Low During CLK High	100		50		15		ns
t_{LDR}	ALE to Valid Data During Read		460		270		175	ns
t_{LDW}	ALE to Valid Data During Write		200		140		110	ns
t_{LL}	ALE Width	140		80		50		ns
t_{LRY}	ALE to READY Stable		110		30		10	ns
t_{RAE}	Trailing Edge of $\overline{READ}$ to Re-Enabling of Address	150		90		50		ns
t_{RD}	$\overline{READ}$ (or $\overline{INTA}$) to Valid Data		300		150		75	ns
t_{RV}	Control Trailing Edge to Leading Edge of Next Control	400		220		160		ns
t_{RDH}	Data Hold Time After $\overline{READ}$ $\overline{INTA}$	0		0		0		ns
t_{RYH}	READY Hold Time	0		0		5		ns
t_{RYS}	READY Setup Time to Leading Edge of CLK	110		100		100		ns
t_{WD}	Data Valid After Trailing Edge of $\overline{WRITE}$	100		60		30		ns
t_{WDL}	LEADING Edge of $\overline{WRITE}$ to Data Valid		40		20		30	ns

NOTES:
1. A_8–A_{15} address Specs apply IO/$\overline{M}$, S_0, and S_1 except A_8–A_{15} are undefined during T_4–T_6 of OF cycle whereas IO/$\overline{M}$, S_0, and S_1 are stable.
2. *Test Conditions:* t_{CYC} = 320 ns (8085AH)/200 ns (8085AH-2);/ 167 ns (8085AH-1); C_L = 150 pF.
3. For all output timing where $C_L \neq 150$ pF use the following correction factors:
   25 pF ≤ C_L < 150 pF: −0.10 ns/pF
   150 pF < C_L ≤ 300 pF: +0.30 ns/pF
4. Output timings are measured with purely capacitive load.
5. To calculate timing specifications at other values of t_{CYC} use Table 5.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT

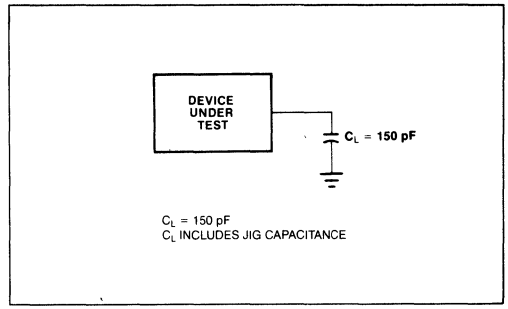


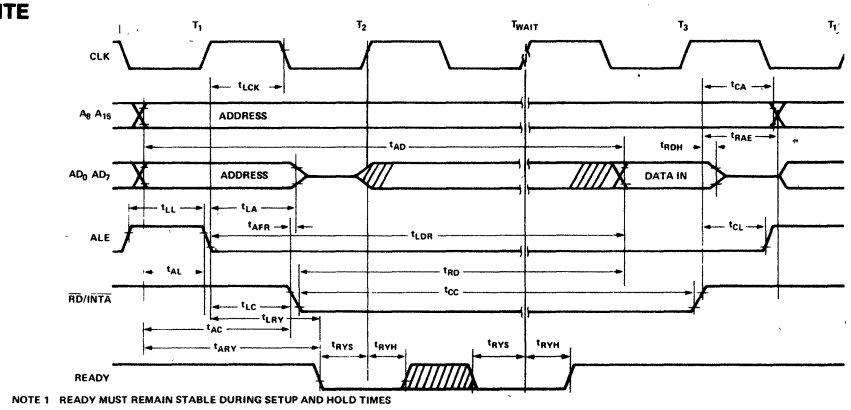
Table 5. Bus Timing Specification as a T_{CYC} Dependent

Symbol	8085AH	8085AH-2	8085AH-1	
t_{AL}	(1/2) T − 45	(1/2) T − 50	(1/2) T − 58	Minimum
t_{LA}	(1/2) T − 60	(1/2) T − 50	(1/2) T − 63	Minimum
t_{LL}	(1/2) T − 20	(1/2) T − 20	(1/2) T − 33	Minimum
t_{LCK}	(1/2) T − 60	(1/2) T − 50	(1/2) T − 68	Minimum
t_{LC}	(1/2) T − 30	(1/2) T − 40	(1/2) T − 58	Minimum
t_{AD}	(5/2 + N) T − 225	(5/2 + N) T − 150	(5/2 + N) T − 192	Maximum
t_{RD}	(3/2 + N) T − 180	(3/2 + N) T − 150	(3/2 + N) T − 175	Maximum
t_{RAE}	(1/2) T − 10	(1/2) T − 10	(1/2) T − 33	Minimum
t_{CA}	(1/2) T − 40	(1/2) T − 40	(1/2) T − 53	Minimum
t_{DW}	(3/2 + N) T − 60	(3/2 + N) T − 70	(3/2 + N) T − 110	Minimum
t_{WD}	(1/2) T − 60	(1/2) T − 40	(1/2) T − 53	Minimum
t_{CC}	(3/2 + N) T − 80	(3/2 + N) T − 70	(3/2 + N) T − 100	Minimum
t_{CL}	(1/2) T − 110	(1/2) T − 75	(1/2) T − 83	Minimum
t_{ARY}	(3/2) T − 260	(3/2) T − 200	(3/2) T − 210	Maximum
t_{HACK}	(1/2) T − 50	(1/2) T − 60	(1/2) T − 83	Minimum
t_{HABF}	(1/2) T + 50	(1/2) T + 50	(1/2) T + 67	Maximum
t_{HABE}	(1/2) T + 50	(1/2) T + 50	(1/2) T + 67	Maximum
t_{AC}	(2/2) T − 50	(2/2) T − 85	(2/2) T − 97	Minimum
t_1	(1/2) T − 80	(1/2) T − 60	(1/2) T − 63	Minimum
t_2	(1/2) T − 40	(1/2) T − 30	(1/2) T − 33	Minimum
t_{RV}	(3/2) T − 80	(3/2) T − 80	(3/2) T − 90	Minimum
t_{LDR}	(4/2) T − 180	(4/2) T − 130	(4/2) T − 159	Maximum

NOTE: N is equal to the total WAIT states. T = t_{CYC}.

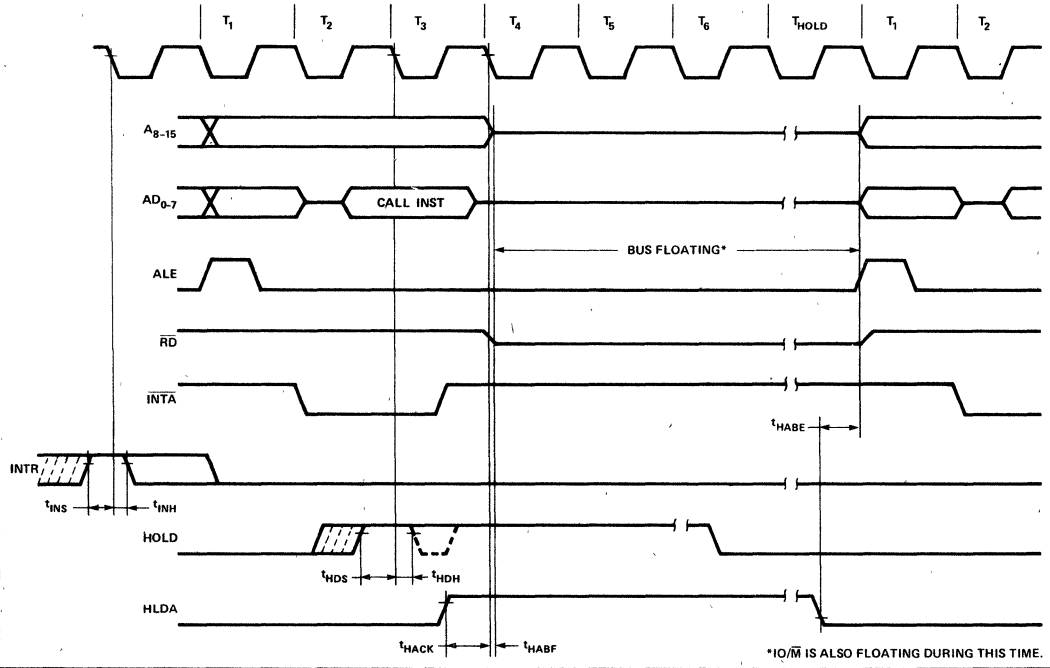
WAVEFORMS (Continued)

READ OPERATION WITH WAIT CYCLE (TYPICAL) — SAME READY TIMING APPLIES TO WRITE



NOTE 1 READY MUST REMAIN STABLE DURING SETUP AND HOLD TIMES

INTERRUPT AND HOLD



*IO/M IS ALSO FLOATING DURING THIS TIME.

Table 6. Instruction Set Summary

Mnemonic	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
MOVE, LOAD, AND STORE									
MOVr1 r2	0	1	D	D	D	S	S	S	Move register to register
MOV M,r	0	1	1	1	0	S	S	S	Move register to memory
MOV r,M	0	1	D	D	D	1	1	0	Move memory to register
MVI r	0	0	D	D	D	1	1	0	Move immediate register
MVI M	0	0	1	1	0	1	1	0	Move immediate memory
LXI B	0	0	0	0	0	0	0	1	Load immediate register Pair B & C
LXI D	0	0	0	1	0	0	0	1	Load immediate register Pair D & E
LXI H	0	0	1	0	0	0	0	1	Load immediate register Pair H & L
STAX B	0	0	0	0	0	0	1	0	Store A indirect
STAX D	0	0	0	1	0	0	1	0	Store A indirect
LDAX B	0	0	0	0	1	0	1	0	Load A indirect
LDAX D	0	0	0	1	1	0	1	0	Load A indirect
STA	0	0	1	1	0	0	1	0	Store A direct
LDA	0	0	1	1	1	0	1	0	Load A direct
SHLD	0	0	1	0	0	0	1	0	Store H & L direct
LHLD	0	0	1	0	1	0	1	0	Load H & L direct
XCHG	1	1	1	0	1	0	1	1	Exchange D & E, H & L Registers
STACK OPS									
PUSH B	1	1	0	0	0	1	0	1	Push register Pair B & C on stack
PUSH D	1	1	0	1	0	1	0	1	Push register Pair D & E on stack
PUSH H	1	1	1	0	0	1	0	1	Push register Pair H & L on stack
PUSH PSW	1	1	1	1	0	1	0	1	Push A and Flags on stack
POP B	1	1	0	0	0	0	0	1	Pop register Pair B & C off stack
POP D	1	1	0	1	0	0	0	1	Pop register Pair D & E off stack
POP H	1	1	1	0	0	0	0	1	Pop register Pair H & L off stack
POP PSW	1	1	1	1	0	0	0	1	Pop A and Flags off stack
XTHL	1	1	1	0	0	0	1	1	Exchange top of stack, H & L
SPHL	1	1	1	1	1	0	0	1	H & L to stack pointer
LXI SP	0	0	1	1	0	0	0	1	Load immediate stack pointer
INX SP	0	0	1	1	0	0	1	1	Increment stack pointer
DCX SP	0	0	1	1	1	0	1	1	Decrement stack pointer
JUMP									
JMP	1	1	0	0	0	0	1	1	Jump unconditional
JC	1	1	0	1	1	0	1	0	Jump on carry
JNC	1	1	0	1	0	0	1	0	Jump on no carry
JZ	1	1	0	0	1	0	1	0	Jump on zero
JNZ	1	1	0	0	0	0	1	0	Jump on no zero
JP	1	1	1	1	0	0	1	0	Jump on positive
JM	1	1	1	1	1	0	1	0	Jump on minus
JPE	1	1	1	0	1	0	1	0	Jump on parity even
JPO	1	1	1	0	0	0	1	0	Jump on parity odd
PCHL	1	1	1	0	1	0	0	1	H & L to program counter
CALL									
CALL	1	1	0	0	1	1	0	1	Call unconditional
CC	1	1	0	1	1	1	0	0	Call on carry
CNC	1	1	0	1	0	1	0	0	Call on no carry

Mnemonic	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
CZ	1	1	0	0	1	1	0	0	Call on zero
CNZ	1	1	0	0	0	1	0	0	Call on no zero
CP	1	1	1	1	0	1	0	0	Call on positive
CM	1	1	1	1	1	1	0	0	Call on minus
CPE	1	1	1	0	1	1	0	0	Call on parity even
CPO	1	1	1	0	0	1	0	0	Call on parity odd
RETURN									
RET	1	1	0	0	1	0	0	1	Return
RC	1	1	0	1	1	0	0	0	Return on carry
RNC	1	1	0	1	0	0	0	0	Return on no carry
RZ	1	1	0	0	1	0	0	0	Return on zero
RNZ	1	1	0	0	0	0	0	0	Return on no zero
RP	1	1	1	1	0	0	0	0	Return on positive
RM	1	1	1	1	1	0	0	0	Return on minus
RPE	1	1	1	0	1	0	0	0	Return on parity even
RPO	1	1	1	0	0	0	0	0	Return on parity odd
RESTART									
RST	1	1	A	A	A	1	1	1	Restart
INPUT/OUTPUT									
IN	1	1	0	1	1	0	1	1	Input
OUT	1	1	0	1	0	0	1	1	Output
INCREMENT AND DECREMENT									
INR r	0	0	D	D	D	1	0	0	Increment register
DCR r	0	0	D	D	D	1	0	1	Decrement register
INR M	0	0	1	1	0	1	0	0	Increment memory
DCR M	0	0	1	1	0	1	0	1	Decrement memory
INX B	0	0	0	0	0	0	1	1	Increment B & C registers
INX D	0	0	0	1	0	0	1	1	Increment D & E registers
INX H	0	0	1	0	0	0	1	1	Increment H & L registers
DCX B	0	0	0	0	1	0	1	1	Decrement B & C
DCX D	0	0	0	1	1	0	1	1	Decrement D & E
DCX H	0	0	1	0	1	0	1	1	Decrement H & L
ADD									
ADD r	1	0	0	0	0	S	S	S	Add register to A
ADC r	1	0	0	0	1	S	S	S	Add register to A with carry
ADD M	1	0	C	0	0	1	1	0	Add memory to A
ADC M	1	0	0	0	1	1	1	0	Add memory to A with carry
ADI	1	1	0	0	0	1	1	0	Add immediate to A
ACI	1	1	0	0	1	1	1	0	Add immediate to A with carry
DAD B	0	0	0	0	1	0	0	1	Add B & C to H & L
DAD D	0	0	0	1	1	0	0	1	Add D & E to H & L
DAD H	0	0	1	0	1	0	0	1	Add H & L to H & L
DAD SP	0	0	1	1	1	0	0	1	Add stack pointer to H & L
SUBTRACT									
SUB r	1	0	0	1	0	S	S	S	Subtract register from A
SBB r	1	0	0	1	1	S	S	S	Subtract register from A with borrow
SUB M	1	0	0	1	0	1	1	0	Subtract memory from A
SBB M	1	0	0	1	1	1	1	0	Subtract memory from A with borrow
SUI	1	1	0	1	0	1	1	0	Subtract immediate from A
SBI	1	1	0	1	1	1	1	0	Subtract immediate from A with borrow

Table 6. Instruction Set Summary (Continued)

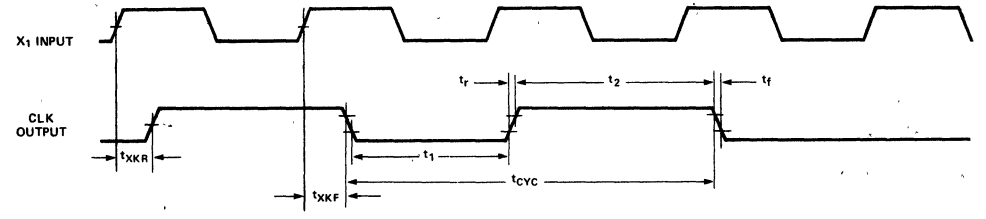
Mnemonic	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
LOGICAL									
ANA r	1	0	1	0	0	S	S	S	And register with A
XRA r	1	0	1	0	1	S	S	S	Exclusive OR register with A
ORA r	1	0	1	1	0	S	S	S	OR register with A
CMP r	1	0	1	1	1	S	S	S	Compare register with A
ANA M	1	0	1	0	0	1	1	0	And memory with A
XRA M	1	0	1	0	1	1	1	0	Exclusive OR memory with A
ORA M	1	0	1	1	0	1	1	0	OR memory with A
CMP M	1	0	1	1	1	1	1	0	Compare memory with A
ANI	1	1	1	0	0	1	1	0	And immediate with A
XRI	1	1	1	0	1	1	1	0	Exclusive OR immediate with A
ORI	1	1	1	1	0	1	1	0	OR immediate with A
CPI	1	1	1	1	1	1	1	0	Compare immediate with A
ROTATE									
RLC	0	0	0	0	0	1	1	1	Rotate A left
RRC	0	0	0	0	1	1	1	1	Rotate A right
RAL	0	0	0	1	0	1	1	1	Rotate A left through carry
RAR	0	0	0	1	1	1	1	1	Rotate A right through carry

Mnemonic	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
SPECIALS									
CMA	0	0	1	0	1	1	1	1	Complement A
STC	0	0	1	1	0	1	1	1	Set carry
CMC	0	0	1	1	1	1	1	1	Complement carry
DAA	0	0	1	0	0	1	1	1	Decimal adjust A
CONTROL									
EI	1	1	1	1	1	0	1	1	Enable Interrupts
DI	1	1	1	1	0	0	1	1	Disable Interrupt
NOP	0	0	0	0	0	0	0	0	No-operation
HLT	0	1	1	1	0	1	1	0	Halt
NEW 8085A INSTRUCTIONS									
RIM	0	0	1	0	0	0	0	0	Read Interrupt Mask
SIM	0	0	1	1	0	0	0	0	Set Interrupt Mask

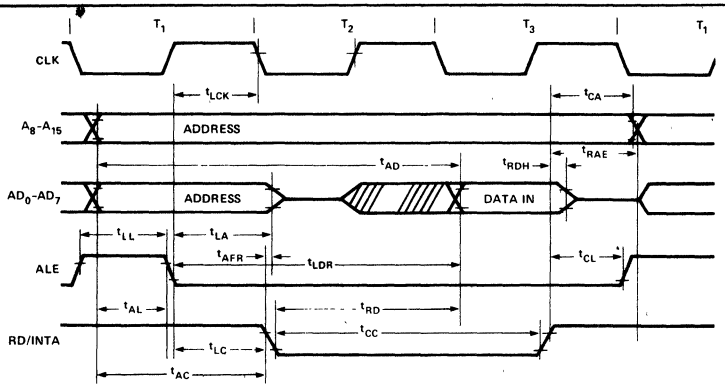
NOTES:
1. DDS or SSS: B 000, C 001, D 010, E011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags.

WAVEFORMS

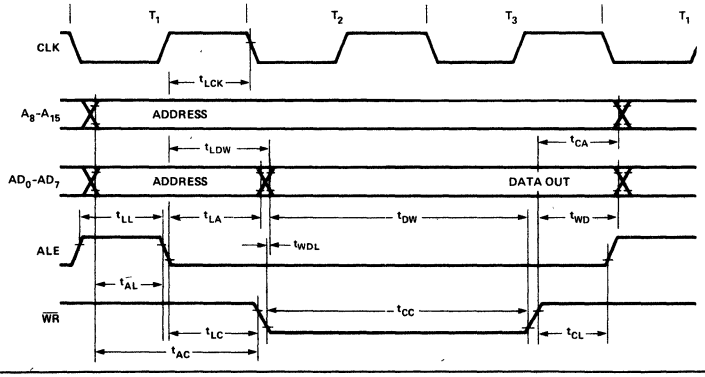
CLOCK



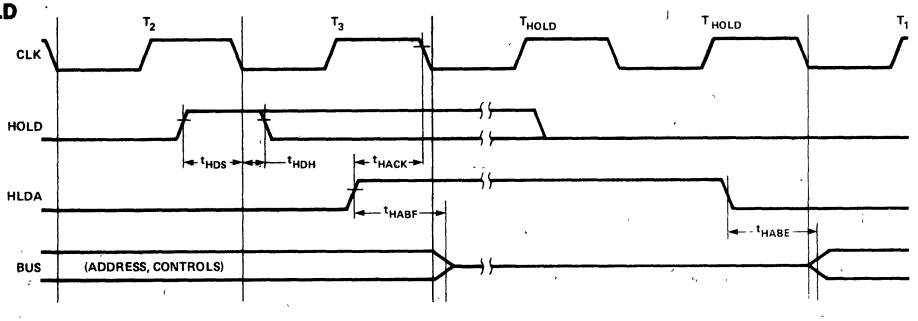
READ



WRITE



HOLD



8085A/8085A-2
SINGLE CHIP 8-BIT N-CHANNEL MICROPROCESSORS

- **Single +5V Power Supply**
- **100% Software Compatible with 8080A**
- **1.3 μs Instruction Cycle (8085A); 0.8 μs (8085A-2)**
- **On-Chip Clock Generator (with External Crystal, LC or RC Network)**
- **On-Chip System Controller; Advanced Cycle Status Information Available for Large System Control**
- **Four Vectored Interrupt Inputs (One is Non-Maskable) Plus an 8080A-Compatible Interrupt**
- **Serial In/Serial Out Port**
- **Decimal, Binary and Double Precision Arithmetic**
- **Direct Addressing Capability to 64k Bytes of Memory**

The Intel® 8085A is a complete 8 bit parallel Central Processing Unit (CPU). Its instruction set is 100% software compatible with the 8080A microprocessor, and it is designed to improve the present 8080A's performance by higher system speed. Its high level of system integration allows a minimum system of three IC's [8085A (CPU), 8156 (RAM/IO) and 8355/8755A (ROM/PROM/IO)] while maintaining total system expandability. The 8085A-2 is a faster version of the 8085A.

The 8085A incorporates all of the features that the 8224 (clock generator) and 8228 (system controller) provided for the 8080A, thereby offering a high level of system integration.

The 8085A uses a multiplexed data bus. The address is split between the 8 bit address bus and the 8 bit data bus. The on-chip address latches of 8155/8156/8355/8755A memory products allow a direct interface with the 8085A.

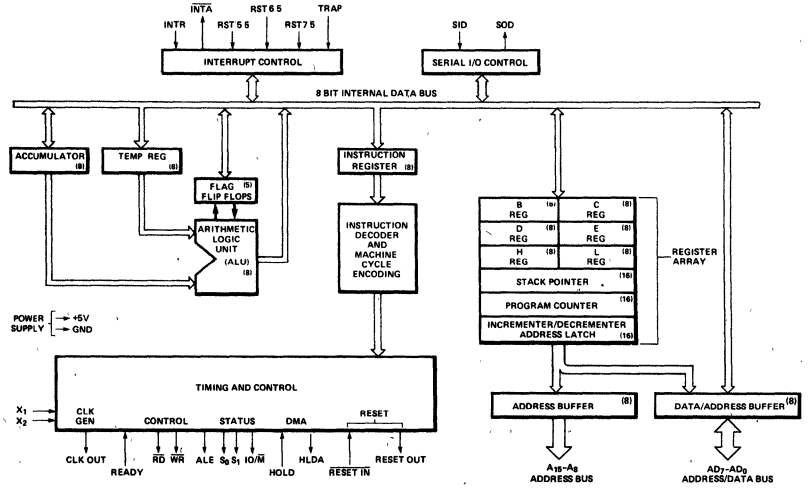


Figure 1. 8085A CPU Functional Block Diagram

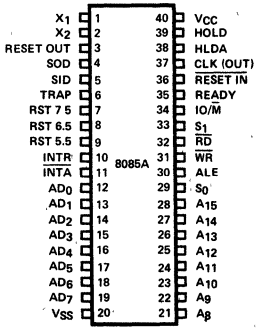


Figure 2. 8085A Pin Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias. . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 0V ±5%, V_{SS} = 0V; unless otherwise specified)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{CC}	Power Supply Current		170	mA	
I_{IL}	Input Leakage		±10	μA	$0 \leqslant V_{IN} \leqslant V_{CC}$
I_{LO}	Output Leakage		±10	μA	$0.45V \leqslant V_{out} \leqslant V_{CC}$
V_{ILR}	Input Low Level, RESET	-0.5	+0.8	V	
V_{IHR}	Input High Level, RESET	2.4	V_{CC} +0.5	V	
V_{HY}	Hysteresis, RESET	0.25		V	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 0V ±5%, V_{SS} = 0V)

Symbol	Parameter	8085A[2]		8085A-2[2]		Units
		Min.	Max.	Min.	Max.	
t_{CYC}	CLK Cycle Period	320	2000	200	2000	ns
t_1	CLK Low Time (Standard CLK Loading)	80		40		ns
t_2	CLK High Time (Standard CLK Loading)	120		70		ns
t_r, t_f	CLK Rise and Fall Time		30		30	ns
t_{XKR}	X_1 Rising to CLK Rising	30	120	30	100	ns
t_{XKF}	X_1 Rising to CLK Falling	30	150	30	110	ns
t_{AC}	A_{8-15} Valid to Leading Edge of Control[1]	270		115		ns
t_{ACL}	A_{0-7} Valid to Leading Edge of Control	240		115		ns
t_{AD}	A_{0-15} Valid to Valid Data In		575		350	ns
t_{AFR}	Address Float After Leading Edge of $\overline{READ}$ ($\overline{INTA}$)		0		0	ns
t_{AL}	A_{8-15} Valid Before Trailing Edge of ALE[1]	115		50		ns
t_{ALL}	A_{0-7} Valid Before Trailing Edge of ALE	90		50		ns
t_{ARY}	READY Valid from Address Valid		220		100	ns
t_{CA}	Address (A_{8-15}) Valid After Control	120		60		ns
t_{CC}	Width of Control Low ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$) Edge of ALE	400		230		ns
t_{CL}	Trailing Edge of Control to Leading Edge of ALE	50		25		ns
t_{DW}	Data Valid to Trailing Edge of $\overline{WRITE}$	420		230		ns
t_{HABE}	HLDA to Bus Enable		210		150	ns
t_{HABF}	Bus Float After HLDA		210		150	ns
t_{HACK}	HLDA Valid to Trailing Edge of CLK	110		40		ns
t_{HDH}	HOLD Hold Time	0		0		ns
t_{HDS}	HOLD Setup Time to Trailing Edge of CLK	170		120		ns
t_{INH}	INTR Hold Time	0		0		ns
t_{INS}	INTR, RST, and TRAP Setup Time to Falling Edge of CLK	160		150		ns
t_{LA}	Address Hold Time After ALE	100		50		ns
t_{LC}	Trailing Edge of ALE to Leading Edge of Control	130		60		ns
t_{LCK}	ALE Low During CLK High	100		50		ns
t_{LDR}	ALE to Valid Data During Read		460		270	ns
t_{LDW}	ALE to Valid Data During Write		200		120	ns
t_{LL}	ALE Width	140		80		ns
t_{LRY}	ALE to READY Stable		110		30	ns

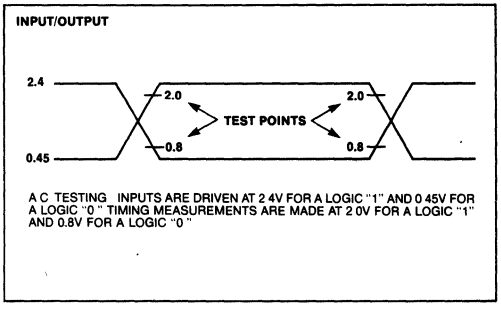
A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8085A[2]		8085A-2[2]		Units
		Min.	Max.	Min.	Max.	
t_{RAE}	Trailing Edge of $\overline{READ}$ to Re-Enabling of Address	150		90		ns
t_{RD}	$\overline{READ}$ (or $\overline{INTA}$) to Valid Data		300		150	ns
t_{RV}	Control Trailing Edge to Leading Edge of Next Control	400		220		ns
t_{RDH}	Data Hold Time After $\overline{READ}$ $\overline{INTA}$[7]	0		0		ns
t_{RYH}	READY Hold Time	0		0		ns
t_{RYS}	READY Setup Time to Leading Edge of CLK	110		100		ns
t_{WD}	Data Valid After Trailing Edge of $\overline{WRITE}$	100		60		ns
t_{WDL}	LEADING Edge of WRITE to Data Valid		40		20	ns

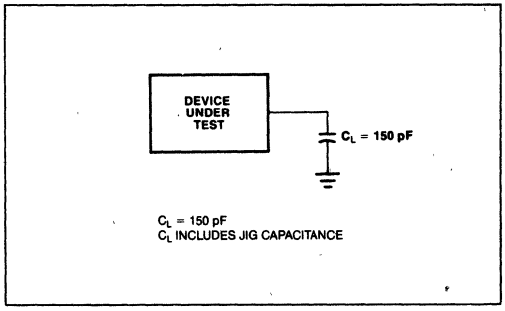
NOTES:

1. A_8-A_{15} address Specs apply to IO/$\overline{M}$, S_0, and S_1 except A_8-A_{15} are undefined during T_4-T_6 of OF cycle whereas IO/$\overline{M}$, S_0, andS_1 are stable.
2. Test conditions: t_{CYC} = 320 ns (8085A)/200 ns (8085A-2); C_L = 150 pF.
3. For all output timing where C_L = 150 pF use the following correction factors:
   25 pF ≤ C_L < 150 pF: −0.10 ns/pF
   150 pF < C_L ≤ 300 pF: +0.30 ns/pF
4. Output timings are measured with purely capacitive load.
5. All timings are measured at output votage V_L = 0.8V, V_H = 2.0V, and 1.5V with 20 ns rise and fall time on inputs.
6. To calculate timing specifications at other values of t_{CYC} use Table 7.
7. Data hold time is guaranteed under all loading conditions.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



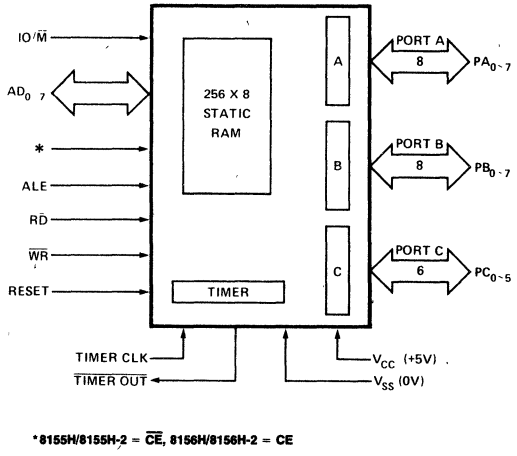
8155H/8156H/8155H-2/8156H-2
2048-BIT STATIC HMOS RAM WITH I/O PORTS AND TIMER

- **Single +5V Power Supply with 10% Voltage Margins**
- **30% Lower Power Consumption than the 8155 and 8156**
- **100% Compatible with 8155 and 8156**
- **256 Word x 8 Bits**
- **Completely Static Operation**
- **Internal Address Latch**
- **2 Programmable 8-Bit I/O Ports**
- **1 Programmable 6-Bit I/O Port**
- **Programmable 14-Bit Binary Counter/ Timer**
- **Compatible with 8085AH, 8085A and 8088 CPU**
- **Multiplexed Address and Data Bus**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8155H and 8156H are RAM and I/O chips implemented in N-Channel, depletion load, silicon gate technology (HMOS), to be used in the 8085AH and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085AH CPU.The 8155H-2 and 8156H-2 have maximum access times of 330 ns for use with the 8085AH-2 and the 5 MHz 8088 CPU.

The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.



*8155H/8155H-2 = $\overline{CE}$, 8156H/8156H-2 = CE

Figure 1. Block Diagram

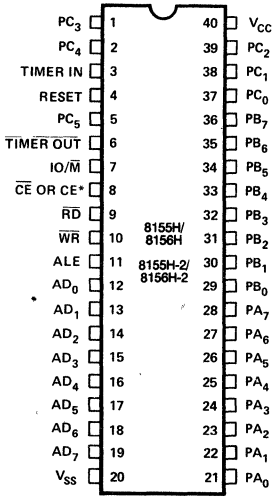


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
RESET	I	**Reset:** Pulse provided by the 8085AH to initialize the system (connect to 8085AH RESET OUT). Input high on this line resets the chip and initializes the three I/O ports to input mode. The width of RESET pulse should typically be two 8085AH clock cycle times.
AD_{0-7}	I/O	**Address/Data:** 3-state Address/Data lines that interface with the CPU lower 8-bit Address/Data Bus. The 8-bit address is latched into the address latch inside the 8155H/56H on the falling edge of ALE. The address can be either for the memory section or the I/O section depending on the IO/$\overline{M}$ input. The 8-bit data is either written into the chip or read from the chip, depending on the $\overline{WR}$ or $\overline{RD}$ input signal.
CE or $\overline{CE}$	I	**Chip Enable:** On the 8155H, this pin is $\overline{CE}$ and is ACTIVE LOW. On the 8156H, this pin is CE and is ACTIVE HIGH.
$\overline{RD}$	I	**Read Control:** Input low on this line with the Chip Enable active enables and AD_{0-7} buffers. If IO/$\overline{M}$ pin is low, the RAM content will be read out to the AD bus. Otherwise the content of the selected I/O port or command/status registers will be read to the AD bus.
$\overline{WR}$	I	**Write Control:** Input low on this line with the Chip Enable active causes the data on the Address/Data bus to be written to the RAM or I/O ports and command/status register, depending on IO/$\overline{M}$.
ALE	I	**Address Latch Enable:** This control signal latches both the address on the AD_{0-7} lines and the state of the Chip Enable and IO/$\overline{M}$ into the chip at the falling edge of ALE.
IO/$\overline{M}$	I	**I/O Memory:** Selects memory if low and I/O and command/status registers if high.
PA_{0-7}(8)	I/O	**Port A:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PB_{0-7}(8)	I/O	**Port B:** These 8 pins are general purpose I/O pins The in/out direction is selected by programming the command register
PC_{0-5}(6)	I/O	**Port C:** These 6 pins can function as either input port, output port, or as control signals for PA and PB. Programming is done through the command register. When PC_{0-5} are used as control signals, they will provide the following: PC_0 — A INTR (Port A Interrupt) PC_1 — $\overline{ABF}$ (Port A Buffer Full) PC_2 — $\overline{A\ STB}$ (Port A Strobe) PC_3 — B INTR (Port B Interrupt) PC_4 — $\overline{B\ BF}$ (Port B Buffer Full) PC_5 — $\overline{B\ STB}$ (Port B Strobe)
TIMER IN	I	**Timer Input:** Input to the counter-timer.
$\overline{TIMER\ OUT}$	O	**Timer Output:** This output can be either a square wave or a pulse, depending on the timer mode.
V_{CC}		**Voltage:** +5 volt supply.
V_{SS}		**Ground:** Ground reference.

FUNCTIONAL DESCRIPTION

The 8155H/8156H contains the following:

- 2k Bit Static RAM organized as 256 x 8
- Two 8-bit I/O ports (PA & PB) and one 6-bit I/O port (PC)
- 14-bit timer-counter

The IO/$\overline{M}$ (IO/Memory Select) pin selects either the five registers (Command, Status, PA_{0-7}, PB_{0-7}, PC_{0-5}) or the memory (RAM) portion.

The 8-bit address on the Address/Data lines, Chip Enable input CE or $\overline{CE}$, and IO/$\overline{M}$ are all latched on-chip at the falling edge of ALE

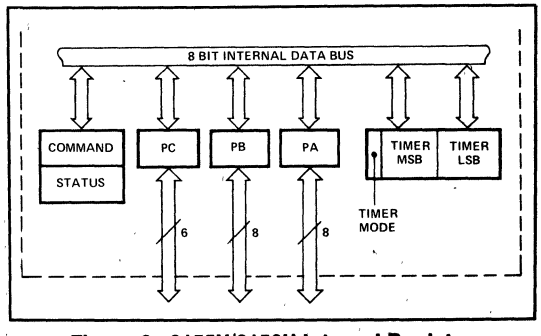


Figure 3. 8155H/8156H Internal Registers

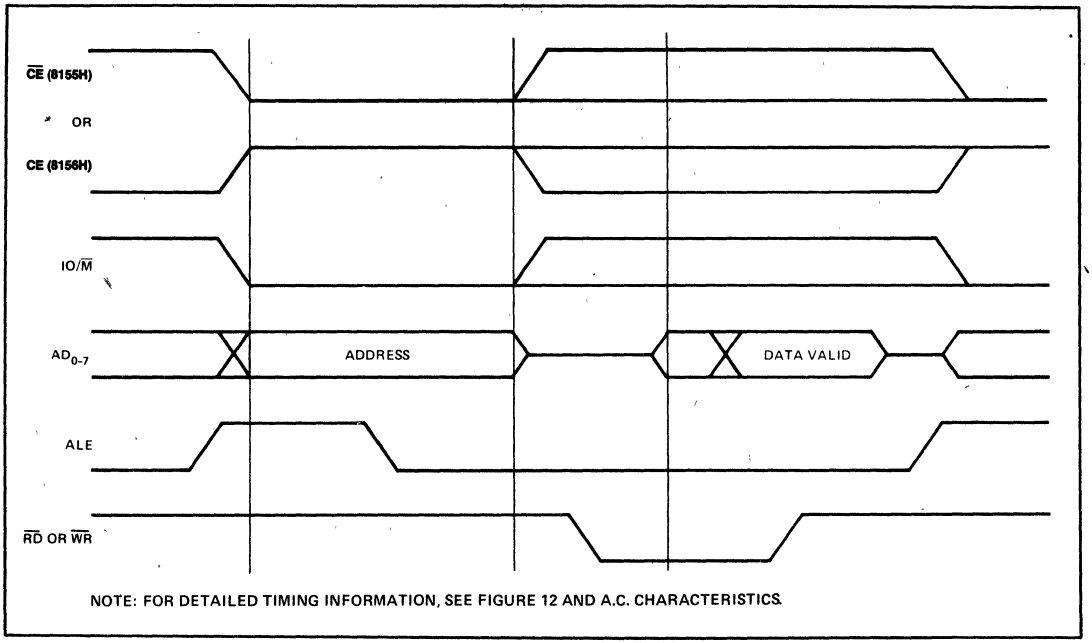


Figure 4. 8155H/8156H On-Board Memory Read/Write Cycle

PROGRAMMING OF THE COMMAND REGISTER

The command register consists of eight latches. Four bits (0-3) define the mode of the ports, two bits (4-5) enable or disable the interrupt from port C when it acts as control port, and the last two bits (6-7) are for the timer.

The command register contents can be altered at any time by using the I/O address XXXXX000 during a WRITE operation with the Chip Enable active and IO/$\overline{M}$ = 1. The meaning of each bit of the command byte is defined in Figure 5. The contents of the command register may never be read.

READING THE STATUS REGISTER

The status register consists of seven latches, one for each bit, six (0-5) for the status of the ports and one (6) for the status of the timer.

The status of the timer and the I/O section can be polled by reading the Status Register (Address XXXXX000). Status word format is shown in Figure 6. Note that you may never write to the status register since the command register shares the same I/O address and the command register is selected when a write to that address is issued.

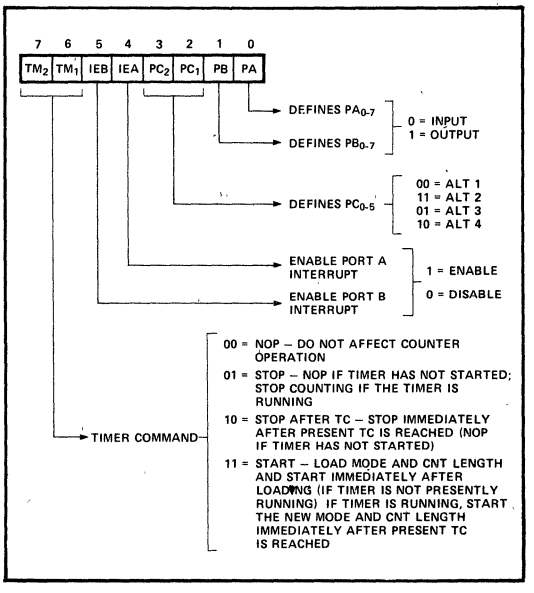


Figure 5. Command Register Bit Assignment

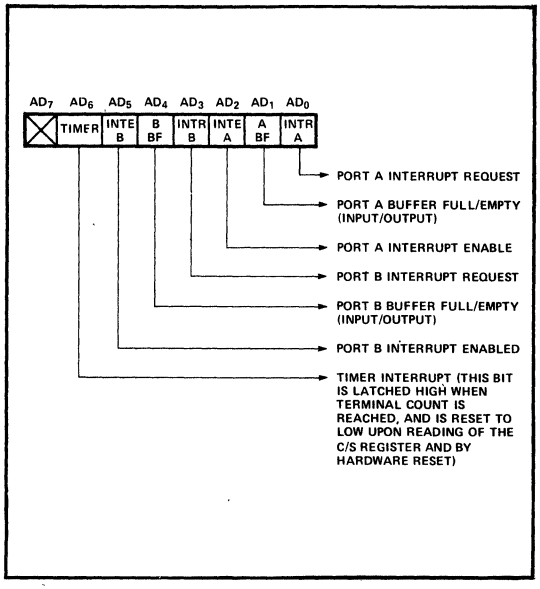


Figure 6. Status Register Bit Assignment

INPUT/OUTPUT SECTION

The I/O section of the 8155H/8156H consists of five registers: (See Figure 7.)

- **Command/Status Register (C/S)** — Both registers are assigned the address XXXXX000. The C/S address serves the dual purpose.

  When the C/S registers are selected during WRITE operation, a command is written into the command register. The contents of this register are *not* accessible through the pins.

  When the C/S (XXXXX000) is selected during a READ operation, the status information of the I/O ports and the timer becomes available on the AD_{0-7} lines.

- **PA Register** — This register can be programmed to be either input or output ports depending on the status of the contents of the C/S Register. Also depending on the command, this port can operate in either the basic mode or the strobed mode (See timing diagram). The I/O pins assigned in relation to this register are PA_{0-7}. The address of this register is XXXXX001.

- **PB Register** — This register functions the same as PA Register. The I/O pins assigned are PB_{0-7}. The address of this register is XXXXX010.

- **PC Register** — This register has the address XXXXX011 and contains only 6 bits. The 6 bits can be programmed to be either input ports, output ports or as control signals for PA and PB by properly programming the AD_2 and AD_3 bits of the C/S register.

  When PC_{0-5} is used as a control port, 3 bits are assigned for Port A and 3 for Port B. The first bit is an interrupt that the 8155H sends out. The second is an output signal indicating whether the buffer is full or empty, and the third is an input pin to accept a strobe for the strobed input mode. (See Table 2.)

When the 'C' port is programmed to either ALT3 or ALT4, the control signals for PA and PB are initialized as follows:

CONTROL	INPUT MODE	OUTPUT MODE
BF	Low	Low
INTR	Low	High
STB	Input Control	Input Control

I/O ADDRESS†								SELECTION
A7	A6	A5	A4	A3	A2	A1	A0	
X	X	X	X	X	0	0	0	Interval Command/Status Register
X	X	X	X	X	0	0	1	General Purpose I/O Port A
X	X	X	X	X	0	1	0	General Purpose I/O Port B
X	X	X	X	X	0	1	1	Port C — General Purpose I/O or Control
X	X	X	X	X	1	0	0	Low-Order 8 bits of Timer Count
X	X	X	X	X	1	0	1	High 6 bits of Timer Count and 2 bits of Timer Mode

X Don't Care

† I/O Address must be qualified by CE = 1 (8156H) or $\overline{CE}$ = 0 (8155H) and IO/$\overline{M}$ = 1 in order to select the appropriate register

Figure 7. I/O Port and Timer Addressing Scheme

Figure 8 shows how I/O PORTS A and B are structured within the 8155H and 8156H:

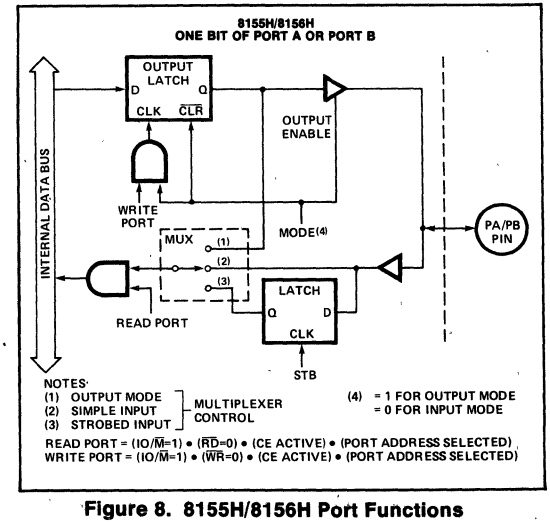


Figure 8. 8155H/8156H Port Functions

Table 2. Port Control Assignment

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	Input Port	Output Port	A INTR (Port A Interrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port	A BF (Port A Buffer Full)	A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A $\overline{STB}$ (Port A Strobe)	A $\overline{STB}$ (Port A Strobe)
PC3	Input Port	Output Port	Output Port	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port B Buffer Full)
PC5	Input Port	Output Port	Output Port	B $\overline{STB}$ (Port B Strobe)

Note in the diagram that when the I/O ports are programmed to be output ports, the contents of the output ports can still be read by a READ operation when appropriately addressed.

The outputs of the 8155H/8156H are "glitch-free" meaning that you can write a "1" to a bit position that was previously "1" and the level at the output pin will not change.

Note also that the output latch is cleared when the port enters the input mode. The output latch cannot be loaded by writing to the port if the port is in the input mode. The result is that each time a port mode is changed from input to output, **the output pins will go low. When the 8155H/56H is RESET, the output latches are all cleared and all 3 ports enter the input mode.**

When in the ALT 1 or ALT 2 modes, the bits of PORT C are structured like the diagram above in the simple input or output mode, respectively.

Reading from an input port with nothing connected to the pins will provide unpredictable results.

Figure 9 shows how the 8155H/8156H I/O ports might be configured in a typical MCS-85 system.

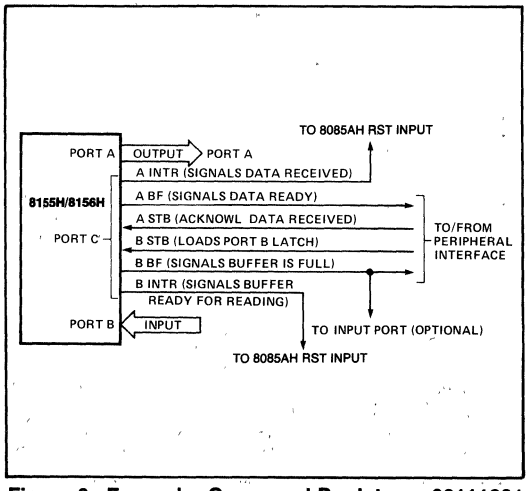


Figure 9. Example: Command Register = 00111001

TIMER SECTION

The timer is a 14-bit down-counter that counts the TIMER IN pulses and provides either a square wave or pulse when terminal count (TC) is reached.

The timer has the I/O address XXXXX100 for the low order byte of the register and the I/O address XXXXX101 for the high order byte of the register. (See Figure 7.)

To program the timer, the COUNT LENGTH REG is loaded first, one byte at a time, by selecting the timer addresses. Bits 0-13 of the high order count register will specify the length of the next count and bits 14-15 of the high order register will specify the timer output mode (see Figure 10). The value loaded into the count length register can have any value from 2H through 3FFH in Bits 0–13.

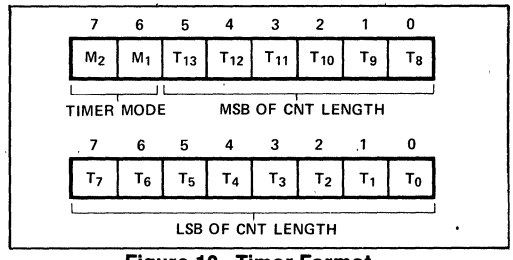


Figure 10. Timer Format

There are four modes to choose from: M2 and M1 define the timer mode, as shown in Figure 11.

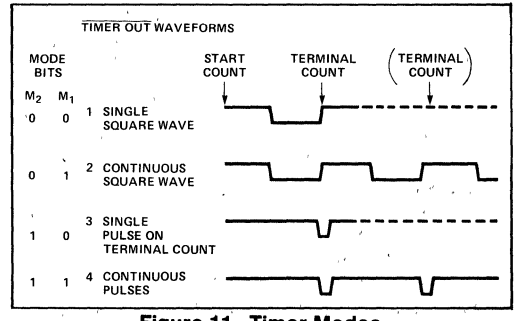


Figure 11. Timer Modes

Bits 6-7 (TM_2 and TM_1) of command register contents are used to start and stop the counter. There are four commands to choose from:

TM_2	TM_1	
0	0	NOP — Do not affect counter operation.
0	1	STOP — NOP if timer has not started; stop counting if the timer is running.
1	0	STOP AFTER TC — Stop immediately after present TC is reached (NOP if timer has not started)
1	1	START — Load mode and CNT length and start immediately after loading (if timer is not presently running). If timer is running, start the new mode and CNT length immediately after present TC is reached.

Note that while the counter is counting, you may load a new count and mode into the count length registers. Before the new count and mode will be used by the counter, you must issue a START command to the counter. This applies even though you may only want to change the count and use the previous mode.

In case of an odd-numbered count, the first half-cycle of the squarewave output, which is high, is one count longer than the second (low) half-cycle, as shown in Figure 12.

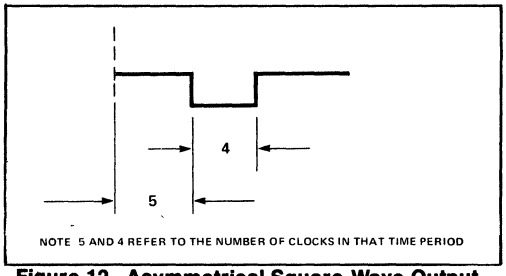


Figure 12. Asymmetrical Square-Wave Output Resulting from Count of 9

The counter in the 8155H is not initialized to any particular mode or count when hardware RESET occurs, but RESET does *stop* the counting. Therefore, counting cannot begin following RESET until a START command is issued via the C/S register.

Please note that the timer circuit on the 8155H/8156H chip is designed to be a square-wave timer, not an event counter. To achieve this, it counts down by twos twice in completing one cycle. Thus, its registers do not contain values directly representing the number of TIMER IN pulses received. You cannot load an initial value of 1 into the count register and cause the timer to operate, as its terminal count value is 10 (binary) or 2 (decimal). (For the detection of single pulses, it is suggested that one **of the hardware interrupt pins on the 8085AH be used.)** After the timer has started counting down, the values residing in the count registers can be used to calculate the actual number of TIMER IN pulses required to complete the timer cycle if desired. To obtain the remaining count, perform the following operations in order:

1. Stop the count
2. Read in the 16-bit value from the count length registers
3. Reset the upper two mode bits
4. Reset the carry and rotate right one position all 16 bits through carry
5. If carry is set, add 1/2 of the full original count (1/2 full count — 1 if full count is odd).

Note: If you started with an odd count and you read the count length register before the third count pulse occurs, you will not be able to discern whether one or two counts has occurred. Regardless of this, the 8155H/56H always counts out the right number of pulses in generating the $\overline{\text{TIMER OUT}}$ waveforms.

8085A MINIMUM SYSTEM CONFIGURATION

Figure 13a shows a minimum system using three chips, containing:

- 256 Bytes RAM
- 2K Bytes ROM
- 38 I/O Pins
- 1 Interval Timer
- 4 Interrupt Levels

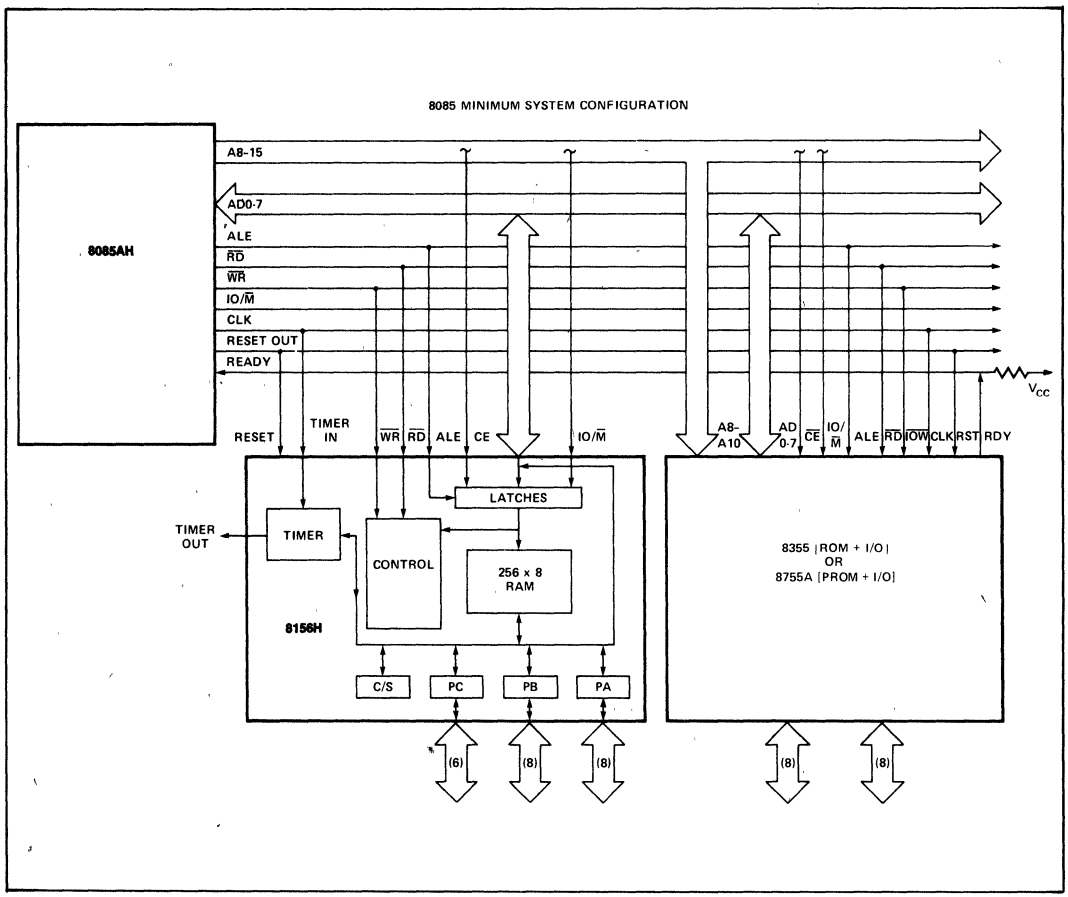


Figure 13a. 8085AH Minimum System Configuration (Memory Mapped I/O)

8088 FIVE CHIP SYSTEM

Figure 13b shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes ROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

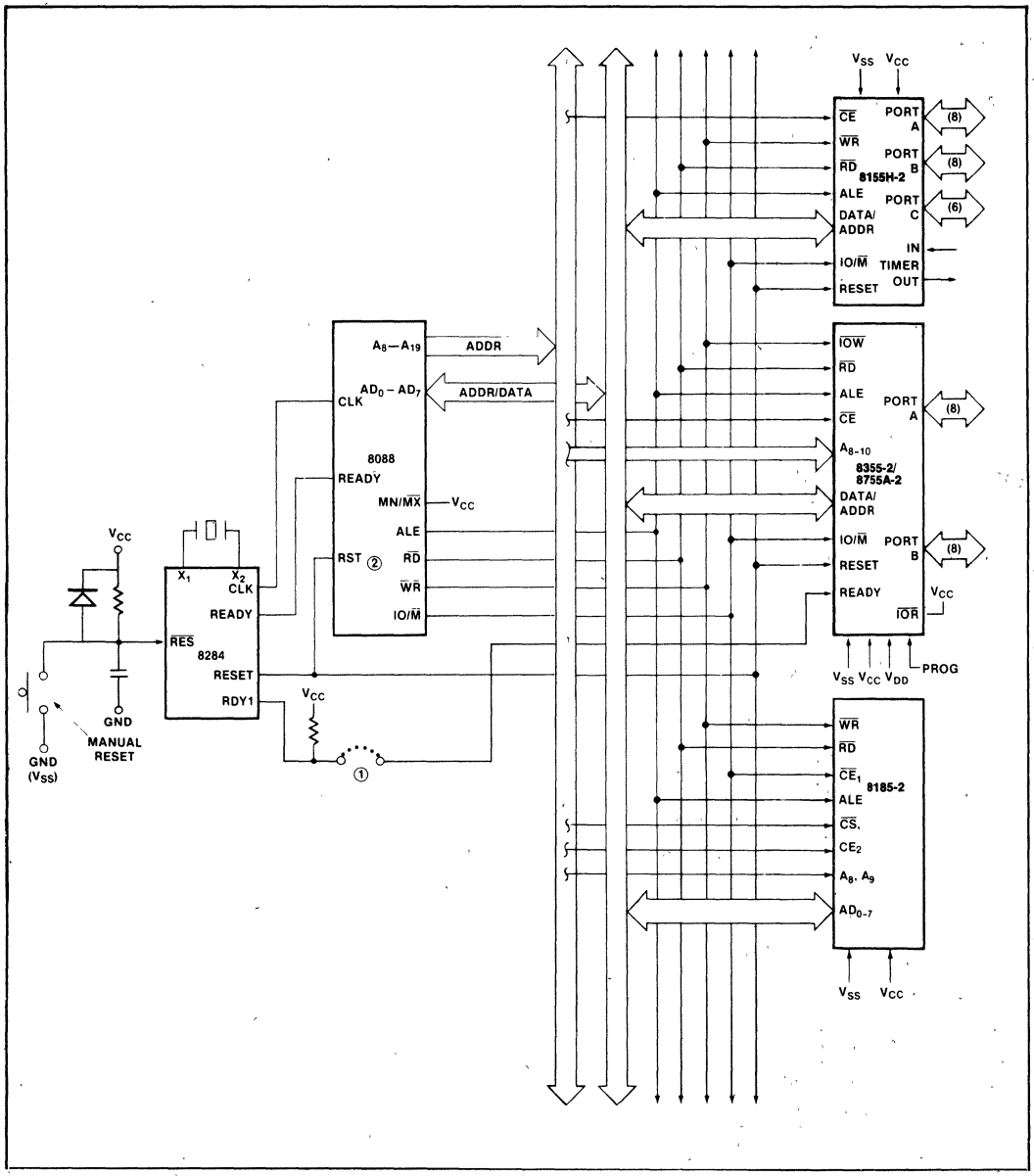


Figure 13b. 8088 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... -0.5V to +7V
Power Dissipation 1.5W

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		125	mA	
I_{IL}(CE)	Chip Enable Leakage 8155H 8156H		 +100 -100	 μA μA	$0V \leq V_{IN} \leq V_{CC}$

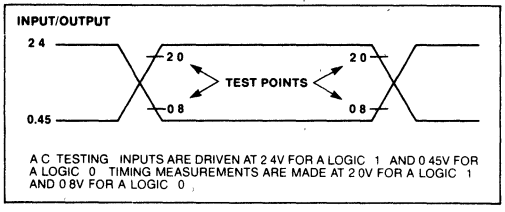
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

		8155H/8156H		8155H-2/8156H-2		
Symbol	Parameter	Min.	Max.	Min.	Max.	Units
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{AD}	Address Stable to Data Out Valid		400		330	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		100		ns
t_{WD}	Data In Hold Time After WRITE	25		25		ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Setup Time	70		50		ns
t_{RP}	Port Input Hold Time	50		10		ns
t_{SBF}	Strobe to Buffer Full		400		300	ns
t_{SS}	Strobe Width	200		150		ns
t_{RBE}	READ to Buffer Empty		400		300	ns
t_{SI}	Strobe to INTR On		400		300	ns

A.C. CHARACTERISTICS (Continued) (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

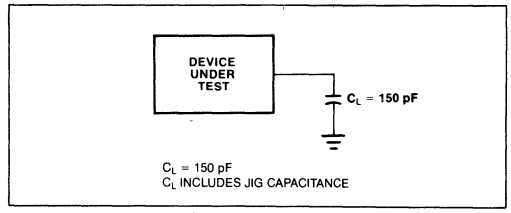
		8155H/8156H		8155H-2/8156H-2		
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer Full		400		300	ns
t_{WI}	WRITE to INTR Off		400		300	ns
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low		400		300	ns
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High		400		300	ns
t_{RDE}	Data Bus Enable from READ Control	10		10		ns
t_1	TIMER-IN Low Time	80		40		ns
t_2	TIMER-IN High Time	120		70		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

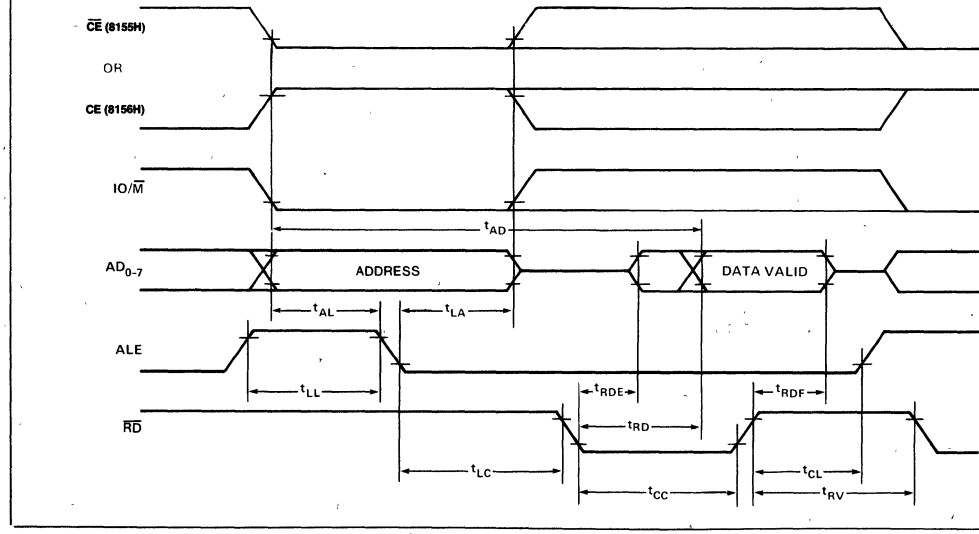
A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

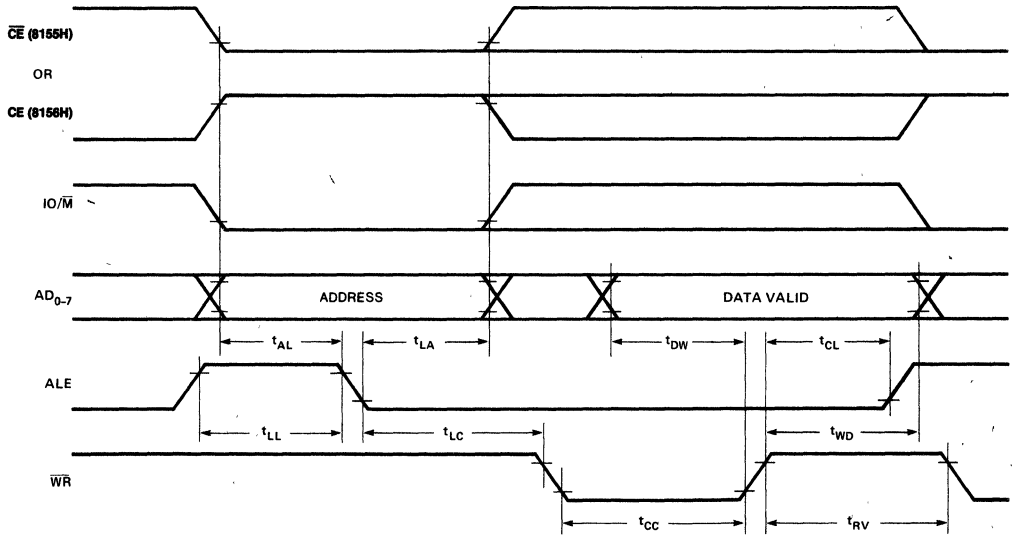
WAVEFORMS

READ

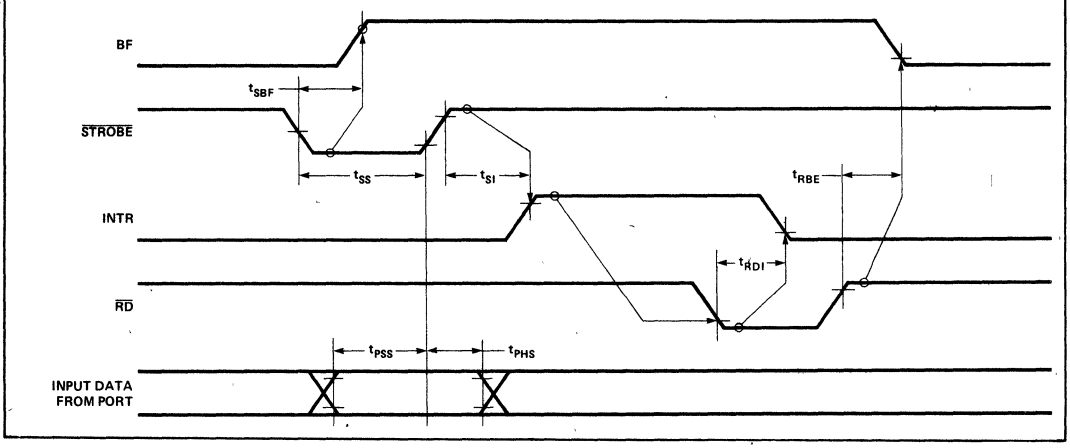


WAVEFORMS (Continued)

WRITE

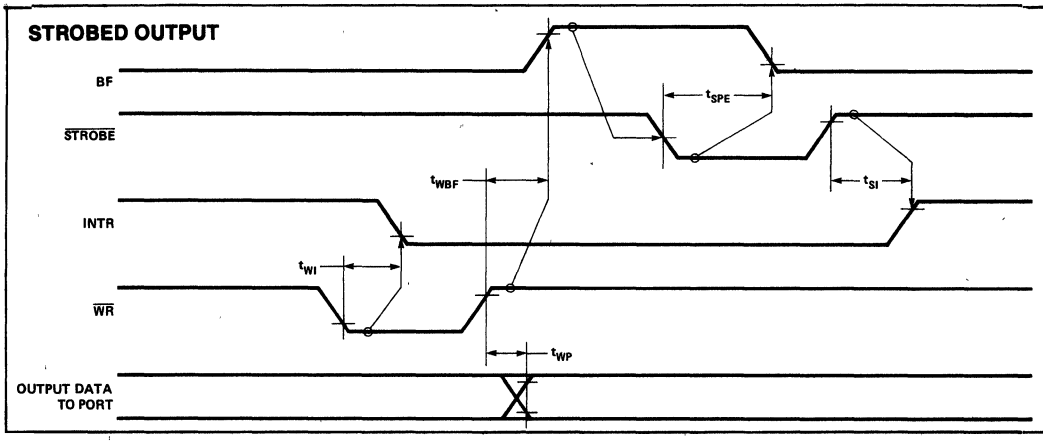


STROBED INPUT

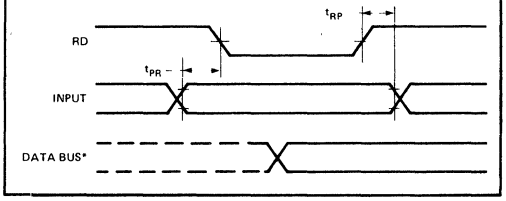


WAVEFORMS (Continued)

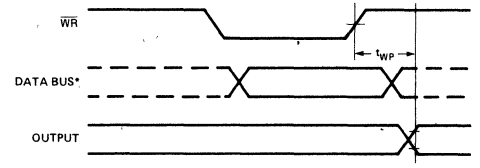
STROBED OUTPUT



BASIC INPUT

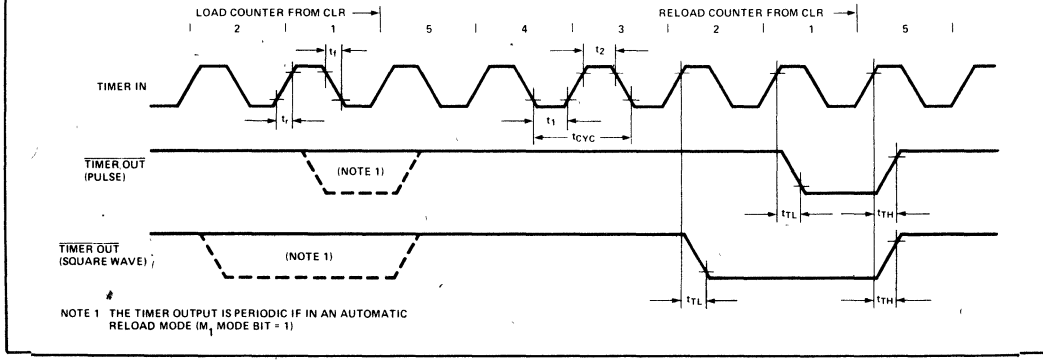


BASIC OUTPUT



*DATA BUS TIMING IS SHOWN IN FIGURE 7

TIMER OUTPUT COUNTDOWN FROM 5 TO 1



NOTE 1 THE TIMER OUTPUT IS PERIODIC IF IN AN AUTOMATIC RELOAD MODE (M_1 MODE BIT = 1)

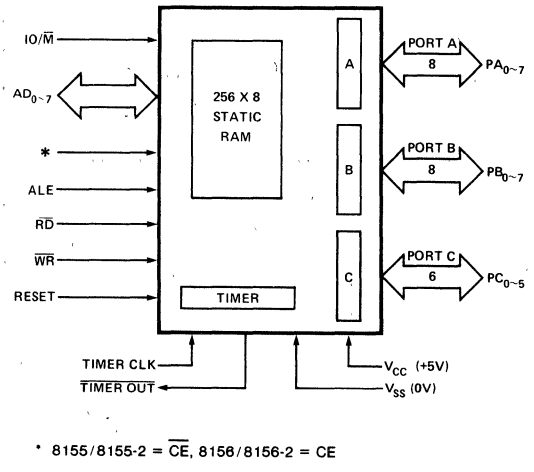
8155/8156/8155-2/8156-2
2048 BIT STATIC MOS RAM WITH I/O PORTS AND TIMER

- 256 Word x 8 Bits
- Single +5V Power Supply
- Completely Static Operation
- Internal Address Latch
- 2 Programmable 8 Bit I/O Ports
- 1 Programmable 6-Bit I/O Port
- Programmable 14-Bit Binary Counter/ Timer
- Compatible with 8085A and 8088 CPU
- Multiplexed Address and Data Bus
- 40 Pin DIP

The 8155 and 8156 are RAM and I/O chips to be used in the 8085A and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085A CPU. The 8155-2 and 8156-2 have maximum access times of 330 ns for use with the 8085A-2 and the 5 MHz 8088 CPU.

The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.



* 8155/8155-2 = $\overline{CE}$, 8156/8156-2 = CE

Figure 1. Block Diagram

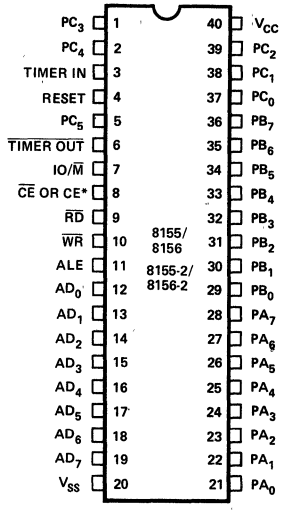


Figure 2. Pin Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... -0.5V to +7V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

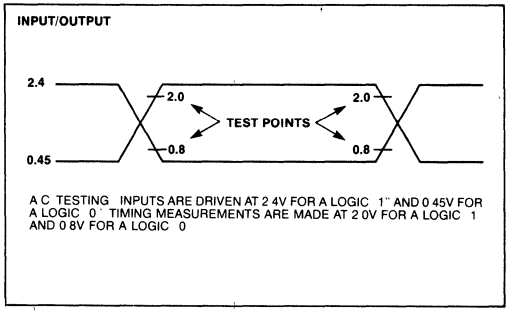
D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		180	mA	
I_{IL}(CE)	Chip Enable Leakage 8155 8156		 +100 -100	 μA μA	$0V \leq V_{IN} \leq V_{CC}$

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

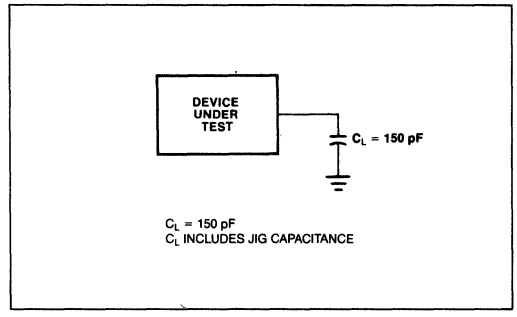
		8155/8156		8155-2/8156-2		
SYMBOL	PARAMETER	MIN.	MAX.	MIN.	MAX.	UNITS
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{AD}	Address Stable to Data Out Valid		400		330	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		100		ns
t_{WD}	Data In Hold Time After WRITE	25		25		ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Setup Time	70		50		ns
t_{RP}	Port Input Hold Time	50		10		ns
t_{SBF}	Strobe to Buffer Full		400		300	ns
t_{SS}	Strobe Width	200		150		ns
t_{RBE}	READ to Buffer Empty		400		300	ns
t_{SI}	Strobe to INTR On		400		300	ns
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer Full		400		300	ns
t_{WI}	WRITE to INTR Off		400		300	ns
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low		400		300	ns
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High		400		300	ns
t_{RDE}	Data Bus Enable from READ Control	10		10		ns
t_1	TIMER-IN Low Time	80		40		ns
t_2	TIMER-IN High Time	120		70		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0"

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

8185/8185-2
1024 x 8-BIT STATIC RAM FOR MCS-85®

- **Multiplexed Address and Data Bus**
- **Directly Compatible with 8085A and iAPX 88 Microprocessors**
- **Low Operating Power Dissipation**
- **Low Standby Power Dissipation**
- **Single +5V Supply**
- **High Density 18-Pin Package**

The Intel® 8185 is an 8192-bit static random access memory (RAM) organized as 1024 words by 8-bits using N-channel Silicon-Gate MOS technology. The multiplexed address and data bus allows the 8185 to interface directly to the 8085A and iAPX 88 microprocessors to provide a maximum level of system integration.

The low standby power dissipation minimizes system power requirements when the 8185 is disabled.

The 8185-2 is a high-speed selected version of the 8185 that is compatible with the 5 MHz 8085A-2 and the 5 MHz iAPX 88.

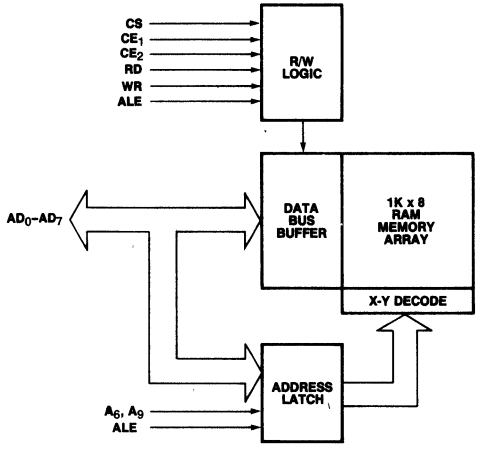
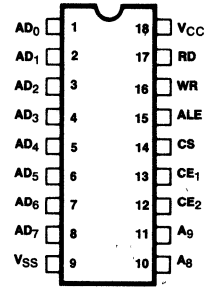


Figure 1. Block Diagram



AD_0–AD_7	ADDRESS/DATA LINES
A_8, A_9	ADDRESS LINES
CS	CHIP SELECT
CE_1	CHIP ENABLE (IO/M)
CE_2	CHIP ENABLE
ALE	ADDRESS LATCH ENABLE
WR	WRITE ENABLE

Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

The 8185 has been designed to provide for direct interface to the multiplexed bus structure and bus timing of the 8085A microprocessor.

At the beginning of an 8185 memory access cycle, the 8-bit address on AD_{0-7}, A_8 and A_9, and the status of $\overline{CE}_1$ and CE_2 are all latched internally in the 8185 by the falling edge of ALE. If the latched status of both $\overline{CE}_1$ and CE_2 are active, the 8185 powers itself up, but no action occurs until the $\overline{CS}$ line goes low and the appropriate $\overline{RD}$ or $\overline{WR}$ control signal input is activated.

The $\overline{CS}$ input is not latched by the 8185 in order to allow the maximum amount of time for address decoding in selecting the 8185 chip. Maximum power consumption savings will occur, however, only when $\overline{CE}_1$ and CE_2 are activated selectively to power down the 8185 when it is not in use. A possible connection would be to wire the 8085A's IO/$\overline{M}$ line to the 8185's $\overline{CE}_1$ input, thereby keeping the 8185 powered down during I/O and interrupt cycles.

Table 1.
Truth Table for
Power Down and Function Enable

$\overline{CE}_1$	CE_2	$\overline{CS}$	(CS*)[2]	8185 Status
1	X	X	0	Power Down and Function Disable[1]
X	0	X	0	Power Down and Function Disable[1]
0	1	1	0	Powered Up and Function Disable[1]
0	1	0	1	Powered Up and Enabled

NOTES:
X: Don't Care.
1: Function Disable implies Data Bus in high impedance state and not writing.
2: CS* = ($\overline{CE}_1$ = 0) • (CE_2 = 1) • ($\overline{CS}$ = 0)
CS* = 1 signifies all chip enables and chip select active

Table 2.
Truth Table for
Control and Data Bus Pin Status

(CS*)	$\overline{RD}$	$\overline{WR}$	AD_{0-7} During Data Portion of Cycle	8185 Function
0	X	X	Hi-Impedance	No Function
1	0	1	Data from Memory	Read
1	1	0	Data to Memory	Write
1	1	1	Hi-Impedance	Reading, but not Driving Data Bus

NOTE:
X: Don't Care.

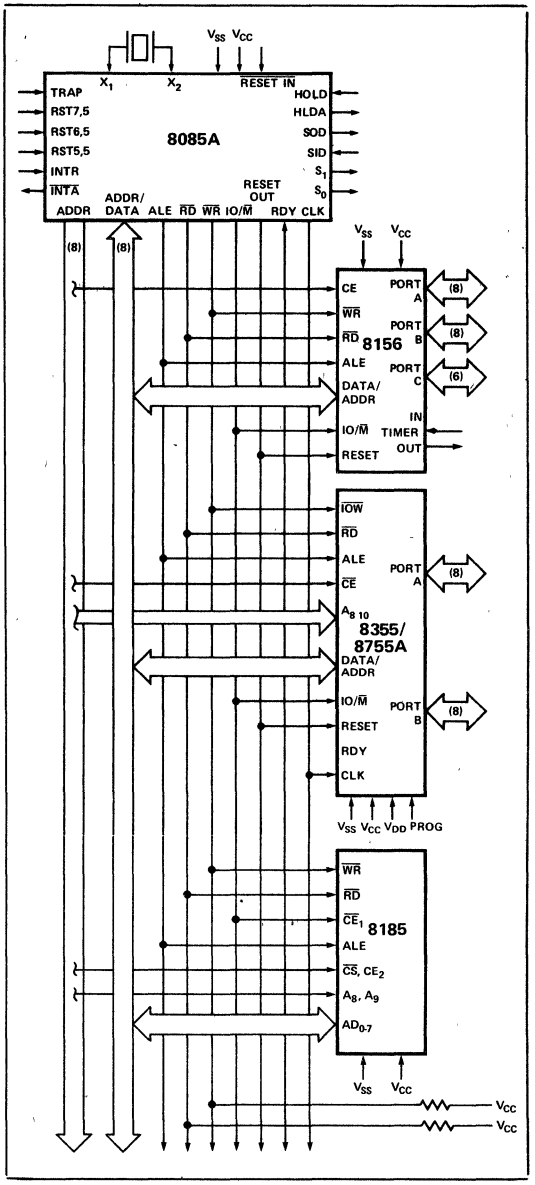


Figure 3. 8185 in an MCS-85 System

4 Chips:
2K Bytes ROM
1.25K Bytes RAM
38 I/O Lines
1 Counter/Timer
2 Serial I/O Lines
5 Interrupt Inputs

iAPX 88 FIVE CHIP SYSTEM:

- 1.25 K Bytes RAM
- 2 K Bytes ROM
- 38 I/O Pins
- 1 Internal Timer
- 2 Interrupt Levels

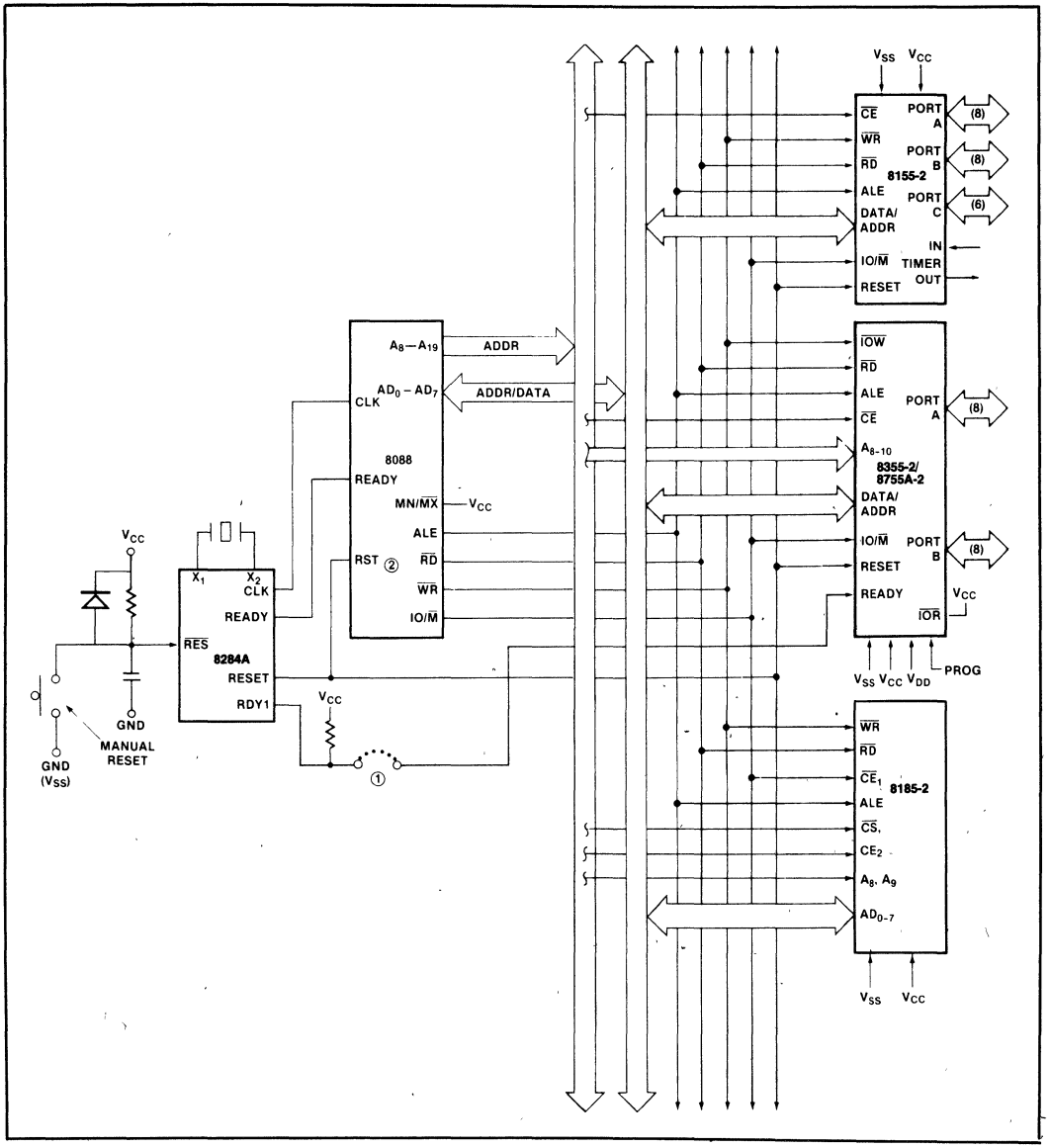


Figure 4. iAPX 88 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
with Respect to Ground ..... -0.5V to +7V
Power Dissipation 1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

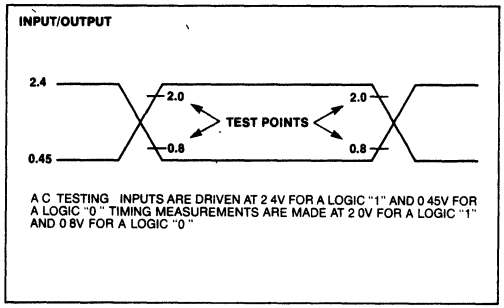
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4			I_{OH} = − 400μA
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current Powered Up		100	mA	
	Powered Down		35	mA	

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

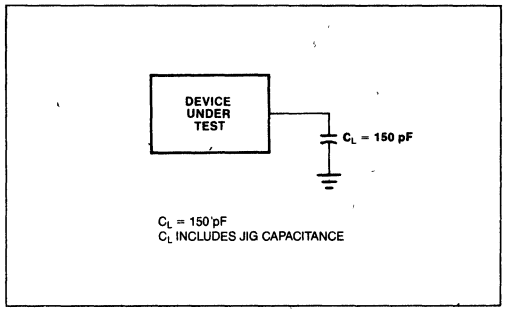
Symbol	Parameter	8185		8185-2		Units
		Min.	Max.	Min.	Max.	
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time After Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{LD}	ALE to Data Out Valid		300		200	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	20		20		ns
t_{SC}	Chip Select Set Up to Control Line	10		10		ns
t_{CS}	Chip Select Hold Time After Control	10		10		ns
t_{ALCE}	Chip Enable Set Up to ALE Falling	30		10		ns
t_{LACE}	Chip Enable Hold Time After ALE	50		30		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



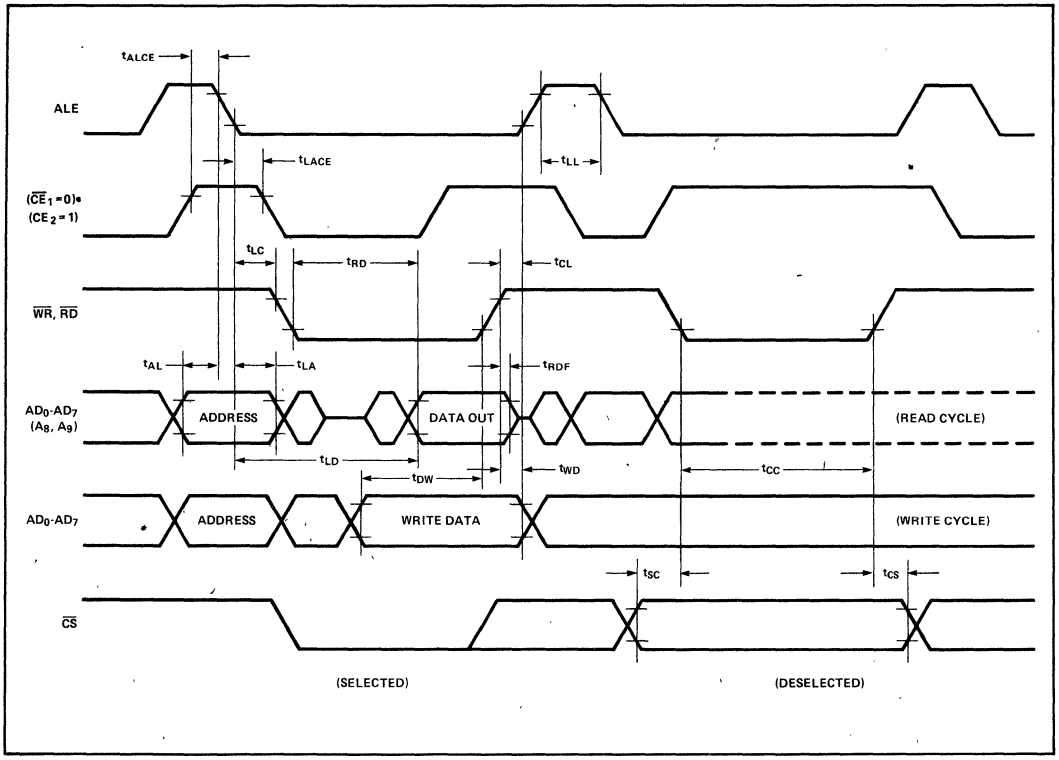
A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0"

A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORM

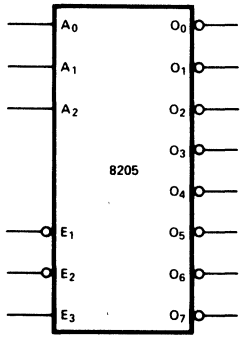


8205
HIGH SPEED 1 OUT OF 8 BINARY DECODER

- **I/O Port or Memory Selector**
- **Simple Expansion — Enable Inputs**
- **High Speed Schottky Bipolar Technology — 18ns Max. Delay**
- **Directly Compatible with TTL Logic Circuits**
- **Low Input Load Current — .25 mA max., 1/6 Standard TTL Input Load**
- **Minimum Line Reflection — Low Voltage Diode Input Clamp**
- **Outputs Sink 10 mA min.**
- **16-Pin Dual-In-Line Ceramic or Plastic Package**

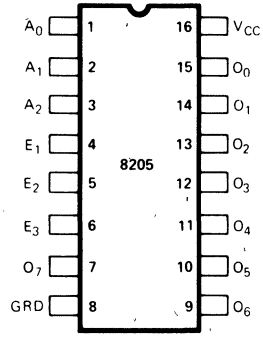
The Intel® 8205 decoder can be used for expansion of systems which utilize input ports, output ports, and memory components with active low chip select input. When the 8205 is enabled, one of its 8 outputs goes "low," thus a single row of a memory system is selected. The 3-chip enable inputs on the 8205 allow easy system expansion. For very large systems, 8205 decoders can be cascaded such that each decoder can drive 8 other decoders for arbitrary memory expansions.

The 8205 is packaged in a standard 16-pin dual in-line package, and its performance is specified over the temperature range of 0°C to +75°C, ambient. The use of Schottky barrier diode clamped transistors to obtain fast switching speeds results in higher performance than equivalent devices made with a gold diffussion process.



ADDRESS			ENABLE			OUTPUTS							
A_0	A_1	A_2	E_1	E_2	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	H	L	H	H
L	H	H	L	L	H	H	H	H	H	H	H	L	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
X	X	X	L	L	L	H	H	H	H	H	H	H	H
X	X	X	H	L	L	H	H	H	H	H	H	H	H
X	X	X	L	H	L	H	H	H	H	H	H	H	H
X	X	X	H	H	L	H	H	H	H	H	H	H	H
X	X	X	H	L	H	H	H	H	H	H	H	H	H
X	X	X	L	H	H	H	H	H	H	H	H	H	H
X	X	X	H	H	H	H	H	H	H	H	H	H	H

Figure 1. Logic Symbol



A_0 A_2	ADDRESS INPUTS
$\overline{E_1}$ $\overline{E_3}$	ENABLE INPUTS
$\overline{O_0}$ $\overline{O_7}$	DECODED OUTPUTS

Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

Decoder

The 8205 contains a one out of eight binary decoder. It accepts a three bit binary code and by gating this input, creates an exclusive output that represents the value of the input code.

For example, if a binary code of 101 was present on the A0, A1 and A2 address input lines, and the device was enabled, an active low signal would appear on the $\overline{O5}$ output line. Note that all of the other output pins are sitting at a logic high, thus the decoded output is said to be exclusive. The decoders outputs will follow the truth table shown below in the same manner for all other input variations.

Enable Gate

When using a decoder it is often necessary to gate the outputs with timing or enabling signals so that the exclusive output of the decoded value is synchronous with the overall system.

The 8205 has a built-in function for such gating. The three enable inputs ($\overline{E1}$, $\overline{E2}$, E3) are ANDed together and create a single enable signal for the decoder. The combination of both active "high" and active "low" device enable inputs provides the designer with a powerfully flexible gating function to help reduce package count in his system.

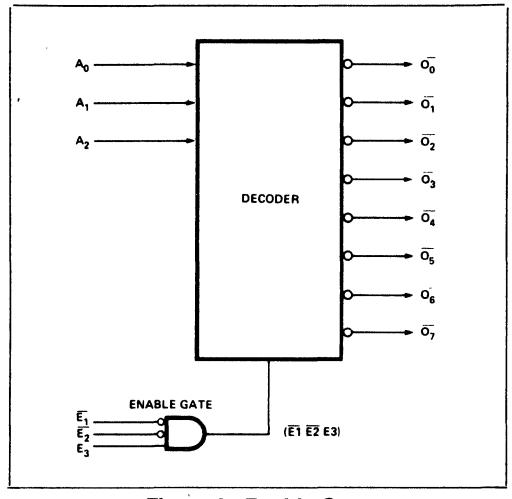


Figure 3. Enable Gate

ADDRESS			ENABLE			OUTPUTS							
A_0	A_1	A_2	E_1	E_2	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	H	L	H	H
L	H	H	L	L	H	H	H	H	H	H	H	L	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
X	X	X	L	L	L	H	H	H	H	H	H	H	H
X	X	X	H	L	L	H	H	H	H	H	H	H	H
X	X	X	L	H	L	H	H	H	H	H	H	H	H
X	X	X	H	H	L	H	H	H	H	H	H	H	H
X	X	X	H	L	H	H	H	H	H	H	H	H	H
X	X	X	L	H	H	H	H	H	H	H	H	H	H
X	X	X	H	H	H	H	H	H	H	H	H	H	H

Applications of the 8205

The 8205 can be used in a wide variety of applications in microcomputer systems. I/O ports can be decoded from the address bus, chip select signals can be generated to select memory devices and the type of machine state such as in 8008 systems can be derived from a simple decoding of the state lines (S0, S1, S2) of the 8008 CPU.

I/O PORT DECODER

Shown in the figure below is a typical application of the 8205. Address input lines are decoded by a group of 8205s (3). Each input has a binary weight. For example, A0 is assigned a value of 1 and is the LSB; A4 is assigned a value of 16 and is the MSB; By connecting them to the decoders as shown, an active low signal that is exclusive in nature and represents the value of the input address lines, is available at the outputs of the 8205s.

This circuit can be used to generate enable signals for I/O ports or any other decoder related application.

Note that no external gating is required to decode up to 24 exclusive devices and that a simple addition of an inverter or two will allow expansion to even larger decoder networks.

CHIP SELECT DECODER

Using a very similar circuit to the I/O port decoder, an array of 8205s can be used to create a simple interface to a 24K memory system.

The memory devices used can be either ROM or RAM and are 1K in storage capacity. 2708s and 2114As are devices typically used for this application. This type of memory device has ten (10) address inputs and an active "low" chip select ($\overline{CS}$). The lower order address bits A0–A9 which come from the microprocessor are "bussed" to all memory elements and the chip select to enable a specific device or group of devices comes from the array of 8205s. The output of the 8205 is active low so it is directly compatible with the memory components.

Basic operation is that the CPU issues an address to identify a specific memory location in which it wishes to "write" or "read" data. The most significant address bits A10-A14 are decoded by the array of 8205s and an exclusive, active low, chip select is generated that enables a specific memory device. The least significant address bits A0-A9 identify a specific location within the selected device. Thus, all addresses throughout the entire memory array are exclusive in nature and are non-redundant.

This technique can be expanded almost indefinitely to support even larger systems with the addition of a few inverters and an extra decoder (8205).

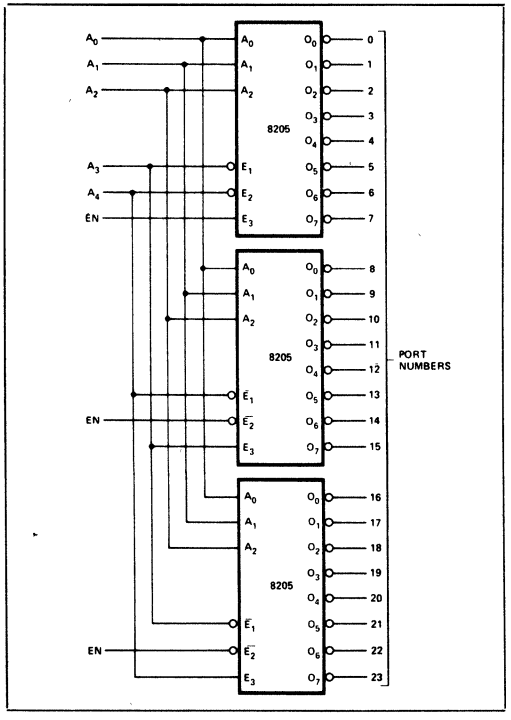


Figure 4. I/O Port Decoder

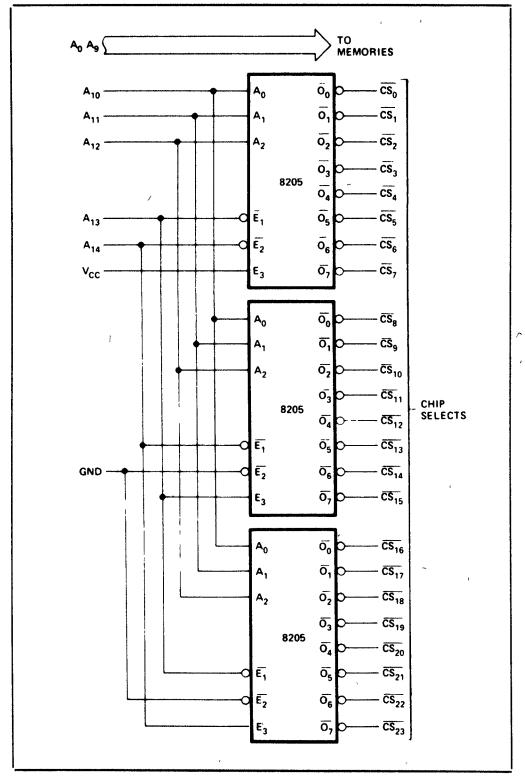


Figure 5. 24K Memory Interface

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias:
- Ceramic ..... −65°C to +125°C
- Plastic ..... −65°C to +75°C

Storage Temperature ..... −65°C to +160°C
All Output or Supply Voltages ..... −0.5 to +7 Volts
All Input Voltages ..... −1.0 to +5.5 Volts
Output Currents ..... 125 mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +75°C, V_{CC} = 5V ±5%)

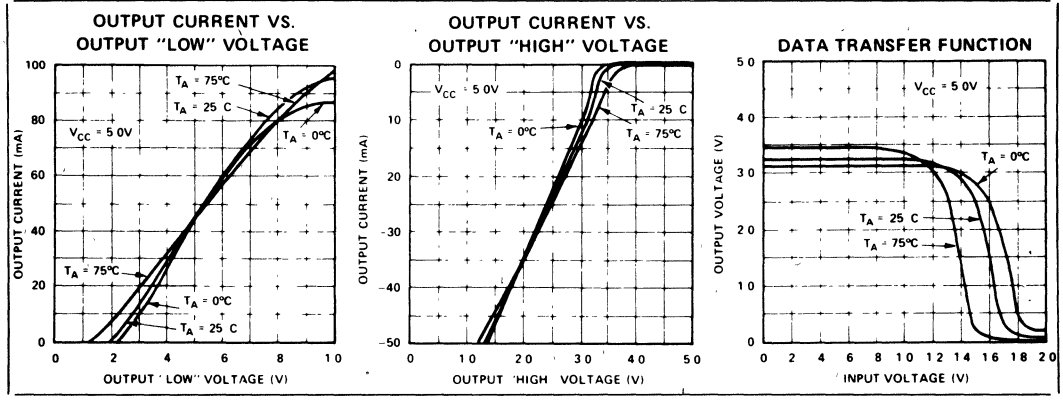
Symbol	Parameter	Limit Min.	Limit Max.	Unit	Test Conditions
I_F	INPUT LOAD CURRENT		−0 25	mA	V_{CC} = 5.25V, V_F = 0.45V
I_R	INPUT LEAKAGE CURRENT		10	µA	V_{CC} = 5 25V, V_R = 5.25V
V_C	INPUT FORWARD CLAMP VOLTAGE		−1 0	V	V_{CC} = 4 75V, I_C = −5.0 mA
V_{OL}	OUTPUT "LOW" VOLTAGE		0 45	V	V_{CC} = 4.75V, I_{OL} = 10.0 mA
V_{OH}	OUTPUT HIGH VOLTAGE	2 4		V	V_{CC} = 4.75V, I_{OH} = −1.5 mA
V_{IL}	INPUT "LOW" VOLTAGE		0 85	V	V_{CC} = 5 0V
V_{IH}	INPUT "HIGH" VOLTAGE	2 0		V	V_{CC} = 5.0V
I_{SC}	OUTPUT HIGH SHORT CIRCUIT CURRENT	−40	−120	mA	V_{CC} = 5.0V, V_{OUT} = 0V
V_{OX}	OUTPUT "LOW" VOLTAGE @ HIGH CURRENT		0 8	V	V_{CC} = 5.0V, I_{OX} = 40 mA
I_{CC}	POWER SUPPLY CURRENT		70	mA	V_{CC} = 5 25V

A.C. CHARACTERISTICS (T_A = 0°C to +75°C, V_{CC} = 5V ±5%; unless otherwise specified)

Symbol	Parameter		Max. Limit	Unit	Test Conditions
t_{++}	ADDRESS OR ENABLE TO OUTPUT DELAY		18	ns	
t_{-+}			18	ns	
t_{+-}			18	ns	
t_{--}			18	ns	
C_{IN} (1)	INPUT CAPACITANCE	P8205	4(typ.)	pF	f = 1 MHz, V_{CC} = 0V
		C8205	5(typ)	pF	V_{BIAS} = 2.0V, T_A = 25°C

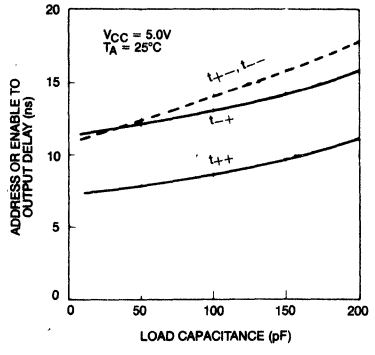
1 This parameter is periodically sampled and is not 100% tested

TYPICAL CHARACTERISTICS

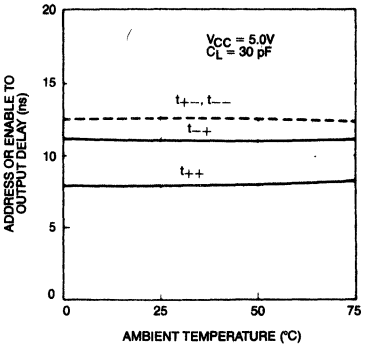


TYPICAL CHARACTERISTICS (Continued)

ADDRESS OR ENABLE TO OUTPUT DELAY VS. LOAD CAPACITANCE



ADDRESS OR ENABLE TO OUTPUT DELAY VS. AMBIENT TEMPERATURE



SWITCHING CHARACTERISTICS

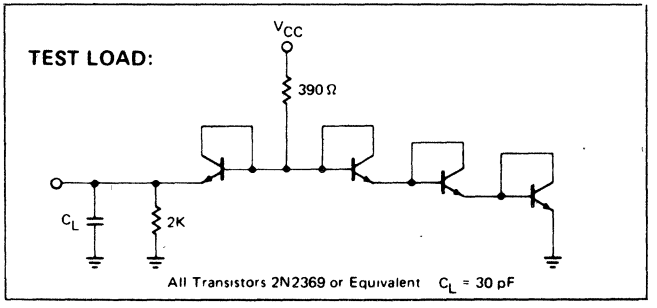
CONDITIONS OF TEST:

Input pulse amplitudes: 2.5V

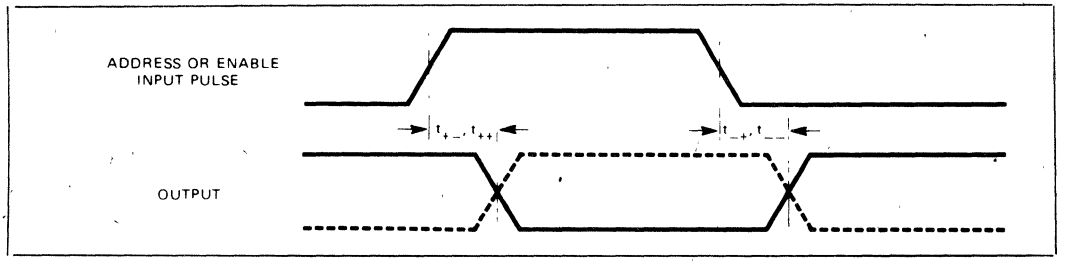
Input rise and fall times: 5 nsec between 1V and 2V

Measurements are made at 1.5V

TEST LOAD



WAVEFORMS



8212
8-BIT INPUT/OUTPUT PORT

- **Fully Parallel 8-Bit Data Register and Buffer**
- **Service Request Flip-Flop for Interrupt Generation**
- **Low Input Load Current — .25mA Max.**
- **Three State Outputs**
- **Outputs Sink 15mA**
- **3.65V Output High Voltage for Direct Interface to 8008, 8080A, or 8085A CPU**
- **Asynchronous Register Clear**
- **Replaces Buffers, Latches and Multiplexers in Microcomputer Systems**
- **Reduces System Package Count**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 8212 input/output port consists of an 8-bit latch with 3-state output buffers along with control and device selection logic. Also included is a service request flip-flop for the generation and control of interrupts to the microprocessor.

The device is multimode in nature. It can be used to implement latches, gated buffers or multiplexers. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with this device.

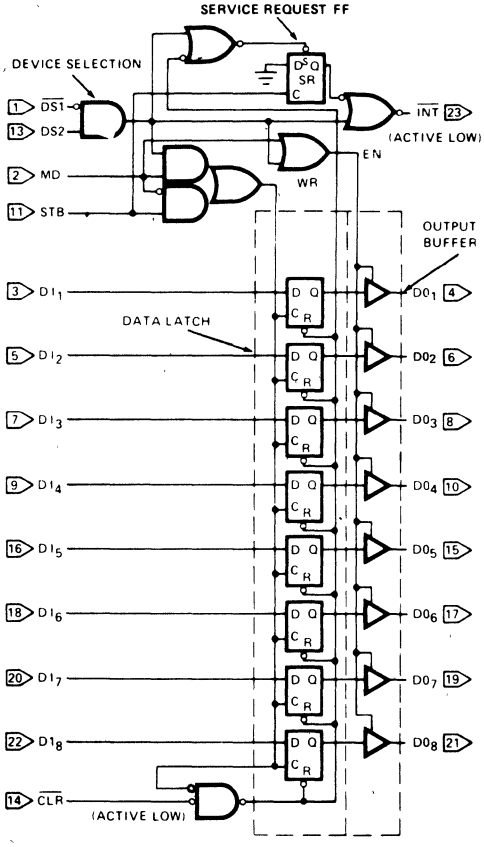
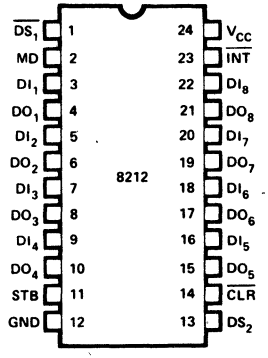


Figure 1. Logic Diagram



DI_1-DI_8	DATA IN
DO_1-DO_8	DATA OUT
$\overline{DS_1}$-DS_2	DEVICE SELECT
MD	MODE
STB	STROBE
$\overline{INT}$	INTERRUPT (ACTIVE LOW)
$\overline{CLR}$	CLEAR (ACTIVE LOW)

Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

Data Latch

The 8 flip-flops that make up the data latch are of a "D" type design. The output (Q) of the flip-flop will follow the data input (D) while the clock input (C) is high. Latching will occur when the clock (C) returns low.

The latched data is cleared by an asynchronous reset input ($\overline{CLR}$). (Note: Clock (C) Overrides Reset ($\overline{CLR}$).)

Output Buffer

The outputs of the data latch (Q) are connected to 3-state, non-inverting output buffers. These buffers have a common control line (EN); this control line either enables the buffer to transmit the data from the outputs of the data latch (Q) or disables the buffer, forcing the output into a high impedance state. (3-state)

The high-impedance state allows the designer to connect the 8212 directly onto the microprocessor bi-directional data bus.

Control Logic

The 8212 has control inputs $\overline{DS1}$, DS2, MD and STB. These inputs are used to control device selection, data latching, output buffer state and service request flip-flop.

$\overline{DS1}$, DS2 (Device Select)

These 2 inputs are used for device selection. When $\overline{DS1}$ is low and DS2 is high ($\overline{DS1} \cdot DS2$) the device is selected. In the selected state the output buffer is enabled and the service request flip-flop (SR) is asynchronously set.

MD (Mode)

This input is used to control the state of the output buffer and to determine the source of the clock input (C) to the data latch.

When MD is high (output mode) the output buffers are enabled and the source of clock (C) to the data latch is from the device selection logic ($\overline{DS1} \cdot DS2$).

When MD is low (input mode) the output buffer state is determined by the device selection logic ($\overline{DS1} \cdot DS2$) and the source of clock (C) to the data latch is the STB (Strobe) input.

STB (Strobe)

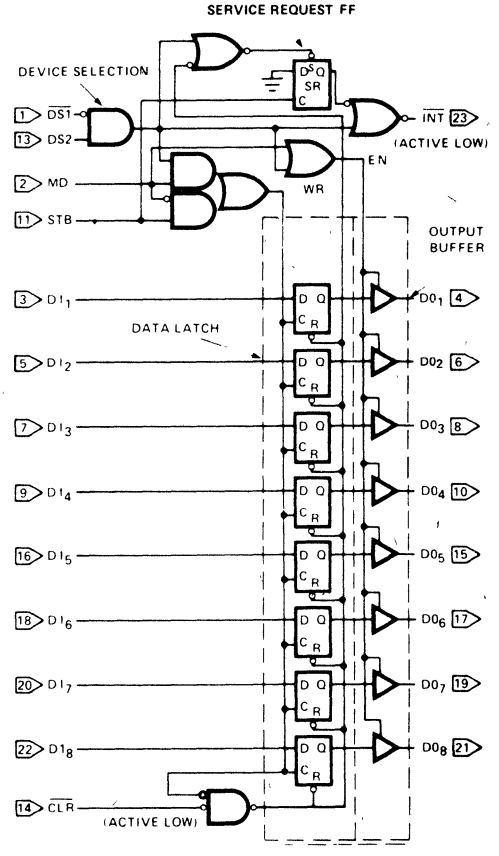
This input is used as the clock (C) to the data latch for the input mode MD = 0) and to synchronously reset the service request flip-flop (SR).

Note that the SR flip-flop is negative edge triggered.

Service Request Flip-Flop

The (SR) flip-flop is used to generate and control interrupts in microcomputer systems. It is asynchronously set by the $\overline{CLR}$ input (active low). When the (SR) flip-flop is set it is in the non-interrupting state.

The output of the (SR) flip-flop (Q) is connected to an inverting input of a "NOR" gate. The other input to the "NOR" gate is non-inverting and is connected to the device selection logic ($\overline{DS1} \cdot DS2$). The output of the "NOR" gate ($\overline{INT}$) is active low (interrupting state) for connection to active low input priority generating circuits.



STB	MD	(DS_1 DS_2)	DATA OUT EQUALS
0	0	0	3 STATE
1	0	0	3 STATE
0	1	0	DATA LATCH
1	1	0	DATA LATCH
0	0	1	DATA LATCH
1	0	1	DATA IN
0	1	1	DATA IN
1	1	1	DATA IN

$\overline{CLR}$ – RESETS DATA LATCH
SETS SR FLIP FLOP
(NO EFFECT ON OUTPUT BUFFER)

CLR	($\overline{DS_1}$ DS_2)	STB	*SR	INT
0	0	0	1	1
0	1	0	1	0
1	1	↓	0	0
1	1	0	1	0
1	0	0	1	1
1	1	↓	1	0

*INTERNAL SR FLIP FLOP

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias Plastic ..... 0°C to +70°C
Storage Temperature ..... -65°C to +160°C
All Output or Supply Voltages ..... -0.5 to +7 Volts
All Input Voltages ..... -1.0 to 5.5 Volts
Output Currents 100mA

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A=0°C to +75°C, V_{CC}= +5V ± 5%)

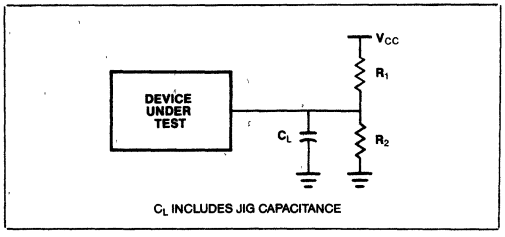
Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
I_F	Input Load Current, ACK, DS_2, CR, DI_1-DI_8 Inputs			-.25	mA	V_F = .45V
I_F	Input Load Current MD Input			-.75	mA	V_F = .45V
I_F	Input Load Current DS_1 Input			-1.0	mA	V_F = .45V
I_R	Input Leakage Current, ACK, DS, CR, DI_1-DI_8 Inputs			10	μA	$V_R \leq V_{CC}$
I_R	Input Leakage Current MO Input			30	μA	$V_R \leq V_{CC}$
I_R	Input Leakage Current DS_1 Input			40	μA	$V_R \leq V_{CC}$
V_C	Input Forward Voltage Clamp			-1	V	I_C = -5mA
V_{IL}	Input "Low" Voltage			.85	V	
V_{IH}	Input "High" Voltage	2.0			V	
V_{OL}	Output "Low" Voltage			.45	V	I_{OL} = 15mA
V_{OH}	Output "High" Voltage	3.65	4.0		V	I_{OH} = -1mA
I_{SC}	Short Circuit Output Current	-15		-75	mA	V_O = 0V, V_{CC} = 5V
$\|I_O\|$	Output Leakage Current High Impedance State			20	μA	V_O = .45V/5.25V_{CC}
I_{CC}	Power Supply Current		90	130	mA	

CAPACITANCE* (F = 1MHz, V_{BIAS} = 2.5V, V_{CC} = +5V, T_A = 25°C)

Symbol	Test	Limits	
		Typ.	Max.
C_{IN}	DS_1 MD Input Capacitance	9pF	12pF
C_{IN}	DS_2, $\overline{CLR}$, STB, DI_1-DI_8 Input Capacitance	5pF	9pF
C_{OUT}	DO_1-DO_8 Output Capacitance	8pF	12pF

*This parameter is sampled and not 100% tested.

A.C. TESTING LOAD CIRCUIT



SWITCHING CHARACTERISTICS

Conditions of Test

Input Pulse Amplitude = 2.5V
Input Rise and Fall Times 5ns
Between 1V and 2V Measurements made at 1.5V
with 15mA and 30pF Test Load

NOTE:

1.

Test	C_L*	R_1	R_2
t_{PD}, t_{WE}, t_R, t_S, t_C	30pF	300Ω	600Ω
t_E, ENABLE↑	30pF	10KΩ	1KΩ
t_E, ENABLE↓	30pF	300Ω	600Ω
t_E, DISABLE↑	5pF	300Ω	600Ω
t_E, DISABLE↓	5pF	10KΩ	1KΩ

*Includes probe and jig capacitance.

A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5V ± 5%)*

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
t_{PW}	Pulse Width	30			ns	
t_{PD}	Data to Output Delay			30	ns	Note 1
t_{WE}	Write Enable to Output Delay			40	ns	Note 1
t_{SET}	Data Set Up Time	15			ns	
t_H	Data Hold Time	20			ns	
t_R	Reset to Output Delay			40	ns	Note 1
t_S	Set to Output Delay			30	ns	Note 1
t_E	Output Enable/Disable Time			45	ns	Note 1
t_C	Clear to Output Delay			55	ns	Note 1

*Note: For extended Temperature EXPRESS use M8212 AC Electricals Parameters.

APPLICATIONS

Basic Schematic Symbols

Two examples of ways to draw the 8212 on system schematics—(1) the top being the detailed view showing pin numbers, and (2) the bottom being the symbolic view showing the system input or output as a system bus (bus containing 8 parallel lines). The output to the data bus is symbolic in referencing 8 parallel lines.

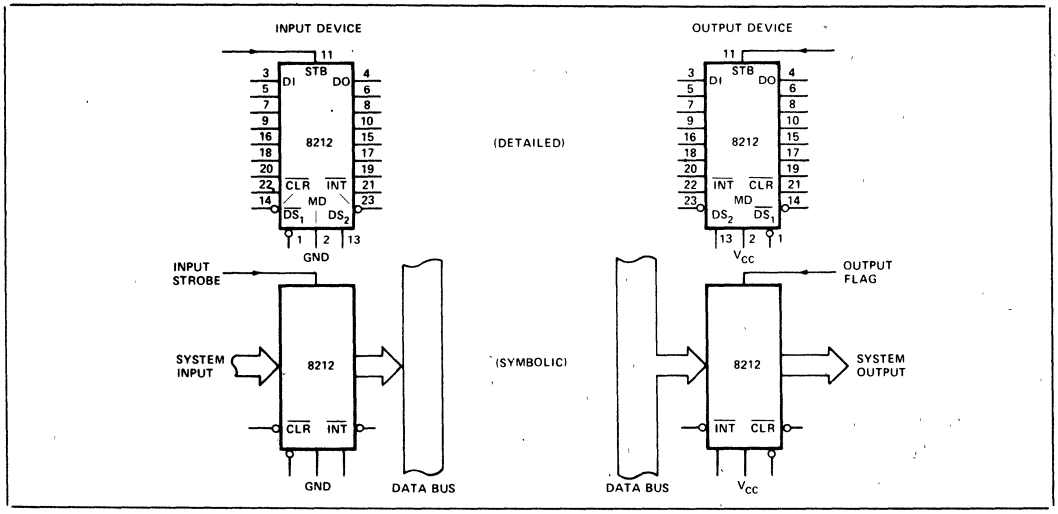


Figure 3. Basic Schematic Symbols

Gated Buffer (3-State)

The simplest use of the 8212 is that of a gated buffer. By tying the mode signal low and the strobe input high, the data latch is acting as a straight through gate. The output buffers are then enabled from the device selection logic $\overline{DS1}$ and DS2.

When the device selection logic is false, the outputs are 3-state.

When the device selection logic is true, the input data from the system is directly transferred to the output. The input data load is 250 micro amps. The output data can sink 15 milli amps. The minimum high output is 3.65 volts.

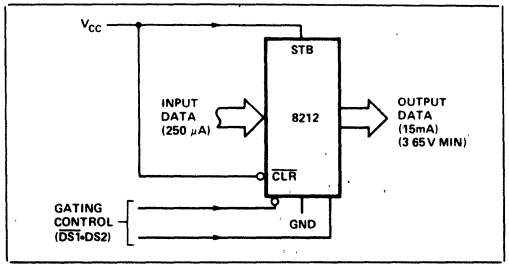


Figure 4. Gated Buffer

Bi-Directional Bus Driver

A pair of 8212's wired (back-to-back) can be used as a symmetrical drive, bi-directional bus driver. The devices are controlled by the data bus input control which is connected to $\overline{DS1}$ on the first 8212 and to DS2 on the second. One device is active, and acting as a straight through buffer the other is in 3-state mode. This is a very useful circuit in small system design.

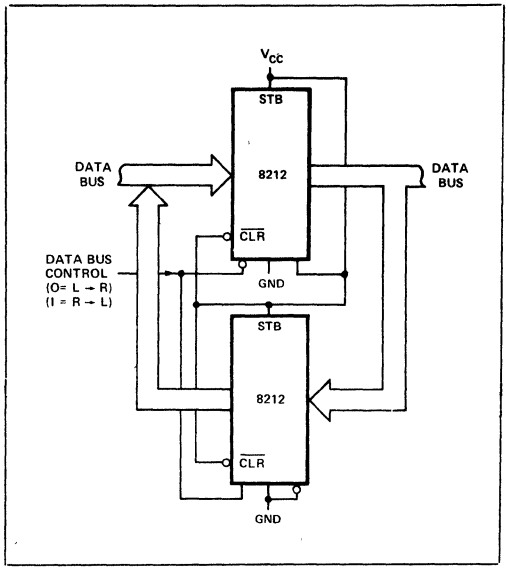


Figure 5. Bidirectional Bus Driver

Interrupting Input Port

This use of an 8212 is that of a system input port that accepts a strobe from the system input source, which in turn clears the service request flip-flop and interrupts the processor. The processor then goes through a service routine, identifies the port, and causes the device selection logic to go true — enabling the system input data onto the data bus.

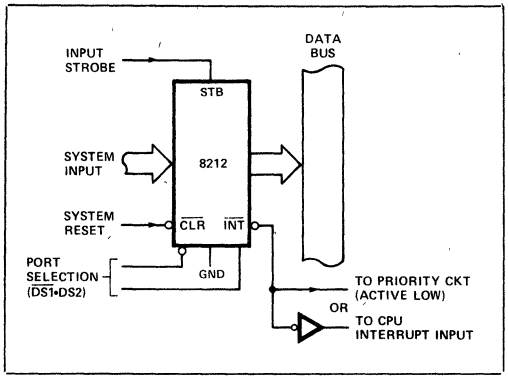


Figure 6. Interrupting Input Port

Interrupt Instruction Port

The 8212 can be used to gate the interrupt instruction, normally RESTART instructions, onto the data bus. The device is enabled from the interrupt acknowledge signal from the microprocessor and from a port selection signal. This signal is normally tied to ground. ($\overline{DS1}$ could be used to multiplex a variety of interrupt instruction ports onto a common bus).

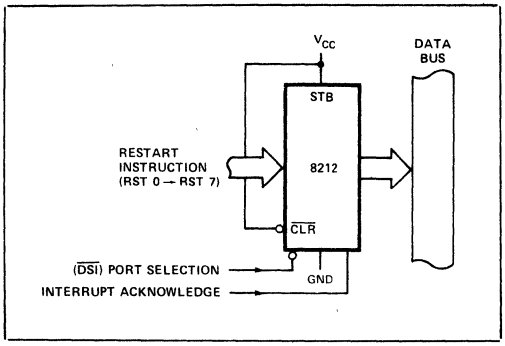


Figure 7. Interrupt Instruction Port

Output Port (With Hand-Shaking)

The 8212 can be used to transmit data from the data bus to a system output. The output strobe could be a hand-shaking signal such as "reception of data" from the device that the system is outputting to. It in turn, can interrupt the system signifying the reception of data. The selection of the port comes from the device selection logic. ($\overline{DS1}$ • DS2)

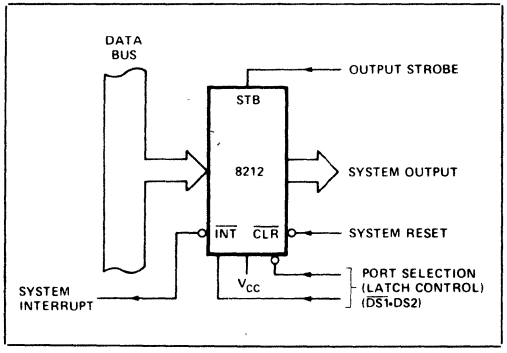
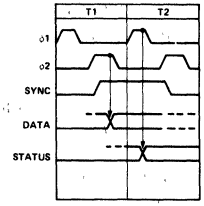
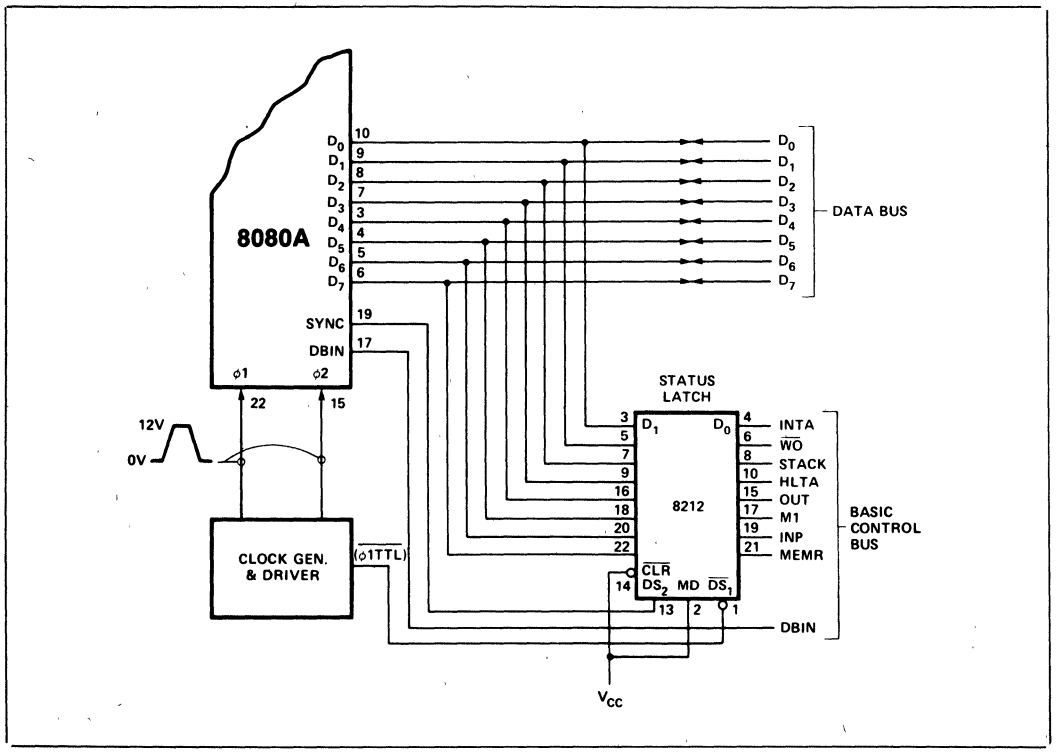


Figure 8. Output Port

808A Status Latch

Here the 8212 is used as the status latch for an 8080A microcomputer system. The input to the 8212 latch is directly from the 8080A data bus. Timing shows that when the SYNC signal is true, which is connected to the DS2 input and the phase 1 signal is true, which is a TTL level coming from the clock generator; then, the status data will be latched into the 8212.



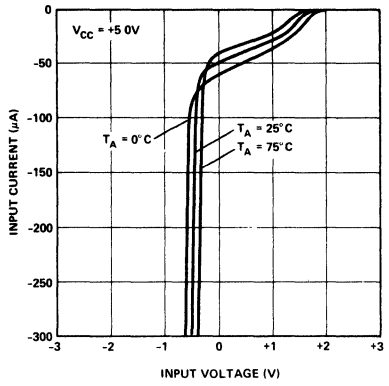


Note: The mode signal is tied high so that the output on the latch is active and enabled all the time.

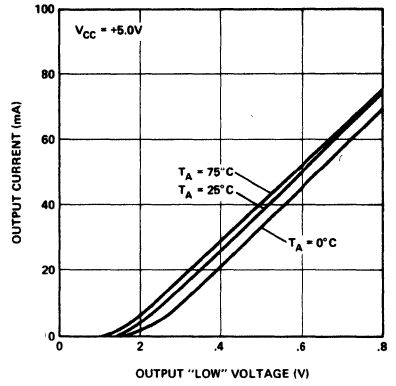
It is shown that the two areas of concern are the bi-directional data bus of the microprocessor and the control bus.

TYPICAL CHARACTERISTICS

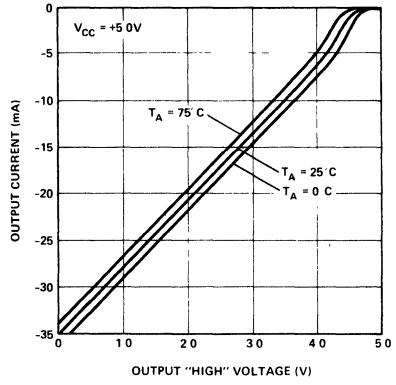
INPUT CURRENT VS. INPUT VOLTAGE



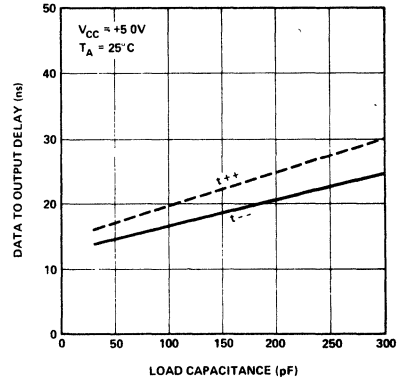
OUTPUT CURRENT VS. OUTPUT "LOW" VOLTAGE



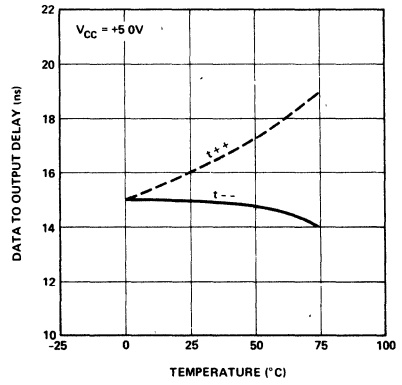
OUTPUT CURRENT VS. OUTPUT "HIGH" VOLTAGE



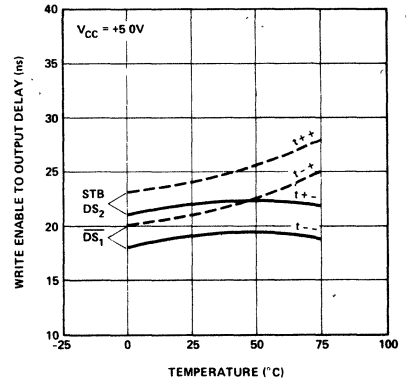
DATA TO OUTPUT DELAY VS. LOAD CAPACITANCE



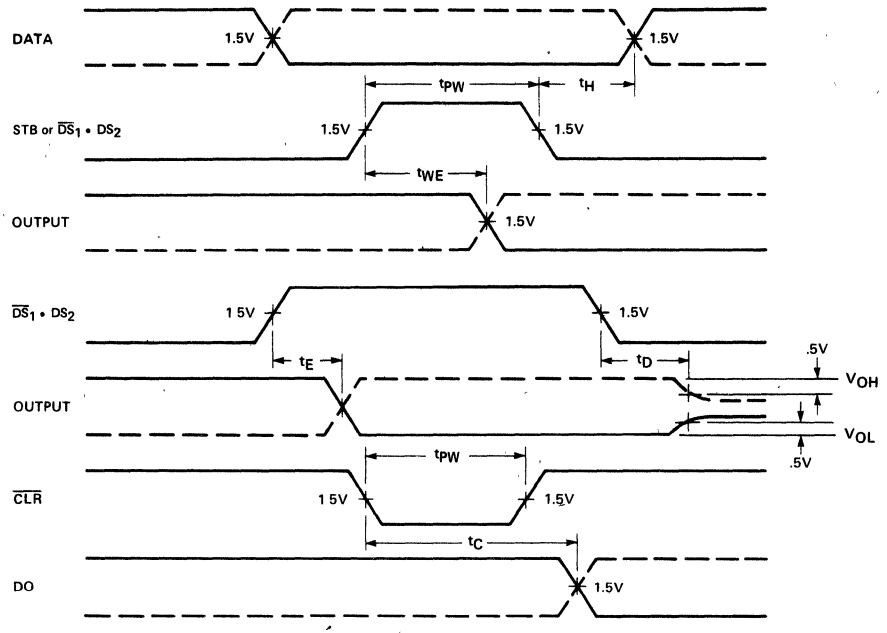
DATA TO OUTPUT DELAY VS. TEMPERATURE

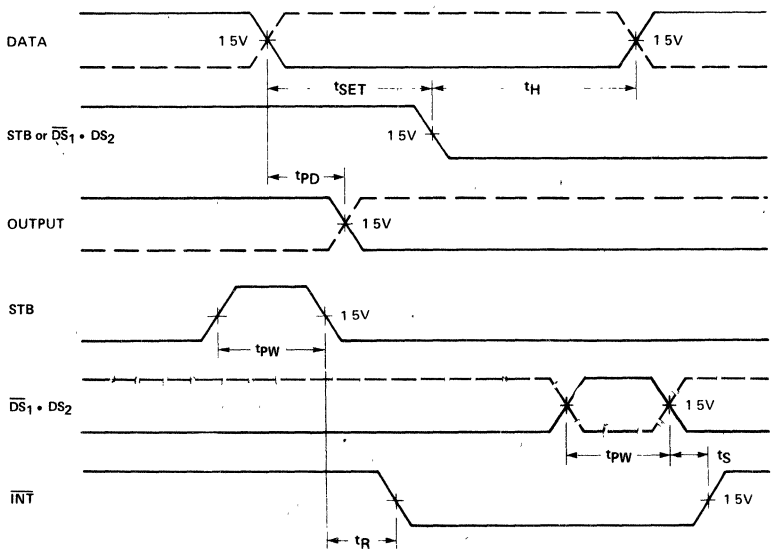


WRITE ENABLE TO OUTPUT DELAY VS. TEMPERATURE



WAVEFORMS


DATA
1.5V
1.5V
tPW
tH
STB or $\overline{DS_1} \cdot DS_2$
1.5V
1.5V
tWE
OUTPUT
1.5V
$\overline{DS_1} \cdot DS_2$
1.5V
1.5V
tE
tD
.5V
VOH
OUTPUT
VOL
.5V
tPW
$\overline{CLR}$
1.5V
1.5V
tC
DO
1.5V


DATA
1.5V
1.5V
tSET
tH
STB or $\overline{DS_1} \cdot DS_2$
1.5V
tPD
OUTPUT
1.5V
STB
1.5V
tPW
$\overline{DS_1} \cdot DS_2$
1.5V
tPW
tS
$\overline{INT}$
1.5V
tR

8216/8226
4-BIT PARALLEL BIDIRECTIONAL BUS DRIVER

- **Data Bus Buffer Driver for 8080 CPU**
- **Low Input Load Current — 0.25 mA Maximum**
- **High Output Drive Capability for Driving System Bus**
- **3.65V Output High Voltage for Direct Interface to 8080 CPU**
- **3-State Outputs**
- **Reduces System Package Count**
- **Available in EXPRESS**
  - **Standard Temperature Range**

The 8216/8226 is a 4-bit bidirectional bus driver/receiver. All inputs are low power TTL compatible. For driving MOS, the DO outputs provide a high 3.65V V_{OH}, and for high capacitance terminated bus structures, the DB outputs provide a high 50 mA I_{OL} capability. A non-inverting (8216) and an inverting (8226) are available to meet a wide variety of applications for buffering in microcomputer systems.

*Note: The specifications for the 3216/3226 are identical with those for the 8216/8226.

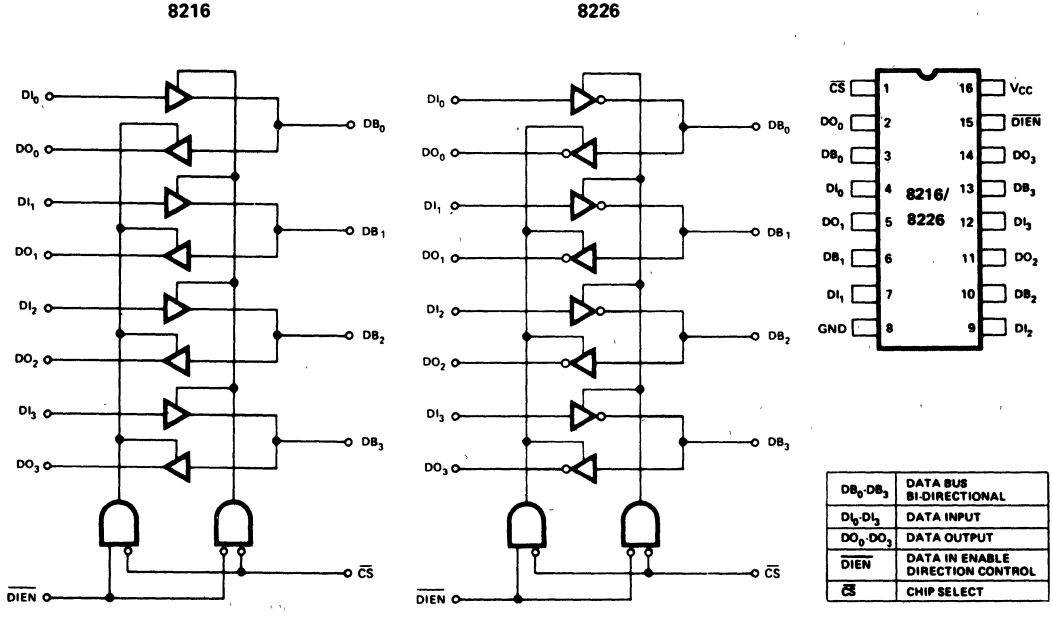


Figure 1. Block Diagrams

Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

Microprocessors like the 8080 are MOS devices and are generally capable of driving a single TTL load. The same is true for MOS memory devices. While this type of drive is sufficient in small systems with few components, quite often it is necessary to buffer the microprocessor and memories when adding components or expanding to a multi-board system.

The 8216/8226 is a four bit bi-directional bus driver specifically designed to buffer microcomputer system components.

Bidirectional Driver

Each buffered line of the four bit driver consists of two separate buffers that are tri-state in nature to achieve direct bus interface and bi-directional capability. On one side of the driver the output of one buffer and the input of another are tied together (DB), this side is used to interface to the system side components such as memories, I/O, etc., because its interface is direct TTL compatible and it has high drive (50mA). On the other side of the driver the inputs and outputs are separated to provide maximum flexibility. Of course, they can be tied together so that the driver can be used to buffer a true bi-directional bus such as the 8080 Data Bus. The DO outputs on this side of the driver have a special high voltage output drive capability (3.65V) so that direct interface to the 8080 and 8008 CPUs is achieved with an adequate amount of noise immunity (350mV worst case).

Control Gating $\overline{DIEN}$, $\overline{CS}$

The $\overline{CS}$ input is actually a device select. When it is "high" the output drivers are all forced to their high-impedance state. When it is at "zero" the device is selected (enabled) and the direction of the data flow is determined by the $\overline{DIEN}$ input.

The $\overline{DIEN}$ input controls the direction of data flow (see Figure 3) for complete truth table. This direction control is accomplished by forcing one of the pair of buffers into its high impedance state and allowing the other to transmit its data. A simple two gate circuit is used for this function.

The 8216/8226 is a device that will reduce component count in microcomputer systems and at the same time enhance noise immunity to assure reliable, high performance operation.

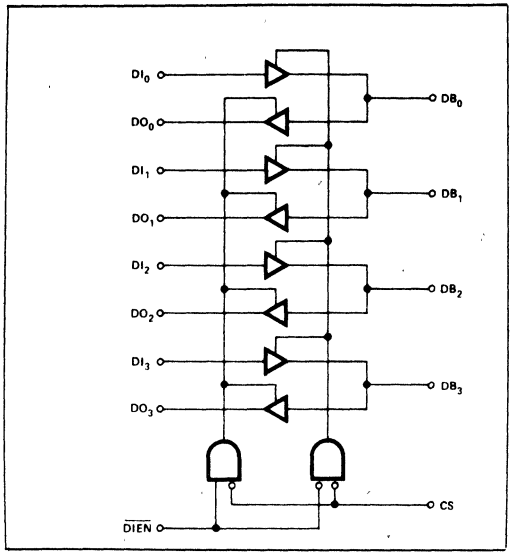
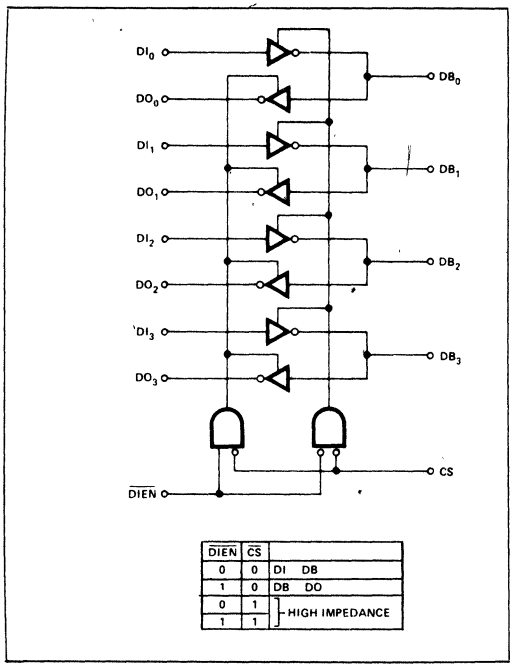


Figure 3a. 8216 Logic Diagram



$\overline{DIEN}$	$\overline{CS}$	
0	0	DI DB
1	0	DB DO
0	1	HIGH IMPEDANCE
1	1	

Figure 3b. 8226 Logic Diagram

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
All Output and Supply Voltages	-0.5V to +7V
All Input Voltages	-1.0V to +5.5V
Output Currents	125 mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5V ± 5%)

Symbol	Parameter		Limits Min.	Typ.	Max.	Unit	Conditions
I_{F1}	Input Load Current $\overline{DIEN}$, $\overline{CS}$			-0.15	-.5	mA	V_F = 0.45
I_{F2}	Input Load Current All Other Inputs			-0.08	-.25	mA	V_F = 0.45
I_{R1}	Input Leakage Current $\overline{DIEN}$, $\overline{CS}$				80	μA	V_R = 5.25V
I_{R2}	Input Leakage Current DI Inputs				40	μA	V_R = 5.25V
V_C	Input Forward Voltage Clamp				-1	V	I_C = -5mA
V_{IL}	Input "Low" Voltage				.95	V	
V_{IH}	Input "High" Voltage		2.0			V	
$\|I_O\|$	Output Leakage Current (3-State)	DO DB			20 100	μA	V_O = .45V/5.25V_{CC}
I_{CC}	Power Supply Current	8216		95	130	mA	
		8226		85	120	mA	
V_{OL1}	Output "Low" Voltage			0.3	.45	V	DO Outputs I_{OL}=15mA DB Outputs I_{OL}=25mA
V_{OL2}	Output "Low" Voltage	8216		0.5	.6	V	DB Outputs I_{OL}=55mA
		8226		0.5	.6	V	DB Outputs I_{OL}=50mA
V_{OH1}	Output "High" Voltage		3.65	4.0		V	DO Outputs I_{OH} = -1mA
V_{OH2}	Output "High" Voltage		2.4	3.0		V	DB Outputs I_{OH} = -10mA
I_{OS}	Output Short Circuit Current		-15 -30	-35 -75	-65 -120	mA mA	DO Outputs $V_O \cong 0V$, DB Outputs V_{CC}=5.0V

NOTE:

Typical values are for T_A = 25°C, V_{CC} = 5.0V.

CAPACITANCE[5] (V_{BIAS} = 2.5V, V_{CC} = 5.0V, T_A = 25°C, f = 1 MHz)

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max.	
C_{IN}	Input Capacitance		4	8	pF
C_{OUT1}	Output Capacitance		6	10	pF
C_{OUT2}	Output Capacitance		13	18	pF

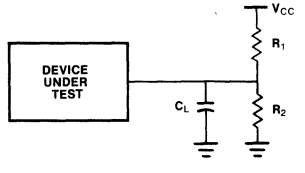
A.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = +5V ± 5%)

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ.[1]	Max.		
T_{PD1}	Input to Output Delay DO Outputs		15	25	ns	C_L=30pF, R_1=300Ω R_2=600Ω
T_{PD2}	Input to Output Delay DB Outputs 8216		19	30	ns	C_L=300pF, R_1=90Ω
	8226		16	25	ns	R_2 = 180Ω
T_E	Output Enable Time 8216		42	65	ns	(Note 2)
	8226		36	54	ns	(Note 3)
T_D	Output Disable Time		16	35	ns	(Note 4)

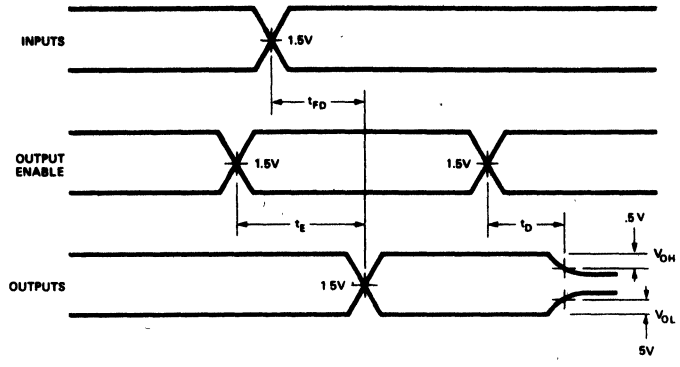
NOTE:
Input pulse amplitude of 2.5V.
Input rise and fall times of 5 ns between 1 and 2 volts.
Output loading is 5 mA and 10 pF.
Speed measurements are made at 1.5 volt levels.

NOTES:
1. Typical values are for T_A = 25°C, V_{CC} = 5.0V.
2. DO Outputs, C_L = 30pF, R_1 = 300/10 KΩ, R_2 = 180/1KΩ; DB Outputs, C_L = 300pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
3. DO Outputs, C_L = 30pF, R_1 = 300/10 KΩ, R_2 = 600/1K; DB Outputs, C_L = 300pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
4. DO Outputs, C_L = 5pF, R_1 = 300/10 KΩ, R_2 = 600/1 KΩ; DB Outputs, C_L = 5pF, R_1 = 90/10 KΩ, R_2 = 180/1 KΩ.
5. This parameter is periodically sampled and not 100% tested.

A.C. TESTING LOAD CIRCUIT



WAVEFORM



AFN-00733C

PRELIMINARY

8218/8219
BIPOLAR MICROCOMPUTER BUS CONTROLLERS FOR MCS-80® AND MCS-85® FAMILIES

- **8218 for Use in MCS-80® Systems**
- **8219 for Use inMCS-85® Systems**
- **Coordinates the Sharing of a Common Bus Between Several CPU's**
- **Reduces Component Count in Multimaster Bus Arbitration Logic**
- **Single +5 Volt Power Supply**
- **28 Pin Package**

The 8218 and 8219 Microcomputer Bus Controllers consist of control logic which allows a bus master device such as a CPU or DMA channel to interface with other masters on a common bus, sharing memory and I/O devices. The 8218 and 8219 consist of:

1. Bus Arbitration Logic which operates from the Bus Clock ($\overline{BCLK}$) and resolves bus contention between devices sharing a common bus.
2. Timing Logic which when initiated by the bus arbitration logic generates timing signals for the memory and I/O command lines to guarantee set-up and hold times of the address/data lines onto the bus. The timing logic also signals to the bus arbitration logic when the current data transfer is completed and the bus is no longer needed.
3. Output Drive Logic which contains the logic and output drivers for the memory and I/O command lines.

An external RC time constant is used with the timing logic to generate the guaranteed address set-up and hold times on the bus. The 8219 can interface directly to the 8085A CPU and the 8218 interfaces to the 8080A CPU chip and the 8257 DMA controller.

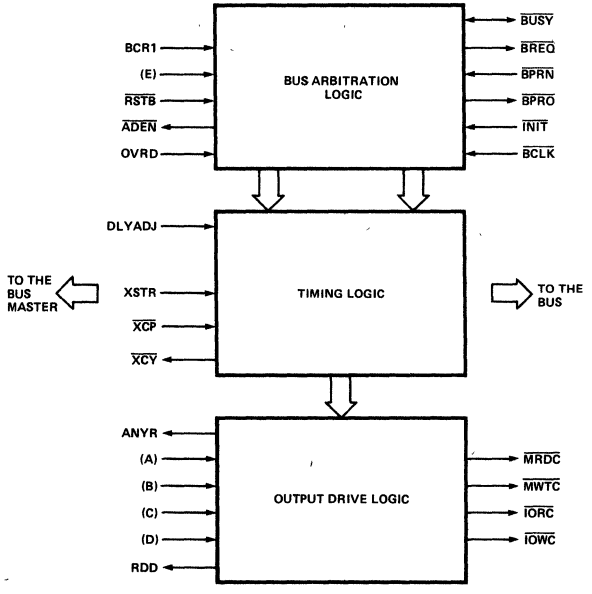
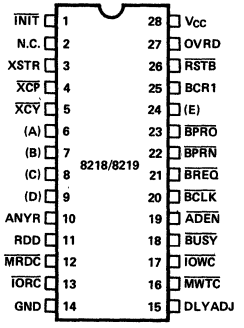


Figure 1. Block Diagram



	8218	8219
(A)	$\overline{IOWR}$	IO/$\overline{M}$
(B)	$\overline{MWTR}$	$\overline{WR}$
(C)	$\overline{IORR}$	$\overline{RD}$
(D)	$\overline{MRDR}$	$\overline{ASRQ}$
(E)	$\overline{BCR2}$	BCR2

N.C. = NO CONNECT

Figure 2. Pin Configuration

Table 1. Pin Description

Signals Interfaced Directly to the System Bus		
Symbol	**Type**	**Name and Function**
$\overline{BREQ}$	O	**Bus Request:** The Bus Request is used with a central parallel priority resolution circuit. It indicates that the device needs to access the bus for one or more data transfers. It is synchronized with the Bus Clock.
$\overline{BUSY}$	I/O	**Bus Busy:** Bus Busy indicates to all master devices on the bus that the bus is in use. It inhibits any other device from getting the bus. It is synchronized with Bus Clock.
$\overline{BCLK}$	I	**Bus Clock:** The negative edge of Bus Clock is used to synchronize the bus contention resolution circuit asynchronously to the CPU clock. It has 100ns min. period, 35%–65% duty cycle. It may be slowed, single stepped or stopped.
$\overline{BPRN}$	I	**Bus Priority In:** The Bus Priority In indicates to a device that no device of a higher priority is requesting the bus. It is synchronous with the Bus clock.
$\overline{BPRO}$	O	**Bus Priority Out:** The Bus Priority Out is used with serial priority resolution circuits. Priority may be transferred to the next lower in priority as $\overline{BPRN}$.
$\overline{INIT}$	I	**Initialize:** The Initialize resets the 8218/8219 to a known internal state.
$\overline{MRDC}$	O	**Memory Read Control:** The Memory Read Control indicates that the Master is requesting a read operation from the addressed location. It is asynchronous to the Bus Clock.
$\overline{MWTC}$	O	**Memory Write Control:** The Memory Write Control indicates that data and an address have been placed on the bus by the Master and the data is to be deposited at that location. It is asynchronous to the Bus Clock.
$\overline{IORC}$	O	**I/O Read Control:** The I/O Read Control indicates that the Master is requesting a read operation from the I/O device addressed. It is asynchronous to the Bus Clock.
$\overline{IOWC}$	O	**I/O Write Control:** The I/O Write Control indicates that Data and an I/O device address has been placed on the bus by the Master and the data is to be deposited to the I/O device. It is asynchronous to the Bus Clock.
Signals Generated or Received by the Bus Master		
BCR1/$\overline{BCR2}$	I	**Bus Control Request:** Bus Control Request 1 or Bus Control Request 2 indicate to the 8218/8219 that the Master device is making a request to control the bus. BCR2 is active low in the 8218 ($\overline{BCR2}$). BCR2 is active high in the 8219.
RSTB	I	**Request Strobe:** Request Strobe latches the status of BCR1 and BCR2 into the 8218/8219. The strobe is active low in the 8218 and negative edge triggered in the 8219.

Signals Generated or Received by the Bus Master (Continued)		
Symbol	**Type**	**Name and Function**
$\overline{ADEN}$	O	**Address and Data Enable:** Address and Data Enable indicates the Master has control of the bus. It is often used to enable Address and Data Buffers on the bus. It is synchronous with Bus Clock.
RDD	O	**Read Data:** Read Data controls the direction of the bi-directional data bus drivers. It is asynchronous to the Bus Clock. A high on RDD indicates a read mode by the master.
OVRD	I	**Override:** Override inhibits automatic deselect between transfers caused by a higher priority bus request. May be used for consecutive data transfers such as read-modify-write operations. It is asynchronous to the Bus Clock.
XSTR	I	**Transfer Start Request:** Transfer Start Request indicates to the 8218/8219 that a new data transfer cycle is requested to start. It is raised for each new word transfer in a multiple data word transfer. It is asynchronous to the Bus Clock.
$\overline{XCP}$	I	**Transfer Complete:** Transfer Complete indicates to the 8218/8219 that the data has been received by the slave device in a write cycle or transmitted by the slave and received by master in a read cycle. It is asynchronous to the Bus Clock.
$\overline{XCY}$	O	**Data Transfer:** Indicates that a data transfer is in progress. It is asynchronous to the Bus Clock.
$\overline{WR}$, $\overline{RD}$, IO/$\overline{M}$	I	**Write, Read, IO/Memory:** WRITE, READ, IO/Memory are the control request inputs used by the 8085 and are internally decoded by the 8219 to produce the request signals $\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$. They are asynchronous to the Bus Clock. (8219 only)
$\overline{ASRQ}$	I	**Asynchronous Bus Request:** Can be used for interrupt status from the 8085. Acts like a level sensitive asynchronous bus request—no RSTB needed. It is asynchronous to the Bus Clock. (8219 only)
$\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$	I	**Memory Read Request, Memory Write Request, I/O Read Request, or I/O Write Request:** Indicate that address and data have been placed on the bus and the appropriate request is being made to the addressed device. Only one of these inputs should be active at any one time. They are synchronous to the Bus Clock. (8218 only)
ANYR	O	**Any Request:** Any Request is the logical OR of the active state of $\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$. It may be tied to XSTR when the rising edge of ANYR is used to initiate a transfer.
DLYADJ	I	**Delay Adjust:** Delay Adjust is used for connection of an external capacitor and resistor to ground to adjust the required set-up and hold time of address to control signal.

FUNCTIONAL DESCRIPTION

The 8218/8219 is a bipolar Bus Control Chip which reduces component count in the interface between a master device and the system Bus. (Master device: 8080, 8085, 8257 (DMA).)

The 8218 and 8219 serve three major functions:

1. Resolve bus contention.
2. Guarantee set-up and hold time of address/data lines to I/O and Memory read/write control signals (adjustable by external capacitor).
3. Provide sufficient drive on all bus command lines.

Bus Arbitration Logic

Bus Arbitration Logic activity begins when the Master makes a request for use of the bus on BCR1 or $\overline{BCR2}$. The request is strobed in by $\overline{RSTB}$. Following the next two falling edges of the bus clock (BCLK) the 8218/8219 outputs a bus request ($\overline{BREQ}$) and forces Bus Priority Out inactive (BPRO). See Figures 1a and 1b.

$\overline{BREQ}$ is used for requesting the bus when priority is decided by a parallel priority resolver circuit.

$\overline{BPRO}$ is used to allow lower priority devices to gain the bus when a serial priority resolving structure is used. $\overline{BPRO}$ would go to $\overline{BPRN}$ of the next lower priority Master.

When priority is granted to the Master (a low on $\overline{BPRN}$ and a high on $\overline{BUSY}$) the Master outputs a $\overline{BUSY}$ signal on the next falling edge of $\overline{BCLK}$. The $\overline{BUSY}$ signal locks the master onto the bus and prohibits the enable of any other masters onto the bus.

At the same time $\overline{BUSY}$ goes active, Address and Data Enable ($\overline{ADEN}$) goes active signifying that the Master has control of the bus. $\overline{ADEN}$ is often used to enable the bus drivers.

The Bus will be released only if the master loses priority; is not in the middle of a transfer, and Override is not active or, if the Master stops requesting the bus, is not in the middle of a data transfer, and Override is not active. $\overline{ADEN}$ then goes inactive.

Provision has been made in the 8218 to allow bus-synchronous requests. This mode is activated when BCR1, $\overline{BCR2}$ and $\overline{RSTB}$ are all low. This action asynchronously sets the synchronization flip flop (FF2) in Figure 3a.

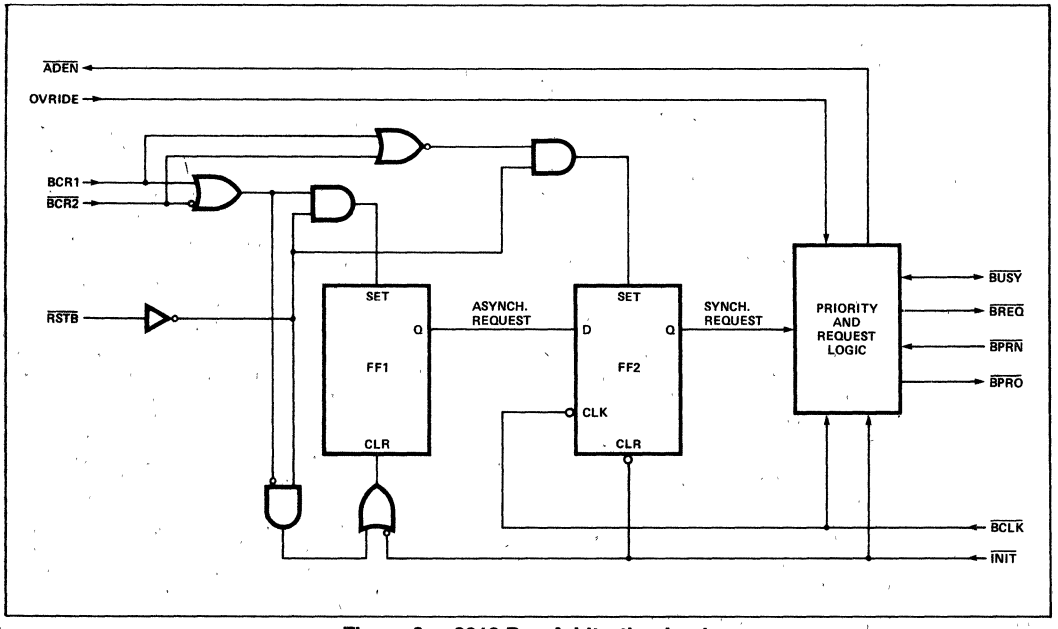


Figure 3a. 8218 Bus Arbitration Logic

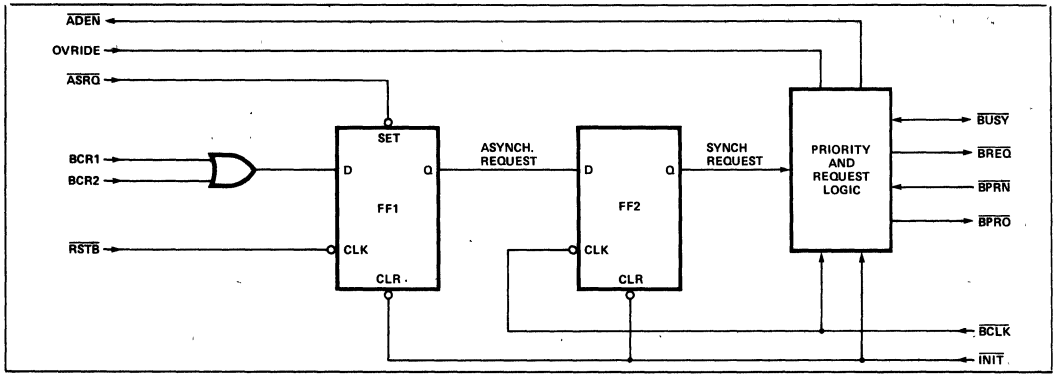


Figure 3b. 8219 Bus Arbitration Logic

Timing Logic

Timing Logic activity begins with the rising edge of XSTR (Transfer Start Request) or with $\overline{ADEN}$ going active, whichever occurs second. This action causes $\overline{XCY}$ (Transfer Cycle) to go active. 50-200ns later (depending on resistance and capacitance at DLYADJ) the appropriate Control Outputs will go active if the control input is active.

XSTR can be raised after the command goes active in the current transfer cycle so that a new transfer can be initiated immediately after the current transfer is complete.

A negative going edge on $\overline{XCP}$ (Transfer Complete) will cause the Control Outputs ($\overline{MRDC}$, etc.) to go inactive. 50-200ns later (depending on capacitance at DLYADJ) $\overline{XCY}$ will go inactive indicating the transfer cycle is completed.

Additional logic within the 8218/8219 guarantees that if a transfer cycle is started ($\overline{XCY}$ is active), but the bus is not requested ($\overline{BREQ}$ is inactive) and there is no command request input (ANYR is output low), then the transfer cycle will be cleared. This allows the bus to be released in applications where advanced bus requests are generated but the processor enters a HALT mode.

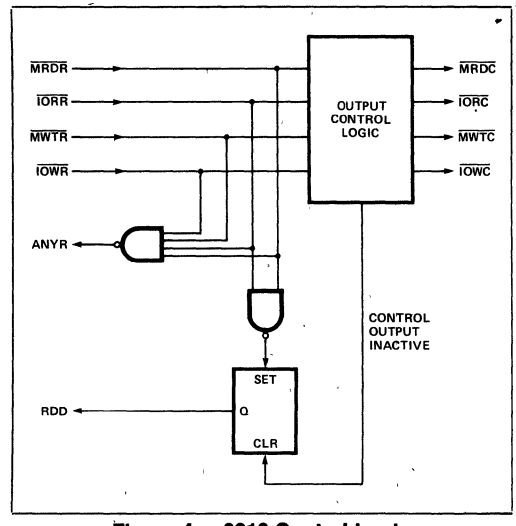


Figure 4a. 8218 Control Logic

Control Logic

The control outputs are generated in the 8219 by decoding the 8085 system control outputs (i.e., RD, WR, IO/M) or in the 8218 by directly buffering the control inputs to the control outputs for use in an 8080 or DMA system (see Figures 4a and 4b). The control outputs may be held high (inactive) by the Timing Logic. Also the control outputs are enabled when the Master gains control of the bus and disabled when control is relinquished.

The Control Logic also has two other outputs, ANYR (Any Request) and RDD (Read Data). ANYR goes high (active) if any control requests (IOWR, etc.) are active. RDD controls the direction of the Masters Bi-directional Data Bus Drivers. The Bus Driver will always be in the Write mode (RDD = Low) except from the start of a Read Control Request to 25 to 70ns after $\overline{XCP}$ is activated.

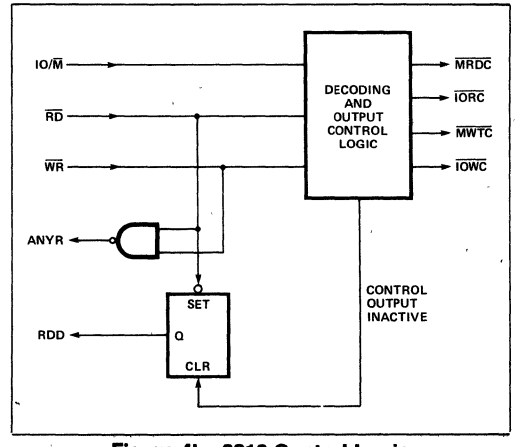


Figure 4b. 8219 Control Logic

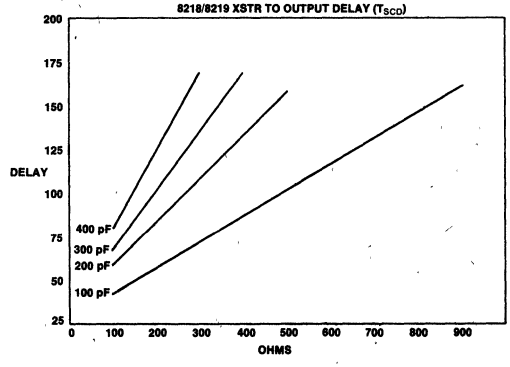
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to +150°C
Supply Voltage (V_{CC}) ..... -0.5V to +7V
Input Voltage ..... -1.0V to V_{CC} + 0.25V
Output Current 100mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_C	Input Clamp Voltage			-1.0	V	V_{CC} = 0.0V, I_C = −5 mA
I_F	Input Load Current					
	$\overline{MRDR}$/$\overline{INTA}$/$\overline{MWTR}$/$\overline{WR}$					V_{CC} = 5.25V
	$\overline{IORR}$/$\overline{RD}$, IOWR/IO/$\overline{M}$			-0.5	mA	V_F = 0.45V
	Other			-0.5	mA	
I_R	Input Leakage Current			100	μA	V_{CC} = 5.25 V_R = 5.25
V_{TH}	Input Threshold Voltage	0.8		2.0	V	V_{CC} = 5V
I_{CC}	Power Supply Current		200	240	mA	V_{CC} = 5.25V
V_{OL}	Output Low Voltage					V_{CC} = 4.75
	$\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$			0.45	V	I_{OL} = 32mA
	$\overline{BREQ}$, $\overline{BUSY}$			0.45	V	I_{OL} = 20mA
	$\overline{XCY}$, $\overline{RDD}$, $\overline{ADEN}$			0.45	V	I_{OL} = 16mA
	$\overline{BPRO}$, ANYR			0.45	V	I_{OL} = 3.2mA
V_{OH}	Output High Voltage					V_{CC} = 4.75V
	$\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$	2.4				I_{OH} = -2mA
	All Other Outputs	2.4				I_{OH} = -400μA
I_{OS}	Short Circuit Output Current	-10		-90	mA	V_{CC} = 5.25V, V_O = 0V
$I_{O\ (OFF)}$	Tri-State Output Current			-100	μA	V_{CC} = 5.25V, V_O = 0.45V
				+100	μA	V_{CC} = 5.25V, V_O = 5.25V
C_{IN}	Input Capacitance Except Busy		10	20	pF	
C_{IO}	Input Capacitance Busy		25	35	pF	

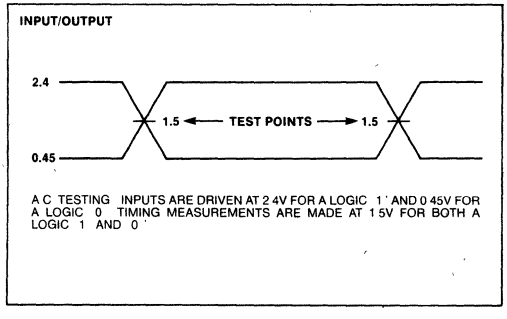


One Shot Delay Versus Delay Adjust Capacitance And Resistance

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

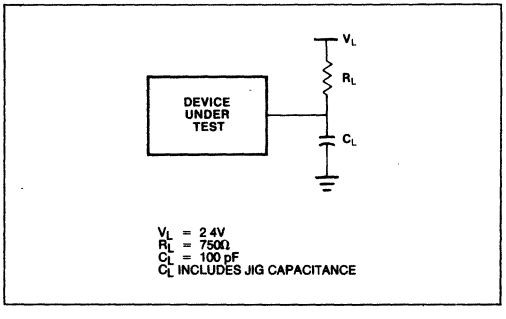
Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
t_{BCY}	Bus Clock Cycle Time	100			ns	35% to 65% Duty Cycle
t_{PW}	Bus Clock Pulse Width	35		0.65 t_{BCY}	ns	
t_{RQS}	$\overline{RSTB}$ to $\overline{BCLK}$ Set-Up Time	25			ns	
t_{CSS}	BCR_1 and $\overline{BCR_2}$ to $\overline{RSTB}$ Set-Up Time	15			ns	
t_{CSH}	BCR_1 and $\overline{BCR_2}$ to $\overline{RSTB}$ Hold Time	15			ns	
t_{RQD}	$\overline{BCLK}$ to $\overline{BREQ}$ Delay			35	ns	
t_{PRNS}	$\overline{BPRN}$ to $\overline{BCLK}$ Set-Up Time	23			ns	
t_{BNO}	$\overline{BRPN}$ to $\overline{BPRO}$ Delay			30	ns	
t_{BYD}	$\overline{BCLK}$ to $\overline{BUSY}$ Delay			55	ns	
t_{CAD}	$\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$ to ANYR Delay			30	ns	
t_{SXD}	XSTR to $\overline{XCY}$ Delay			40	ns	
t_{SCD}	XSTR to $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$ Delay	50		200	ns	Adjustable by External R/C
t_{XSW}	XSTR Pulse Width	30			ns	
t_{XCD}	$\overline{XCP}$ to $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$ Delay			50	ns	
t_{XCW}	$\overline{XCP}$ Pulse Width	35			ns	
t_{CCD}	$\overline{XCP}$ to $\overline{XCY}$ Delay	50		200	ns	Adjustable by External R/C
t_{CMD}	$\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$ to $\overline{MRDC}$, $\overline{MWTC}$, $\overline{IORC}$, $\overline{IOWC}$			35	ns	
t_{CRD}	$\overline{MRDR}$, $\overline{MWTR}$, $\overline{IORR}$, $\overline{IOWR}$ to RDD Delay			25	ns	
t_{RW}	$\overline{RSTB}$ Min. Neg. Pulse Width	30			ns	
t_{CPD}	$\overline{BCLK}$ to $\overline{BPRO}$ Delay			40	ns	
t_{XRD}	$\overline{XCP}$ to $\overline{RDD}$ Delay	25		70	ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 ' AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 1 5V FOR BOTH A LOGIC 1 AND 0 '

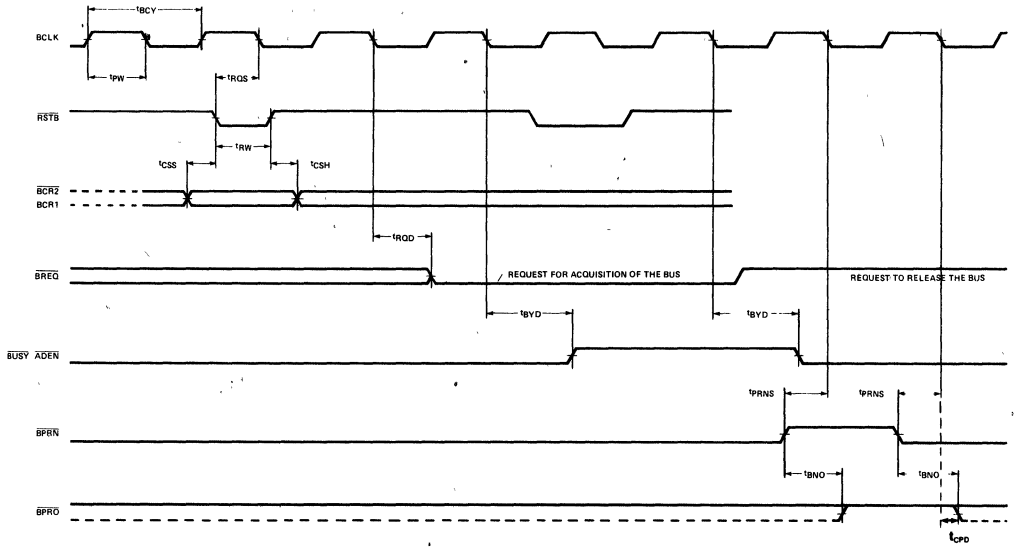
A.C. TESTING LOAD CIRCUIT



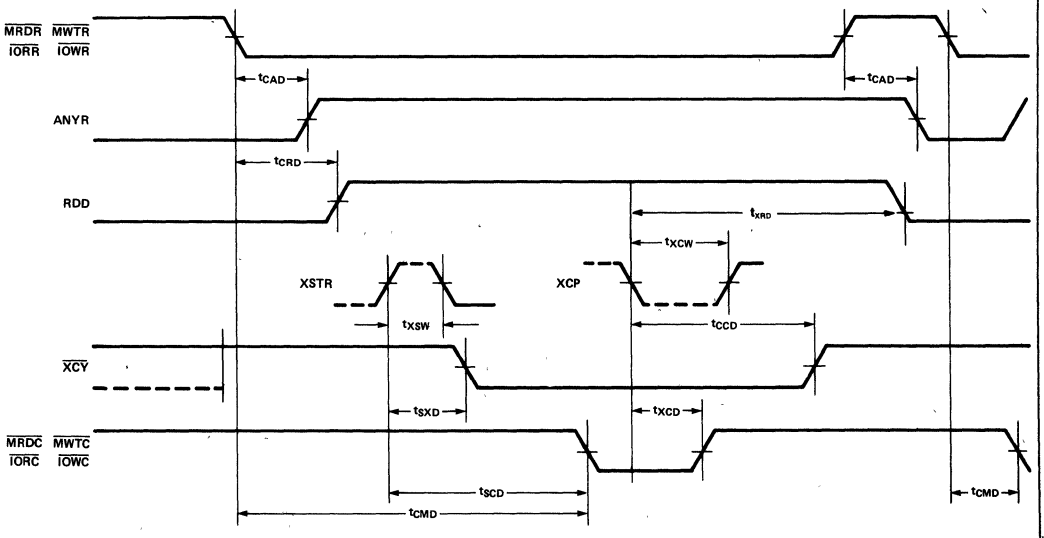
V_L = 2.4V
R_L = 750Ω
C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

SYNCHRONOUS BUS TIMING (System Bus Previously Not In Use)

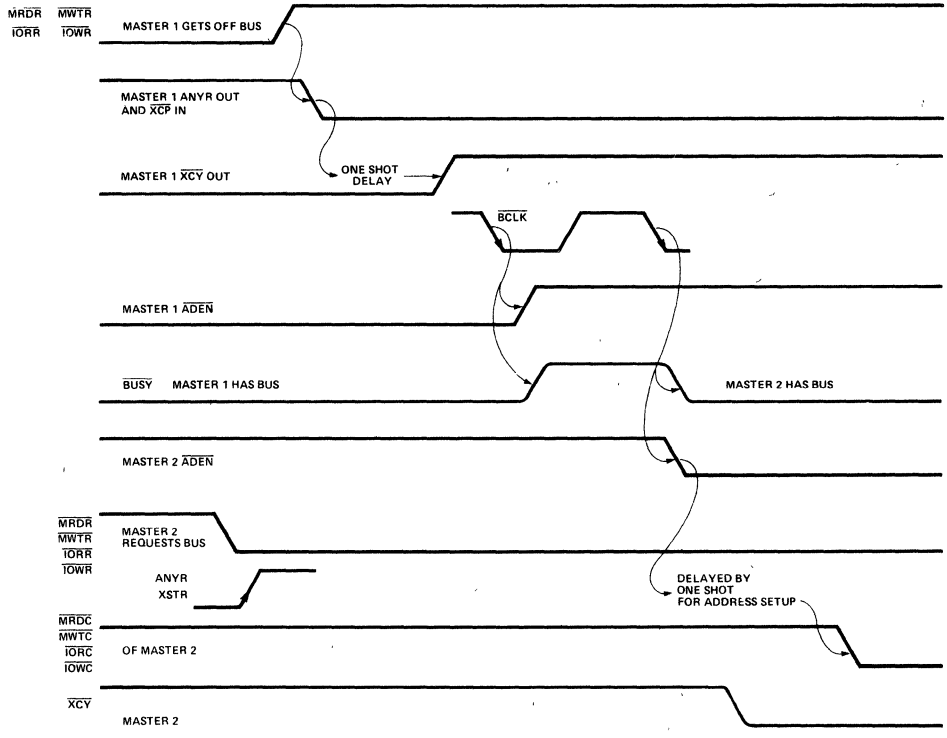


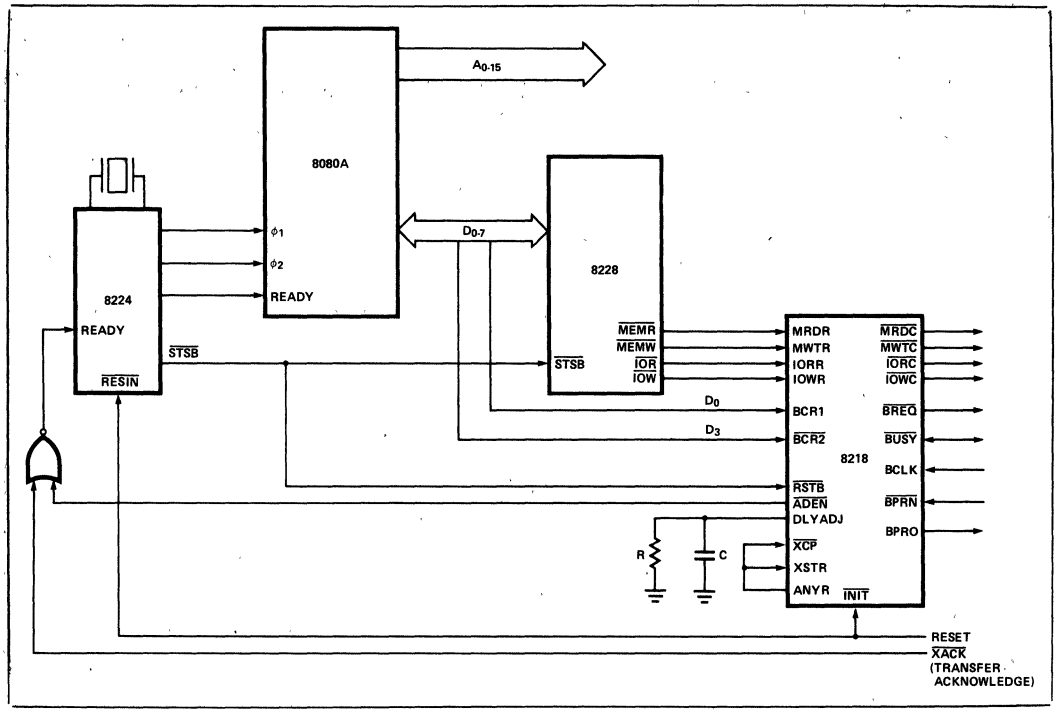
CONTROL CYCLE (System Bus Previously Not In Use)



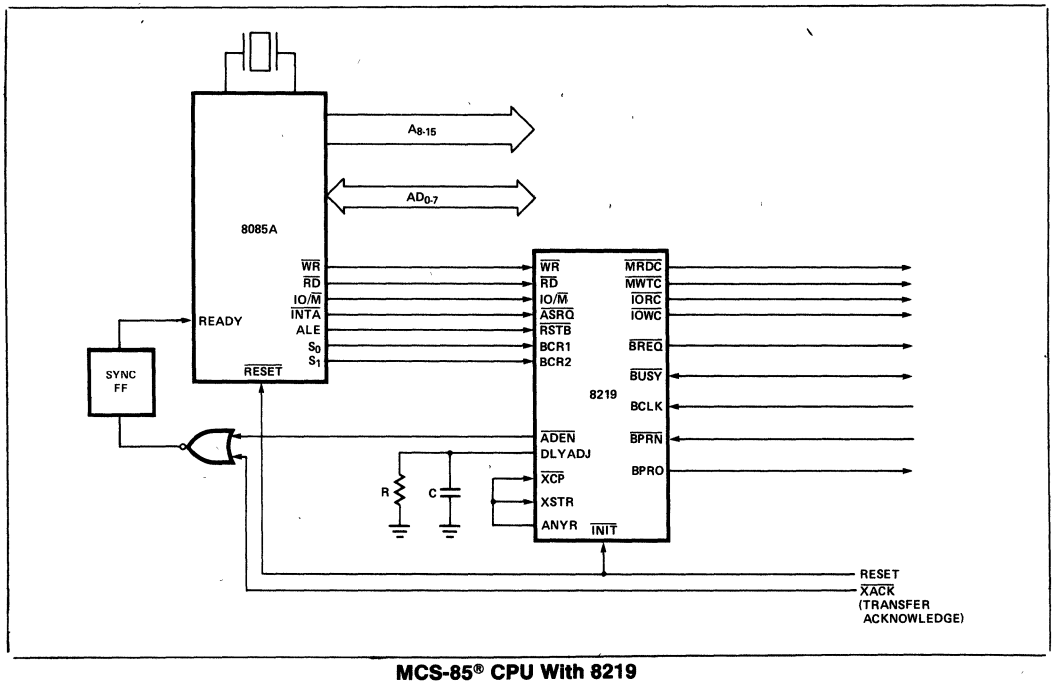
WAVEFORMS (Continued)

BUS CONTROL EXCHANGE (Master No. 1 Leaving Bus And Master No. 2 Getting On Bus)

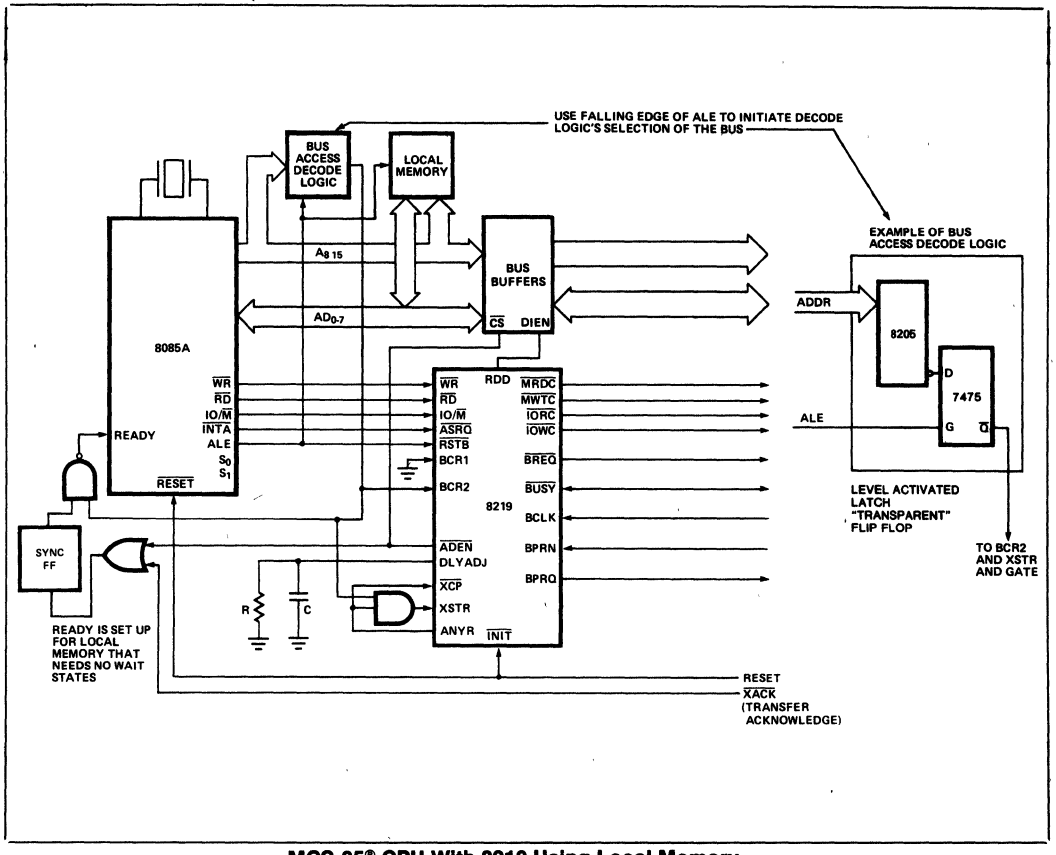




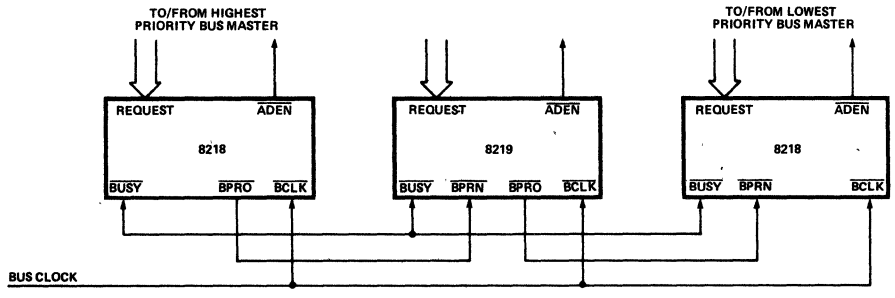
MCS-80® CPU With 8218



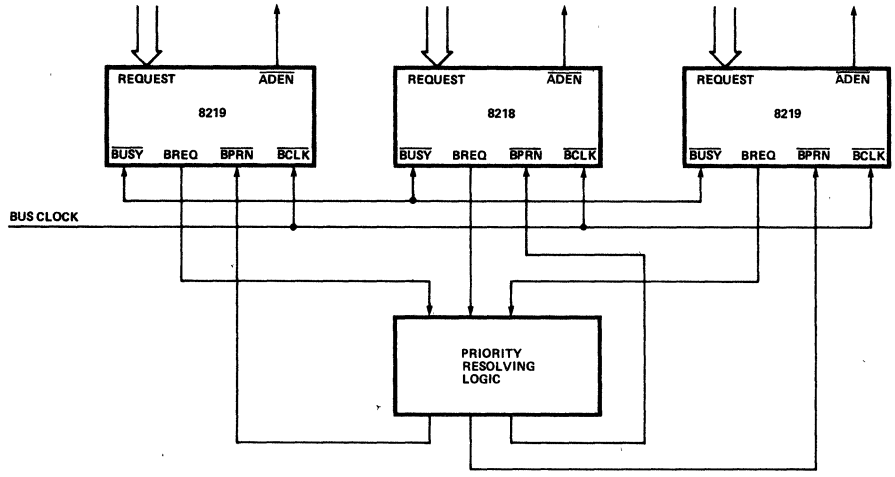
MCS-85® CPU With 8219



MCS-85® CPU With 8219 Using Local Memory



"DAISY CHAIN" CONFIGURATION



PARALLEL REQUEST CONFIGURATION

Two Methods of Connecting Multiple 8218/8219's To Resolve Bus Contention Among Multiple Masters

8224
CLOCK GENERATOR AND DRIVER FOR 8080A CPU

- **Single Chip Clock Generator/Driver for 8080A CPU**
- **Power-Up Reset for CPU**
- **Ready Synchronizing Flip-Flop**
- **Advanced Status Strobe**
- **Oscillator Output for External System Timing**
- **Crystal Controlled for Stable System Operation**
- **Reduces System Package Count**
- **Available in EXPRESS**
  - Standard Temperature Range

The Intel® 8224 is a single chip clock generator/driver for the 8080A CPU. It is controlled by a crystal, selected by the designer to meet a variety of system speed requirements.

Also included are circuits to provide power-up reset, advance status strobe, and synchronization of ready.

The 8224 provides the designer with a significant reduction of packages used to generate clocks and timing for 8080A.

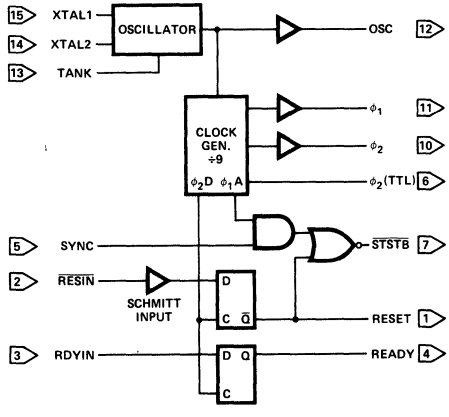


Figure 1. Block Diagram

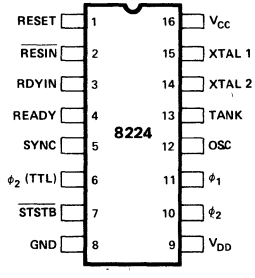


Figure 2. Pin Configuration

RESIN	RESET INPUT
RESET	RESET OUTPUT
RDYIN	READY INPUT
READY	READY OUTPUT
SYNC	SYNC INPUT
STSTB	STATUS STB (ACTIVE LOW)
ϕ_1	8080 CLOCKS
ϕ_2	8080 CLOCKS

XTAL 1	CONNECTIONS FOR CRYSTAL
XTAL 2	CONNECTIONS FOR CRYSTAL
TANK	USED WITH OVERTONE XTAL
OSC	OSCILLATOR OUTPUT
ϕ_2 (TTL)	ϕ_2 CLK (TTL LEVEL)
V_{CC}	+5V
V_{DD}	+12V
GND	0V

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to 150°C
Supply Voltage, V_{CC} . . . . . −0.5V to +7V
Supply Voltage, V_{DD} . . . . . −0.5V to +13.5V
Input Voltage . . . . . −1.5V to +7V
Output Current 100mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5.0V ±5%, V_{DD} = +12V ±5%)

Symbol	Parameter	Limits Min.	Limits Typ.	Limits Max.	Units	Test Conditions
I_F	Input Current Loading			-.25	mA	V_F = .45V
I_R	Input Leakage Current			10	μA	V_R = 5.25V
V_C	Input Forward Clamp Voltage			1.0	V	I_C = -5mA
V_{IL}	Input "Low" Voltage			.8	V	V_{CC} = 5.0V
V_{IH}	Input "High" Voltage	2.6 2.0			V	Reset Input All Other Inputs
V_{IH}-V_{IL}	RESIN Input Hysteresis	.25			V	V_{CC} = 5.0V
V_{OL}	Output "Low" Voltage			.45 .45	V V	(ϕ_1,ϕ_2), Ready, Reset, $\overline{STSTB}$ I_{OL} =2.5mA All Other Outputs I_{OL} = 15mA
V_{OH}	Output "High" Voltage ϕ_1 , ϕ_2 READY, RESET All Other Outputs	 9.4 3.6 2.4			 V V V	 I_{OH} = -100μA I_{OH} = -100μA I_{OH} = -1mA
I_{SC}[1]	Output Short Circuit Current (All Low Voltage Outputs Only)	-10		-60	mA	V_O = 0V V_{CC} = 5.0V
I_{CC}	Power Supply Current			115	mA	
I_{DD}	Power Supply Current			12	mA	

Note: 1. Caution, ϕ_1 and ϕ_2 output drivers do not have short circuit protection

Crystal Requirements

Tolerance: 0.005% at 0°C-70°C
Resonance: Series (Fundamental)*
Load Capacitance: 20-35 pF
Equivalent Resistance: 75-20 ohms
Power Dissipation (Min): 4 mW

*With tank circuit use 3rd overtone mode.

A.C. CHARACTERISTICS (V_{CC} = +5.0V ±5%, V_{DD} = +12.0V ±5%, T_A = 0°C to 70°C)

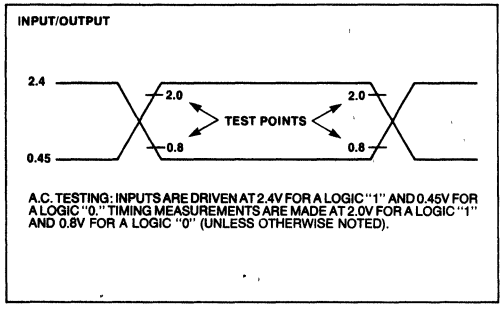
Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	ϕ_1 Pulse Width	$\frac{2tcy}{9}$ - 20ns			ns	C_L = 20pF to 50pF
$t_{\phi 2}$	ϕ_2 Pulse Width	$\frac{5tcy}{9}$ - 35ns				
t_{D1}	ϕ_1 to ϕ_2 Delay	0				
t_{D2}	ϕ_2 to ϕ_1 Delay	$\frac{2tcy}{9}$ - 14ns				
t_{D3}	ϕ_1 to ϕ_2 Delay	$\frac{2tcy}{9}$		$\frac{2tcy}{9}$ + 20ns		
t_R	ϕ_1 and ϕ_2 Rise Time			20		
t_F	ϕ_1 and ϕ_2 Fall Time			20		
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	ϕ_2TTL,CL=30 R_1=300Ω R_2=600Ω
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	$\frac{6tcy}{9}$ - 30ns		$\frac{6tcy}{9}$		
t_{PW}	$\overline{STSTB}$ Pulse Width	$\frac{tcy}{9}$ - 15ns				$\overline{STSTB}$, CL=15pF R_1 = 2K R_2 = 4K
t_{DRS}	RDYIN Setup Time to Status Strobe	50ns - $\frac{4tcy}{9}$				
t_{DRH}	RDYIN Hold Time After $\overline{STSTB}$	$\frac{4tcy}{9}$				
t_{DR}	RDYIN or RESIN to ϕ_2 Delay	$\frac{4tcy}{9}$ - 25ns				Ready & Reset CL=10pF R_1=2K R_2=4K
t_{CLK}	CLK Period		$\frac{tcy}{9}$			
f_{max}	Maximum Oscillating Frequency			27	MHz	
C_{in}	Input Capacitance			8	pF	V_{CC}=+5.0V V_{DD}=+12V V_{BIAS}=2.5V f=1MHz

A.C. CHARACTERISTICS (Continued)

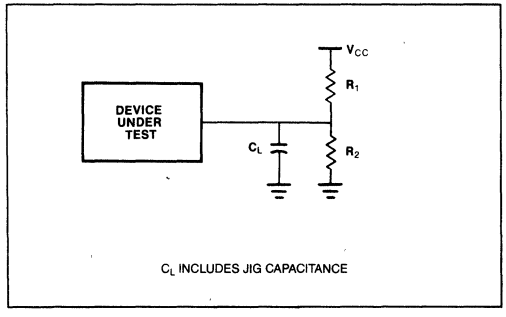
(For t_{CY} = 488.28 ns) (T_A = 0°C to 70°C, V_{DD} = +5V ±5%, V_{DD} = +12V ±5%)

Symbol	Parameter	Limits Min.	Typ.	Max.	Units	Test Conditions
$t_{\phi 1}$	ϕ_1 Pulse Width	89			ns	t_{CY}=488.28ns
$t_{\phi 2}$	ϕ_2 Pulse Width	236			ns	
t_{D1}	Delay ϕ_1 to ϕ_2	0			ns	
t_{D2}	Delay ϕ_2 to ϕ_1	95			ns	ϕ_1 & ϕ_2 Loaded to C_L = 20 to 50pF
t_{D3}	Delay ϕ_1 to ϕ_2 Leading Edges	109		129	ns	
t_r	Output Rise Time			20	ns	
t_f	Output Fall Time			20	ns	
t_{DSS}	ϕ_2 to $\overline{STSTB}$ Delay	296		326	ns	
$t_{D\phi 2}$	ϕ_2 to ϕ_2 (TTL) Delay	-5		+15	ns	
t_{PW}	Status Strobe Pulse Width	40			ns	Ready & Reset Loaded to 2mA/10pF
t_{DRS}	RDYIN Setup Time to $\overline{STSTB}$	-167			ns	
t_{DRH}	RDYIN Hold Time after $\overline{STSTB}$	217			ns	All measurements referenced to 1.5V unless specified otherwise.
t_{DR}	READY or RESET to ϕ_2 Delay	192			ns	
f_{MAX}	Oscillator Frequency			18.432	MHz	

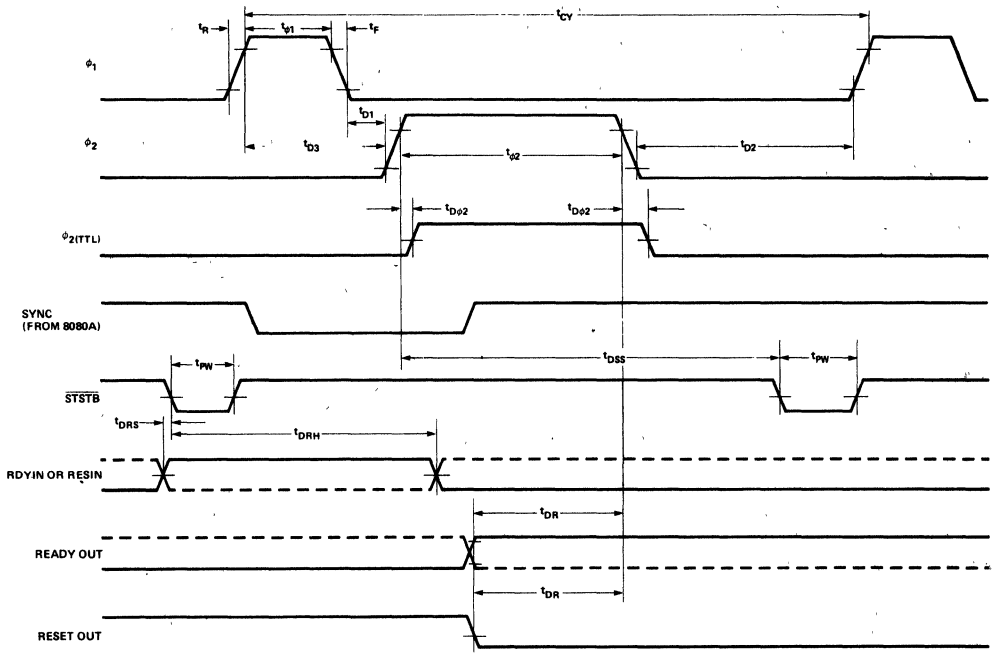
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



WAVEFORMS



VOLTAGE MEASUREMENT POINTS: ϕ_1, ϕ_2 Logic "0" = 1.0V, Logic "1" = 8.0V. All other signals measured at 1.5V.

8228/8238
SYSTEM CONTROLLER AND BUS DRIVER FOR 8080A CPU

- **Single Chip System Control for MCS-80® Systems**
- **Built-In Bidirectional Bus Driver for Data Bus Isolation**
- **Allows the Use of Multiple Byte Instructions (e.g. CALL) for Interrupt Acknowledge**
- **User Selected Single Level Interrupt Vector (RST 7)**
- **28-Pin Dual In-Line Package**
- **Reduces System Package Count**
- **8238 Had Advanced IOW/MEMW for Large System Timing Control**
- **Available in EXPRESS - Standard Temperature Range**

The Intel® 8228 is a single chip system controller and bus driver for MCS-80. It generates all signals required to directly interface MCS-80 family RAM, ROM, and I/O components.

A bidirectional bus driver is included to provide high system TTL fan-out. It also provides isolation of the 8080 data bus from memory and I/O. This allows for the optimization of control signals, enabling the systems designer to use slower memory and I/O. The isolation of the bus driver also provides for enhanced system noise immunity.

A user selected single level interrupt vector (RST 7) is provided to simplify real time, interrupt driven, small system requirements. The 8228 also generates the correct control signals to allow the use of multiple byte instructions (e.g., CALL) in response to an interrupt acknowledge by the 8080A. This feature permits large, interrupt driven systems to have an unlimited number of interrupt levels.

The 8228 is designed to support a wide variety of system bus structures and also reduce system package count for cost effective, reliable design of the MCS-80 systems.

Note: The specifications for the 3228/3238 are identical with those for the 8228/8238

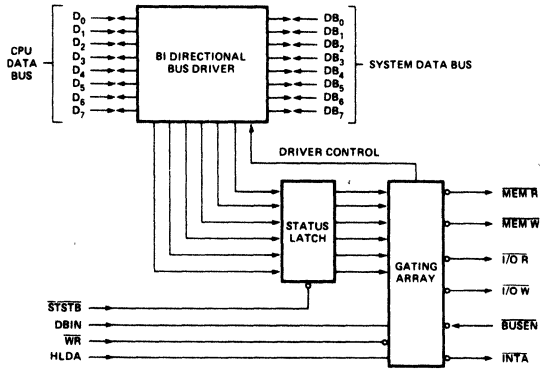


Figure 1. Block Diagram

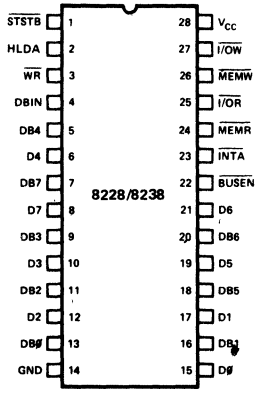


Figure 2. Pin Configuration

D7 D0	DATA BUS (8080 SIDE)	INTA	INTERRUPT ACKNOWLEDGE
DB7 DB0	DATA BUS (SYSTEM SIDE)	HLDA	HLDA (FROM 8080)
I/OR	I/O READ	WR	WR (FROM 8080)
I/OW	I/O WRITE	BUSEN	BUS ENABLE INPUT
MEMR	MEMORY READ	STSTB	STATUS STROBE (FROM 8224)
MEMW	MEMORY WRITE	VCC	+5V
DBIN	DBIN (FROM 8080)	GND	0 VOLTS

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... −0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Supply Voltage, V_{CC} ..... −0.5V to +7V
Input Voltage ..... −1.5V to +7V
Output Current ..... 100 mA

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not limited. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Limits Min.	Typ.[1]	Max.	Unit	Test Conditions
V_C	Input Clamp Voltage, All Inputs		.75	-1.0	V	V_{CC}=4.75V; I_C=-5mA
I_F	Input Load Current, $\overline{STSTB}$			500	μA	V_{CC} = 5.25V
	D_2 & D_6			750	μA	V_F = 0.45V
	D_0, D_1, D_4, D_5, & D_7			250	μA	
	All Other Inputs			250	μA	
I_R	Input Leakage Current $\overline{STSTB}$			100	μA	V_{CC} = 5.25V
	DB_0-DB_7			20	μA	V_R = 5.25V
	All Other Inputs			100	μA	
V_{TH}	Input Threshold Voltage, All Inputs	0.8		2.0	V	V_{CC} = 5V
I_{CC}	Power Supply Current		140	190	mA	V_{CC}=5.25V
V_{OL}	Output Low Voltage, D_0-D_7			.45	V	V_{CC}=4.75V; I_{OL}=2mA
	All Other Outputs			.45	V	I_{OL} = 10mA
V_{OH}	Output High Voltage, D_0-D_7	3.6	3.8		V	V_{CC}=4.75V; I_{OH}=-10μA
	All Other Outputs	2.4			V	I_{OH} = -1mA
I_{OS}	Short Circuit Current, All Outputs	15		90	mA	V_{CC}=5V
$I_{O(off)}$	Off State Output Current, All Control Outputs			100	μA	V_{CC}=5.25V; V_O=5.25
				-100	μA	V_O=.45V
I_{INT}	INTA Current			5	mA	(See INTA Test Circuit)

Note 1. Typical values are for T_A = 25°C and nominal supply voltages.

CAPACITANCE (V_{BIAS} = 2.5V, V_{CC} = 5.0V, T_A = 25°C, f = 1 MHz)

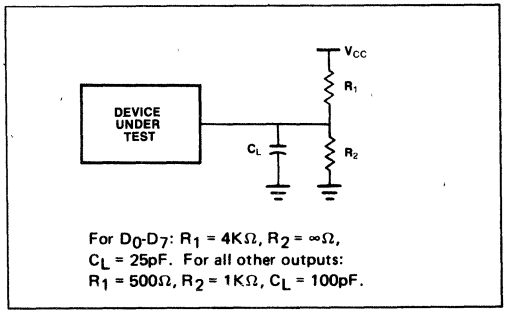
This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max.	
C_{IN}	Input Capacitance		8	12	pF
C_{OUT}	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

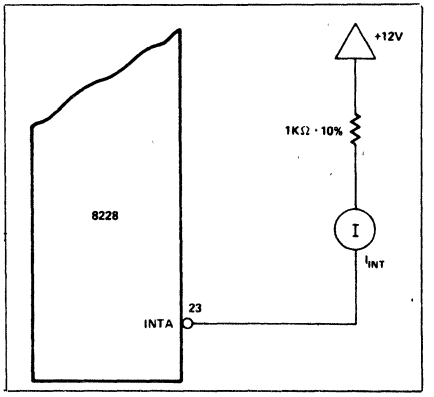
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Limits		Units	Condition
		Min.	Max.		
t_{PW}	Width of Status Strobe	22		ns	
t_{SS}	Setup Time, Status Inputs D_0-D_7	8		ns	
t_{SH}	Hold Time, Status Inputs D_0-D_7	5		ns	
t_{DC}	Delay from $\overline{STSTB}$ to any Control Signal	20	60	ns	C_L = 100pF
t_{RR}	Delay from DBIN to Control Outputs		30	ns	C_L = 100pF
t_{RE}	Delay from DBIN to Enable/Disable 8080 Bus		45	ns	C_L = 25pF
t_{RD}	Delay from System Bus to 8080 Bus during Read		30	ns	C_L = 25pF
t_{WR}	Delay from $\overline{WR}$ to Control Outputs	5	45	ns	C_L = 100pF
t_{WE}	Delay to Enable System Bus DB_0-DB_7 after $\overline{STSTB}$		30	ns	C_L = 100pF
t_{WD}	Delay from 8080 Bus D_0-D_7 to System Bus DB_0-DB_7 during Write	5	40	ns	C_L = 100pF
t_E	Delay from $\overline{System\ Bus\ Enable}$ to System Bus DB_0-DB_7		30	ns	C_L = 100pF
t_{HD}	HLDA to Read Status Outputs		25	ns	
t_{DS}	Setup Time, System Bus Inputs to HLDA	10		ns	
t_{DH}	Hold Time, System Bus Inputs to HLDA	20		ns	C_L = 100pF

A.C. TESTING LOAD CIRCUIT

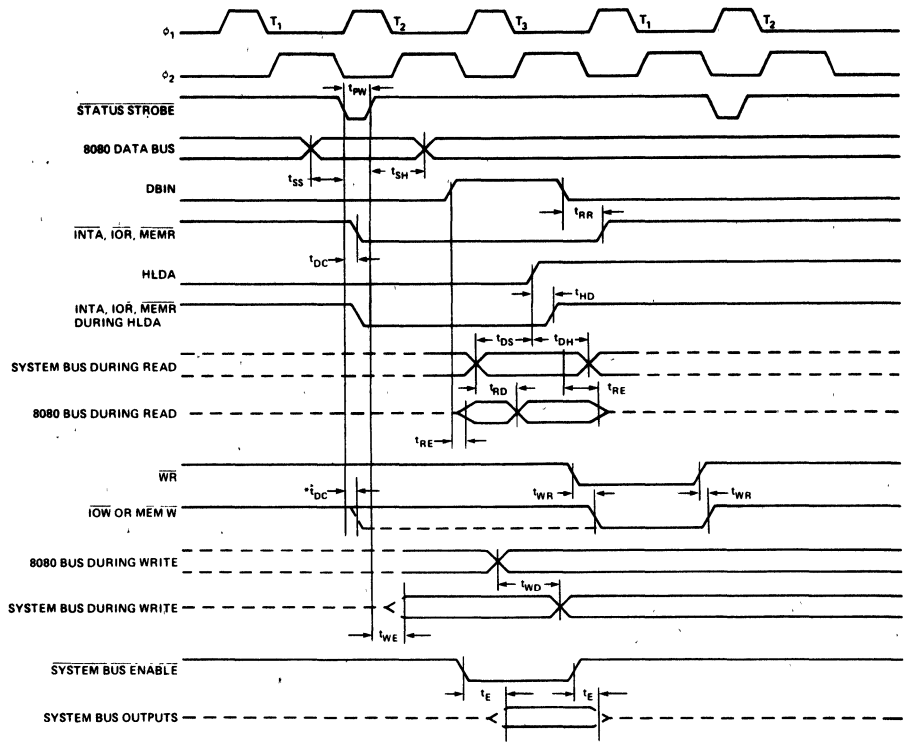


For D_0-D_7: R_1 = 4KΩ, R_2 = ∞Ω, C_L = 25pF. For all other outputs: R_1 = 500Ω, R_2 = 1KΩ, C_L = 100pF.



INTA Test Circuit (for RST 7)

WAVEFORM



VOLTAGE MEASUREMENT POINTS: D_0-D_7 (when outputs) Logic "0" = 0.8V, Logic "1" = 3.0V. All other signals measured at 1.5V.

*ADVANCED $\overline{IOW}$/$\overline{MEMW}$ FOR 8238 ONLY.

8237A/8237A-4/8237A-5
HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER

- **Enable/Disable Control of Individual DMA Requests**
- **Four Independent DMA Channels**
- **Independent Autoinitialization of all Channels**
- **Memory-to-Memory Transfers**
- **Memory Block Initialization**
- **Address Increment or Decrement**
- **High performance: Transfers up to 1.6M Bytes/Second with 5 MHz 8237A-5**
- **Directly Expandable to any Number of Channels**
- **End of Process Input for Terminating Transfers**
- **Software DMA Requests**
- **Independent Polarity Control for DREQ and DACK Signals**
- **Available in EXPRESS**
  - **Standard Temperature Range**

The 8237A Multimode Direct Memory Access (DMA) Controller is a peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information from the system memory. Memory-to-memory transfer capability is also provided. The 8237A offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The 8237A is designed to be used in conjunction with an external 8-bit address register such as the 8282. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process (EOP).

Each channel has a full 64K address and word count capability.

The 8237A-4 and 8237A-5 are 4 MHz and 5 MHz selected versions of the standard 3 MHz 8237A respectively.

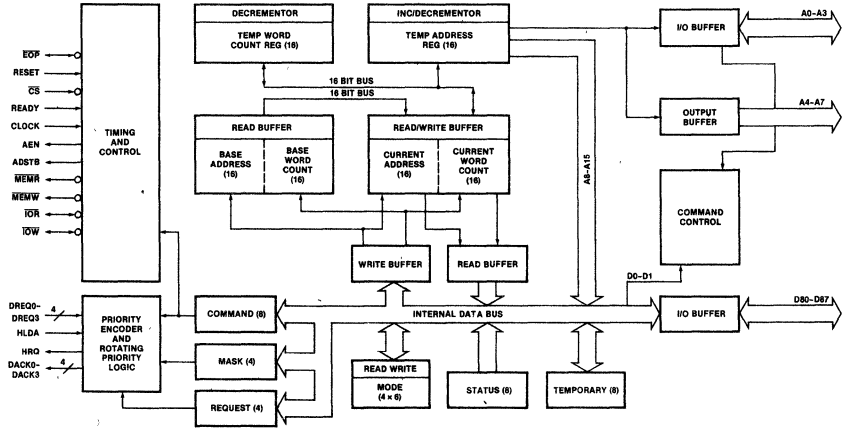


Figure 1. Block Diagram

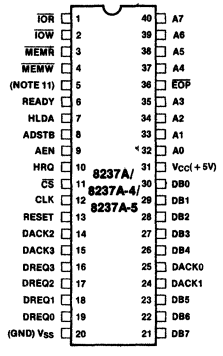


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Ground.
CLK	I	**Clock Input:** Clock input controls the internal operations of the 8237A and its rate of data transfers. The input may be driven at up to 3 MHz for the standard 8237A and up to 5 MHz for the 8237A-5.
CS	I	**Chip Select:** Chip Select is an active low input used to select the 8237A as an I/O device during the Idle cycle. This allows CPU communication on the data bus.
RESET	I	**Reset:** Reset is an active high input which clears the Command, Status, Request and Temporary registers. It also clears the first/last flip/flop and sets the Mask register. Following a Reset the device is in the Idle cycle.
READY	I	**Ready:** Ready is an input used to extend the memory read and write pulses from the 8237A to accommodate slow memories or I/O peripheral devices. Ready must not make transitions during its specified setup/hold time.
HLDA	I	**Hold Acknowledge:** The active high Hold Acknowledge from the CPU indicates that it has relinquished control of the system busses.
DREQ0-DREQ3	I	**DMA Request:** The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service. In fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority. A request is generated by activating the DREQ line of a channel. DACK will acknowledge the recognition of DREQ signal. Polarity of DREQ is programmable. Reset intializes these lines to active high. DREQ must be maintained until the corresponding DACK goes active.
DB0-DB7	I/O	**Data Bus:** The Data Bus lines are bidirectional three-state signals connected to the system data bus. The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU. The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers. During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobed into an external latch by ADSTB. In memory-to-memory operations, data from the memory comes into the 8237A on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.
$\overline{IOR}$	I/O	**I/O Read:** I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.
$\overline{IOW}$	I/O	**I/O Write:** I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A. In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer.
$\overline{EOP}$	I/O	**End of Process:** End of Process is an active low bidirectional signal. Information concerning the completion of DMA services is available at the bidirectional $\overline{EOP}$ pin. The 8237A allows an external signal to terminate an active DMA service. This is accomplished by pulling the $\overline{EOP}$ input low with an external $\overline{EOP}$ signal. The 8237A also generates a pulse when the terminal count (TC) for any channel is reached. This generates an $\overline{EOP}$ signal which is output through the $\overline{EOP}$ Line. The reception of $\overline{EOP}$, either internal or external, will cause the 8237A to terminate the service, reset the request, and, if Autoinitialize is enabled, to write the base registers to the current registers of that channel. The mask bit and TC bit in the status word will be set for the currently active channel by $\overline{EOP}$ unless the channel is programmed for Autoinitialize. In that case, the mask bit remains unchanged. During memory-to-memory transfers, $\overline{EOP}$ will be output when the TC for channel 1 occurs. $\overline{EOP}$ should be tied high with a pull-up resistor if it is not used to prevent erroneous end of process inputs.
A0-A3	I/O	**Address:** The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
A4-A7	O	**Address:** The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.
HRQ	O	**Hold Request:** This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ. After HRQ goes active at least one clock cycle (TCY) must occur before HLDA goes active.
DACK0-DACK3	O	**DMA Acknowledge:** DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.
AEN	O	**Address Enable:** Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.
ADSTB	O	**Address Strobe:** The active high, Address Strobe is used to strobe the upper address byte into an external latch.
$\overline{\text{MEMR}}$	O	**Memory Read:** The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.
$\overline{\text{MEMW}}$	O	**Memory Write:** The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

FUNCTIONAL DESCRIPTION

The 8237A block diagram includes the major logic blocks and all of the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The 8237A contains 344 bits of internal memory in the form of registers. Figure 3 lists these registers by name and shows the size of each. A detailed description of the registers and their functions can be found under Register Description.

Name	Size	Number
Base Address Registers	16 bits	4
Base Word Count Registers	16 bits	4
Current Address Registers	16 bits	4
Current Word Count Registers	16 bits	4
Temporary Address Register	16 bits	1
Temporary Word Count Register	16 bits	1
Status Register	8 bits	1
Command Register	8 bits	1
Temporary Register	8 bits	1
Mode Registers	6 bits	4
Mask Register	4 bits	1
Request Register	4 bits	1

Figure 3. 8237A Internal Registers

The 8237A contains three basic blocks of control logic. The Timing Control block generates internal timing and external control signals for the 8237A. The Program Command Control block decodes the various commands given to the 8237A by the microprocessor prior to servicing a DMA Request. It also decodes the Mode Control word used to select the type of DMA during the servicing. The Priority Encoder block resolves priority contention between DMA channels requesting service simultaneously.

The Timing Control block derives internal timing from the clock input. In 8237A systems this input will usually be the $\phi 2$ TTL clock from an 8224 or CLK from an 8085AH or 8284A. For 8085AH-2 systems above 3.9 MHz, the 8085 CLK(OUT) does not satisfy 8237A-5 clock LOW and HIGH time requirements. In this case, an external clock should be used to drive the 8237A-5.

DMA Operation

The 8237A is designed to operate in two major cycles. These are called Idle and Active cycles. Each device cycle is made up of a number of states. The 8237A can assume seven separate states, each composed of one full clock period. State I (SI) is the inactive state. It is entered when the 8237A has no valid DMA requests pending. While in SI, the DMA controller is inactive but may be in the Program Condition, being programmed by the processor. State S0 (S0) is the first state of a DMA service. The 8237A has requested a hold but the processor has not yet returned an acknowledge. The 8237A may still be programmed until it receives HLDA from the CPU. An acknowledge from the CPU will signal that DMA transfers may begin. S1, S2, S3 and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing, wait states (SW) can be inserted between S2 or S3 and S4 by the use of the Ready line on the 8237A. Note that the data is transferred directly from the I/O device to memory (or vice versa) with $\overline{\text{IOR}}$ and $\overline{\text{MEMW}}$ (or $\overline{\text{MEMR}}$ and $\overline{\text{IOW}}$) being active at the same time. The data is not read into or driven out of the 8237A in I/O-to-memory or memory-to-I/O DMA transfers.

Memory-to-memory transfers require a read-from and a write-to-memory to complete each transfer. The states, which resemble the normal working states, use two digit numbers for identification. Eight states are required for a single transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half

and the last four states (S21, S22, S23, S24) for the write-to-memory half of the transfer.

IDLE CYCLE

When no channel is requesting service, the 8237A will enter the Idle cycle and perform "SI" states. In this cycle the 8237A will sample the DREQ lines every clock cycle to determine if any channel is requesting a DMA service. The device will also sample $\overline{CS}$, looking for an attempt by the microprocessor to write or read the internal registers of the 8237A. When $\overline{CS}$ is low and HLDA is low, the 8237A enters the Program Condition. The CPU can now establish, change or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0-A3 are inputs to the device and select which registers will be read or written. The $\overline{IOR}$ and $\overline{IOW}$ lines are used to select and time reads or writes. Due to the number and size of the internal registers, an internal flip-flop is used to generate an additional bit of address. This bit is used to determine the upper or lower byte of the 16-bit Address and Word Count registers. The flip-flop is reset by Master Clear or Reset. A separate software command can also reset this flip-flop.

Special software commands can be executed by the 8237A in the Program Condition. These commands are decoded as sets of addresses with the $\overline{CS}$ and $\overline{IOW}$. The commands do not make use of the data bus. Instructions include Clear First/Last Flip-FLop and Master Clear.

ACTIVE CYCLE

When the 8237A is in the Idle cycle and a non-masked channel requests a DMA service, the device will output an HRQ to the microprocessor and enter the Active cycle. It is in this cycle that the DMA service will take place, in one of four modes:

Single Transfer Mode — In Single Transfer mode the device is programmed to make one transfer only. The word count will be decremented and the address decremented or incremented following each transfer. When the word count "rolls over" from zero to FFFFH, a Terminal Count (TC) will cause an Autoinitialize if the channel has been programmed to do so.

DREQ must be held active until DACK becomes active in order to be recognized. If DREQ is held active throughout the single transfer, HRQ will go inactive and release the bus to the system. It will again go active and, upon receipt of a new HLDA, another single transfer will be performed, in 8080A, 8085AH, 8088, or 8086 system this will ensure one full machine cycle execution between DMA transfers. Details of timing between the 8237A and other bus control protocols will depend upon the characteristics of the microprocessor involved.

Block Transfer Mode — In Block Transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external End of Process (EOP) is encountered. DREQ need only be held active until DACK becomes active. Again, an Autoinitialization will occur at the end of the service if the channel has been programmed for it.

Demand Transfer Mode — In Demand Transfer mode the device is programmed to continue making transfers until a TC or external $\overline{EOP}$ is encountered or until DREQ goes inactive. Thus transfers may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ. During the time between services when the microprocessor is allowed to operate, the intermediate values of address and word count are stored in the 8237A Current Address and Current Word Count registers. Only an $\overline{EOP}$ can cause an Autoinitialize at the end of the service. $\overline{EOP}$ is generated either by TC or by an external signal.

Cascade Mode—This mode is used to cascade more than one 8237A together for simple system expansion. The HRQ and HLDA signals from the additional 8237A are connected to the DREQ and DACK signals of a channel of the initial 8237A. This allows the DMA requests of the additional device to propagate through the priority network circuitry of the preceding device. The priority chain is preserved and the new device must wait for its turn to acknowledge requests. Since the cascade channel of the initial 8237A is used only for prioritizing the additional device, it does not output any address or control signals of its own. These could conflict with the outputs of the active channel in the added device. The 8237A will respond to DREQ and DACK but all other outputs except HRQ will be disabled. The ready input is ignored.

Figure 4 shows two additional devices cascaded into an initial device using two of the previous channels. This forms a two level DMA system. More 8237As could be added at the second level by using the remaining channels of the first level. Additional devices can also be added by cascading into the channels of the second level devices, forming a third level.

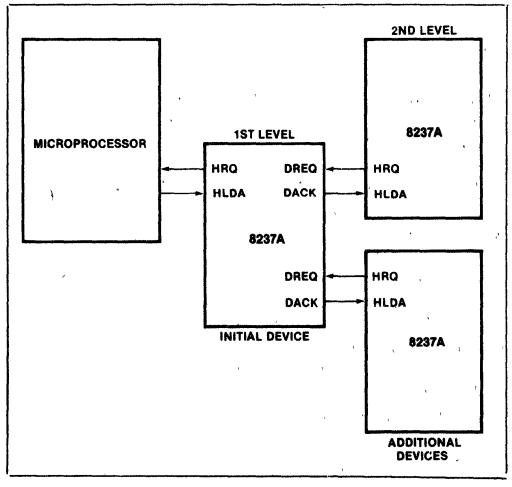


Figure 4. Cascaded 8237As

TRANSFER TYPES

Each of the three active transfer modes can perform three different types of transfers. These are Read, Write and Verify. Write transfers move data from and I/O device to the memory by activating $\overline{MEMW}$ and $\overline{IOR}$. Read transfers move data from memory to an I/O device by activating $\overline{MEMR}$ and $\overline{IOW}$. Verify transfers are pseudo transfers. The 8237A operates as in Read or Write transfers generating addresses, and responding to EOP, etc. However, the memory and I/O control lines all remain inactive. The ready input is ignored in verify mode.

Memory-to-Memory—To perform block moves of data from one memory address space to another with a minimum of program effort and time, the 8237A includes a memory-to-memory transfer feature. Programming a bit in the Command register selects channels 0 to 1 to operate as memory-to-memory transfer channels. The transfer is initiated by setting the software DREQ for channel 0. The 8237A requests a DMA service in the normal manner. After HLDA is true, the device, using four state transfers in Block Transfer mode, reads data from the memory. The channel 0 Current Address register is the source for the address used and is decremented or incremented in the normal manner. The data byte read from the memory is stored in the 8237A internal Temporary register. Channel 1 then performs a four-state transfer of the data from the Temporary register to memory using the address in its Current Address register and incrementing or decrementing it in the normal manner. The channel 1 current Word Count is decremented. When the word count of channel 1 goes to FFFFH, a TC is generated causing an $\overline{EOP}$ output terminating the service.

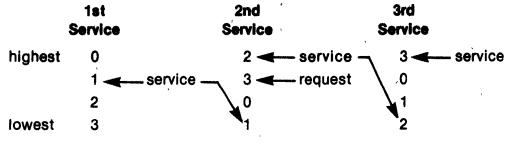
Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory.

The 8237A will respond to external $\overline{EOP}$ signals during memory-to-memory transfers. Data comparators in block search schemes may use this input to terminate the service when a match is found. The timing of memory-to-memory transfers is found in Figure 12. Memory-to-memory operations can be detected as an active AEN with no DACK outputs.

Autoinitialize—By programming a bit in the Mode register, a channel may be set up as an Autoinitialize channel. During Autoinitialize initialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word count registers of that channel following $\overline{EOP}$. The base registers are loaded simultaneously with the current registers by the microprocessor and remain unchanged throughout the DMA service. The mask bit is not altered when the channel is in Autoinitialize. Following Autoinitialize the channel is ready to perform another DMA service, without CPU intervention, as soon as a valid DREQ is detected. In order to Autoninitialize both channels in a memory-to-memory transfer, both word counts should be programmed identically. If interrupted externally, $\overline{EOP}$ pulses should be applied in both bus cycles.

Priority—The 8237A has two types of priority encoding available as software selectable options. The first is Fixed Priority which fixes the channels in priority order based upon the descending value of their number. The channel with the lowest priority is 3 followed by 2, 1 and the highest priority channel, 0. After the recognition of any one channel for service, the other channels are prevented from interferring with that service until it is completed.

The second scheme is Rotating Priority. The last channel to get service becomes the lowest priority channel with the others rotating accordingly.



With Rotating Priority in a single chip DMA system, any device requesting service is guaranteed to be recognized after no more than three higher priority services have occurred. This prevents any one channel from monopolizing the system.

Compressed Timing — In order to achieve even greater throughput where system characteristics permit, the 8237A can compress the transfer time to two clock cycles. From Figure 11 it can be seen that state S3 is used to extend the access time of the read pulse. By removing state S3, the read pulse width is made equal to the write pulse width and a transfer consists only of state S2 to change the address and state S4 to perform the read/write. S1 states will still occur when A8–A15 need updating (see Address Generation). Timing for compressed transfers is found in Figure 14.

Address Generation — In order to reduce pin count, the 8237A multiplexes the eight higher order address bits on the data lines. State S1 is used to output the higher order address bits to an external latch from which they may be placed on the address bus. The falling edge of Address Strobe (ADSTB) is used to load these bits from the data lines to the latch. Address Enable (AEN) is used to enable the bits onto the address bus through a three-state enable. The lower order address bits are output by the 8237A directly. Lines A0–A7 should be connected to the address bus. Figure 11 shows the time relationships between CLK, AEN, ADSTB, DB0–DB7 and A0–A7.

During Block and Demand Transfer mode services, which include multiple transfers, the addresses generated will be sequential. For many transfers the data held in the external address latch will remain the same. This data need only change when a carry or borrow from A7 to A8 takes place in the normal sequence of addresses. To save time and speed transfers, the 8237A executes S1 states only when updating of A8–A15 in the latch is necessary. This means for long services, S1 states and Address Strobes may occur only once every 256 transfers, a savings of 255 clock cycles for each 256 transfers.

REGISTER DESCRIPTION

Current Address Register — Each channel has a 16-bit Current Address register. This register holds the value of the address used during DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the address are stored in the Current Address register during the transfer. This register is written or read by the microprocessor in successive 8-bit bytes. It may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize takes place only after an $\overline{EOP}$.

Current Word Register — Each channel has a 16-bit Current Word Count register. This register determines the number of transfers to be performed. The actual number of transfers will be one more than the number programmed in the Current Word Count register (i.e., programming a count of 100 will result in 101 transfers). The word count is decremented after each transfer. The intermediate value of the word count is stored in the register during the transfer. When the value in the register goes from zero to FFFFH, a TC will be generated. This register is loaded or read in successive 8-bit bytes by the microprocessor in the Program Condition. Following the end of a DMA service it may also be reinitialized by an Autoinitialization back to its original value. Autoinitialize can occur only when an $\overline{EOP}$ occurs. If it is not Autoinitialized, this register will have a count of FFFFH after TC.

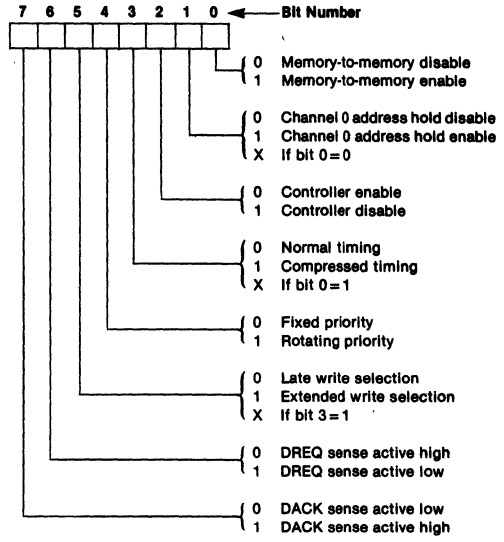
Base Address and Base Word Count Registers — Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original value of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. The base registers are written simultaneously with their corresponding current register in 8-bit bytes in the Program Condition by the microprocessor. These registers cannot be read by the microprocessor.

Command Register — This 8-bit register controls the operation of the 8237A. It is programmed by the microprocessor in the Program Condition and is cleared by Reset or a Master Clear instruction. The following table lists the function of the command bits. See Figure 6 for address coding.

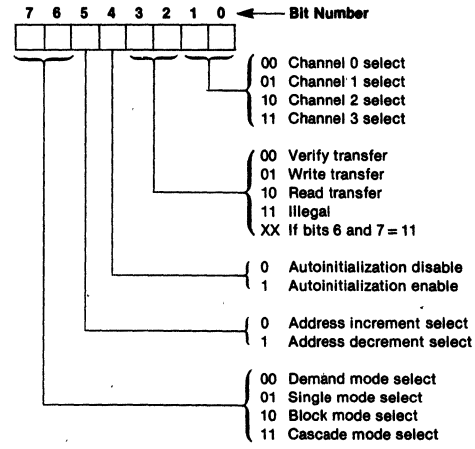
Mode Register — Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register is to be written.

Request Register — The 8237A can respond to requests for DMA service which are initiated by software as well as by a DREQ. Each channel has a request bit associated with it in the 4-bit Request register. These are nonmaskable and subject to prioritization by the Priority Encoder network. Each register bit is set or reset separately under software control or is cleared upon generation of a TC or external $\overline{EOP}$. The entire register is cleared by a Reset. To set or reset a bit, the software loads the proper form of the data word. See Figure 5 for register address coding. In order to make a software request, the channel must be in Block Mode.

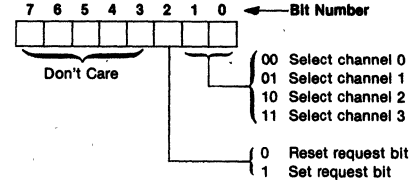
Command Register



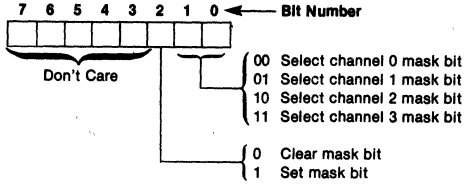
Mode Register



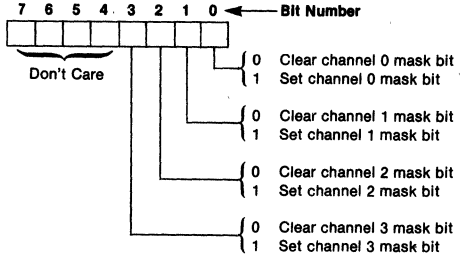
Request Register



Mask Register — Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. Each mask bit is set when its associated channel produces an $\overline{EOP}$ if the channel is not programmed for Autoinitialize. Each bit of the 4-bit Mask register may also be set or cleared separately under software control. The entire register is also set by a Reset. This disables all DMA requests until a clear Mask register instruction allows them to occur. The instruction to separately set or clear the mask bits is similar in form to that used with the Request register. See Figure 5 for instruction addressing.



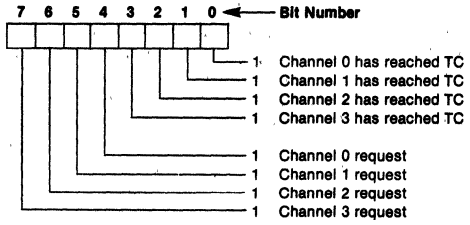
All four bits of the Mask register may also be written with a single command.



Register	Operation	Signals						
		$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0
Command	Write	0	1	0	1	0	0	0
Mode	Write	0	1	0	1	0	1	1
Request	Write	0	1	0	1	0	0	1
Mask	Set/Reset	0	1	0	1	0	1	0
Mask	Write	0	1	0	1	1	1	1
Temporary	Read	0	0	1	1	1	0	1
Status	Read	0	0	1	1	0	0	0

Figure 5. Definition of Register Codes

Status Register — The Status register is available to be read out of the 8237A by the microprocessor. It contains information about the status of the devices at this point. This information includes which channels have reached a terminal count and which channels have pending DMA requests. Bits 0-3 are set every time a TC is reached by that channel or an external $\overline{EOP}$ is applied. These bits are cleared upon Reset and on each Status Read. Bits 4-7 are set whenever their corresponding channel is requesting service.



Temporary Register — The Temporary register is used to hold data during memory-to-memory transfers. Following the completion of the transfers, the last word moved can be read by the microprocessor in the Program Condition. The Temporary register always contains the last byte transferred in the previous memory-to-memory operation, unless cleared by a Reset.

Software Commands—These are additional special software commands which can be executed in the Program Condition. They do not depend on any specific bit pattern on the data bus. The three software commands are:

Clear First/Last Flip-Flop: This command is executed prior to writing or reading new address or word count information to the 8237A. This initializes the flip-flop to a known state so that subsequent accesses to register contents by the microprocessor will address upper and lower bytes in the correct sequence.

Master Clear: This software instruction has the same effect as the hardware Reset. The Command, Status, Request, Temporary, and Internal First/Last Flip-Flop registers are cleared and the Mask register is set. The 8237A will enter the Idle cycle.

Clear Mask Register: This command clears the mask bits of all four channels, enabling them to accept DMA requests.

Figure 6 lists the address codes for the software commands:

Signals						Operation
A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	
1	0	0	0	0	1	Read Status Register
1	0	0	0	1	0	Write Command Register
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Write Request Register
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Write Single Mask Register Bit
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Write Mode Register
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Clear Byte Pointer Flip/Flop
1	1	0	1	0	1	Read Temporary Register
1	1	0	1	1	0	Master Clear
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Clear Mask Register
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Write All Mask Register Bits

Figure 6. Software Command Codes

Channel	Register	Operation	Signals							Internal Flip-Flop	Data Bus DB0–DB7
			$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A3	A2	A1	A0		
0	Base and Current Address	Write	0	1	0	0	0	0	0	0	A0–A7
			0	1	0	0	0	0	0	1	A8–A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0–A7
			0	0	1	0	0	0	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	0	0	1	0	W0–W7
			0	1	0	0	0	0	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W)0–W7
			0	0	1	0	0	0	1	1	W8–W15
1	Base and Current Address	Write	0	1	0	0	0	1	0	0	A0–A7
			0	1	0	0	0	1	0	1	A8–A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0–A7
			0	0	1	0	0	1	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	0	1	1	0	W0–W7
			0	1	0	0	0	1	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0-W7
			0	0	1	0	0	1	1	1	W8–W15
2	Base and Current Address	Write	0	1	0	0	1	0	0	0	A0–A7
			0	1	0	0	1	0	0	1	A8–A15
	Current Address	Read	0	0	1	0	1	0	0	0	A0–A7
			0	0	1	0	1	0	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	1	0	1	0	W0–W7
			0	1	0	0	1	0	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W)0–W7
			0	0	1	0	1	0	1	1	W8–W15
3	Base and Current Address	Write	0	1	0	0	1	1	0	0	A0–A7
			0	1	0	0	1	1	0	1	A8–A15
	Current Address	Read	0	0	1	0	1	1	0	0	A0–A7
			0	0	1	0	1	1	0	1	A8–A15
	Base and Current Word Count	Write	0	1	0	0	1	1	1	0	W0–W7
			0	1	0	0	1	1	1	1	W8–W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W)0–W7
			0	0	1	0	1	1	1	1	W8–W15

Figure 7. Word Count and Address Register Command Codes

PROGRAMMING

The 8237A will accept programming from the host processor any time that HLDA is inactive; this is true even if HRQ is active. The responsibility of the host is to assure that programming and HLDA are mutually exclusive. Note that a problem can occur if a DMA request occurs, on an unmasked channel while the 8237A is being programmed. For instance, the CPU may be starting to reprogram the two byte Address register of channel 1 when channel 1 receives a DMA request. If the 8237A is enabled (bit 2 in the command register is 0) and channel 1 is unmasked, a DMA service will occur after only one byte of the Address register has been reprogrammed. This can be avoided by disabling the controller (setting bit 2 in the command register) or masking the channel before programming any other registers. Once the programming is complete, the controller can be enabled/unmasked.

After power-up it is suggested that all internal locations, especially the Mode registers, be loaded with some valid value. This should be done even if some channels are unused.

APPLICATION INFORMATION

Figure 8 shows a convenient method for configuring a DMA system with the 8237A controller and an 8080A/8085AH microprocessor system. The multimode DMA controller issues a HRQ to the processor whenever there is at least one valid DMA request from a peripheral device. When the processor replies with a HLDA signal, the 8237A takes control of the address bus, the data bus and the control bus. The address for the first transfer operation comes out in two bytes — the least significant 8 bits on the eight address outputs and the most significant 8 bits on the data bus. The contents of the data bus are then latched into the 8282 8-bit latch to complete the full 16 bits of the address bus. The 8282 is a high speed, 8-bit, three-state latch in a 20-pin package. After the initial transfer takes place, the latch is updated only after a carry or borrow is generated in the least significant address byte. Four DMA channels are provided when one 8237A is used.

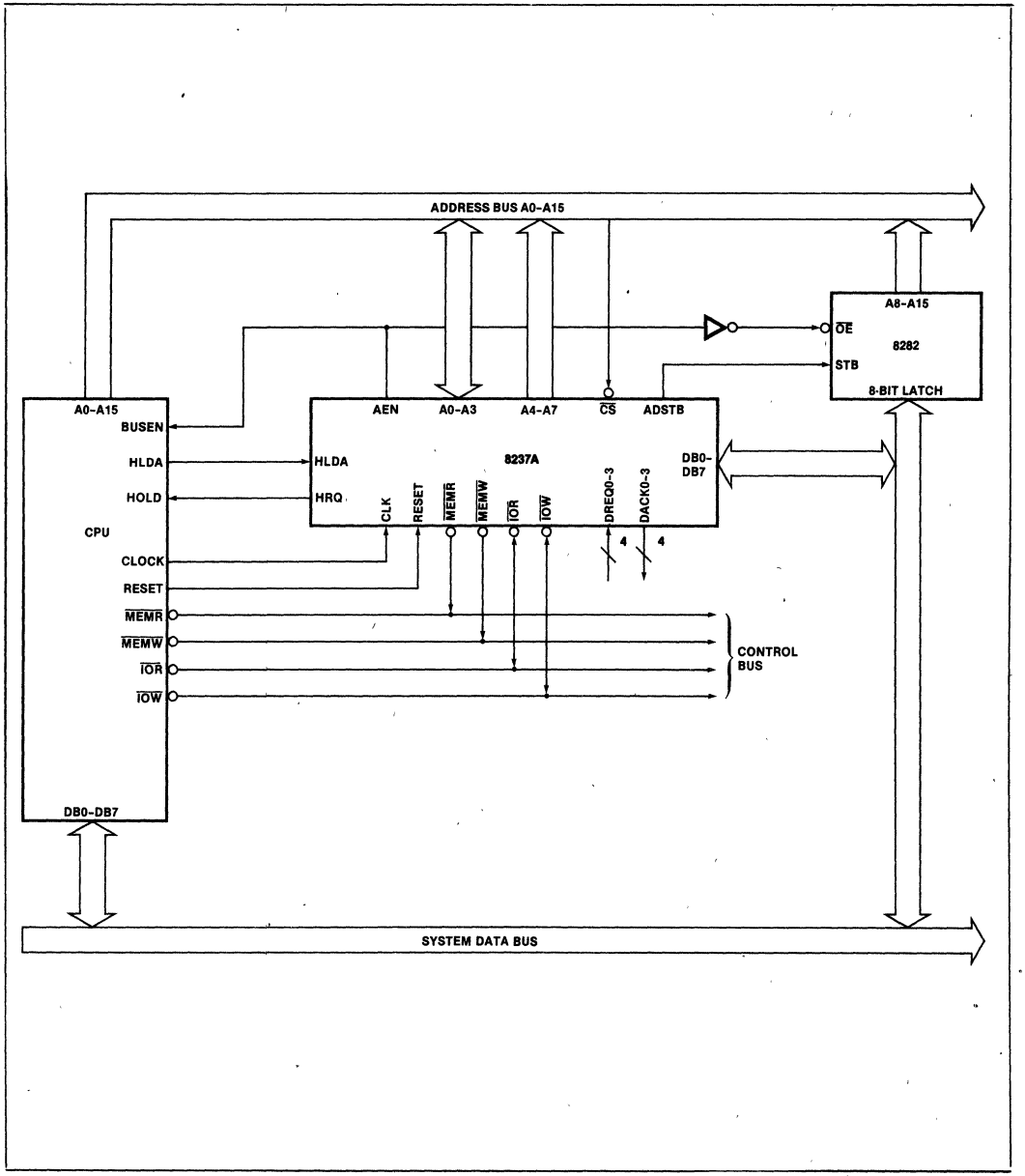


Figure 8. 8237A System Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on any Pin with
Respect to Ground ..... −0.5 to 7V
Power Dissipation ..... 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

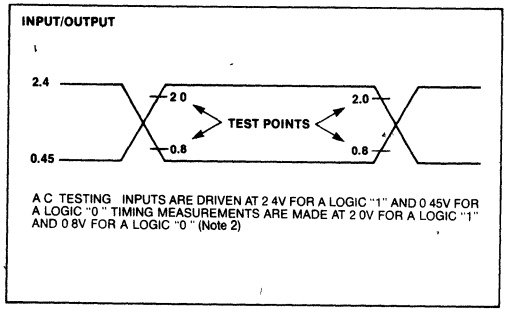
Symbol	Parameter	Min.	Typ.(1)	Max.	Unit	Test Conditions
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -200 μA
		3.3			V	I_{OH} = -100 μA (HRQ Only)
V_{OL}	Output LOW Voltage			45	V	I_{OL} = 2.0mA (data Bus)$\overline{EDP}$ I_{OL} = 3.2mA (other outputs) (Note 8) I_{OL} = 2.5mA (ADSTB) (Note 8)
V_{IH}	Input HIGH Voltage	2.2		V_{CC} + 0.5	V	
V_{IL}	Input LOW Voltage	-0.5		0.8	V	
I_{LI}	Input Load Current			±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current			±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	V_{CC}Supply Current		110	130	mA	T_A = +25°C
			130	150	mA	T_A = 0°C
C_O	Output Capacitance		4	8	pF	fc = 1.0 MHz, Inputs = 0V
C_I	Input Capacitance		8	15	pF	
C_{IO}	I/O Capacitance		10	18	pF	

NOTES:

1. Typical values are for T_A = 25°C, nominal supply voltage and nominal processing parameters
2. Input timing parameters assume transition times of 20 ns or less. Waveform measurement points for both input and output signals are 2 0V for HIGH and 0 8V for LOW, unless otherwise noted.
3. Output loading is 1 TTL gate plus 150pF capacitance, unless otherwise noted.
4. The net $\overline{IOW}$ or $\overline{MEMW}$ Pulse width for normal write will be TCY-100 ns and for extended write will be 2TCY-100 ns. The net $\overline{IOR}$ or $\overline{MEMR}$ pulse width for normal read will be 2TCY-50 ns and for compressed read will be TCY-50 ns.
5. TDQ is specified for two different output HIGH levels TDQ1 is measured at 2.0V. TDQ2 is measured at 3.3V. The value for TDQ2 assumes an external 3.3kΩ pull-up resistor connected form HRQ to V_{CC}.
6. DREQ should be held active until DACK is returned.
7. DREQ and DACK signals may be active high or active low. Timing diagrams assume the active high mode.
8. A revision of the 8237A is planned for shipment in April 1984, which will improve the following characteristics.
   1. V_{IH} from 2.2V to 2.0V
   2. V_{OL} from 0.45V to 0.4V on all outputs. Test condition I_{OL} = 3.2 mA

   Please contact your local sales office at that time for more information.
9. Successive read and/or write operations by the external processor to program or examine the controller must be timed to allow at least 600 ns for the 8237A, at least 500 ns for the 8237A-4 and at least 400 ns for the 8237A-5, as recovery time between active read or write pulses
10. $\overline{EOP}$ is an open collector output. This parameter assumes the presence of a 2.2K pullup to V_{CC}.
11. Pin 5 is an input that should always be at a logic high level An internal pull-up resistor will establish a logic high when the pin is left floating. It is recommended however, that pin 5 be tied to V_{CC}.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0" (Note 2)

A.C. CHARACTERISTICS—DMA (MASTER) MODE ($T_A = 0°C$ to $70°C$, $V_{CC} = +5V \pm 5\%$, GND = 0V)

Symbol	Parameter	8237A		8237A-4		8237A-5		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAEL	AEN HIGH from CLK LOW (S1) Delay Time		300		225		200	ns
TAET	AEN LOW from CLK HIGH (SI) Delay Time		200		150		130	ns
TAFAB	ADR Active to Float Delay from CLK HIGH		150		120		90	ns
TAFC	$\overline{READ}$ or $\overline{WRITE}$ Float from CLK HIGH		150		120		120	ns
TAFDB	DB Active to Float Delay from CLK HIGH		250		190		170	ns
TAHR	ADR from $\overline{READ}$ HIGH Hold Time	TCY–100		TCY–100		TCY–100		ns
TAHS	DB from ADSTB LOW Hold Time	50		40		30		ns
TAHW	ADR from $\overline{WRITE}$ HIGH Hold Time	TCY–50		TCY–50		TCY–50		ns
TAK	DACK Valid from CLK LOW Delay Time (Note 7)		250		220		170	ns
	$\overline{EOP}$ HIGH from CLK HIGH Delay Time (Note 10)		250		190		170	ns
	$\overline{EOP}$ LOW from CLK HIGH Delay Time		250		190		170	ns
TASM	ADR Stable from CLK HIGH		250		190		170	ns
TASS	DB to ADSTB LOW Setup Time	100		100		100		ns
TCH	Clock High Time (Transitions $\leq$ 10 ns)	120		100		80		ns
TCL	Clock LOW Time (Transitions $\leq$ 10 ns)	150		110		68		ns
TCY	CLK Cycle Time	320		250		200		ns
TDCL	CLK HIGH to $\overline{READ}$ or $\overline{WRITE}$ LOW Delay (Note 4)		270		200		190	ns
TDCTR	$\overline{READ}$ HIGH from CLK HIGH (S4) Delay Time (Note 4)		270		210		190	ns
TDCTW	$\overline{WRITE}$ HIGH from CLK HIGH (S4) Delay Time (Note 4)		200		150		130	ns
TDQ1	HRQ Valid from CLK HIGH Delay Time (Note 5)		160		120		120	ns
TDQ2			250		190		120	ns
TEPS	$\overline{EOP}$ LOW from CLK LOW Setup Time	60		45		40		ns
TEPW	$\overline{EOP}$ Pulse Width	300		225		220		ns
TFAAB	ADR Float to Active Delay from CLK HIGH		250		190		170	ns
TFAC	$\overline{READ}$ or $\overline{WRITE}$ Active from CLK HIGH		200		150		150	ns
TFADB	DB Float to Active Delay from CLK HIGH		300		225		200	ns
THS	HLDA Valid to CLK HIGH Setup Time	100		75		75		ns
TIDH	Input Data from $\overline{MEMR}$ HIGH Hold Time	0		0		0		ns
TIDS	Input Data to $\overline{MEMR}$ HIGH Setup Time	250		190		170		ns
TODH	Output Data from $\overline{MEMW}$ HIGH Hold Time	20		20		10		ns
TODV	Output Data Valid to $\overline{MEMW}$ HIGH	200		125		125		ns
TQS	DREQ to CLK LOW (SI, S4) Setup Time (Note 7)	0		0		0		ns
TRH	CLK to READY LOW Hold Time	20		20		20		ns
TRS	READY to CLK LOW Setup Time	100		60		60		ns
TSTL	ADSTB HIGH from CLK HIGH Delay Time		200		150		130	ns
TSTT	ADSTB LOW from CLK HIGH Delay Time		140		110		90	ns

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE (T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)

Symbol	Parameter	8237A		8237A-4		8237A-5		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAR	ADR Valid or $\overline{CS}$ LOW to $\overline{READ}$ LOW	50		50		50		ns
TAW	ADR Valid to $\overline{WRITE}$ HIGH Setup Time	200		150		130		ns
TCW	CS LOW to $\overline{WRITE}$ HIGH Setup Time	200		150		130		ns
TDW	Data Valid to $\overline{WRITE}$ HIGH Setup Time	200		150		130		ns
TRA	ADR or CS Hold from $\overline{READ}$ HIGH	0		0		0		ns
TRDE	Data Access from $\overline{READ}$ LOW (Note 3)		200		200		140	ns
TRDF	DB Float Delay from $\overline{READ}$ HIGH	20	100	20	100	0	70	ns
TRSTD	Power Supply HIGH to RESET LOW Setup Time	500		500		500		ns
TRSTS	RESET to First $\overline{IOWR}$	2TCY		2TCY		2TCY		ns
TRSTW	RESET Pulse Width	300		300		300		ns
TRW	$\overline{READ}$ Width	300		250		200		ns
TWA	ADR from $\overline{WRITE}$ HIGH Hold Time	20		20		20		ns
TWC	CS HIGH from $\overline{WRITE}$ HIGH Hold Time	20		20		20		ns
TWD	Data from $\overline{WRITE}$ HIGH Hold Time	30		30		30		ns
TWWS	Write Width	200		200		160		ns

WAVEFORMS

SLAVE MODE WRITE TIMING

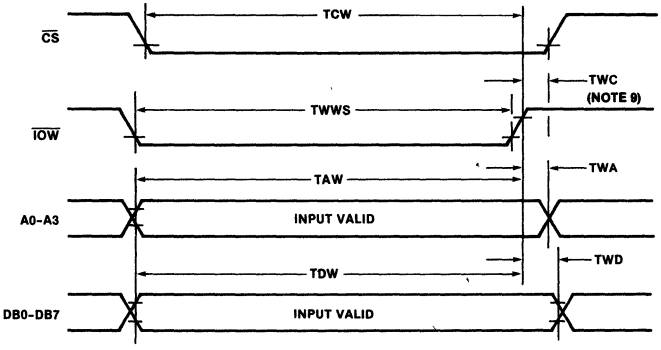


Figure 9. Slave Mode Write

SLAVE MODE READ TIMING

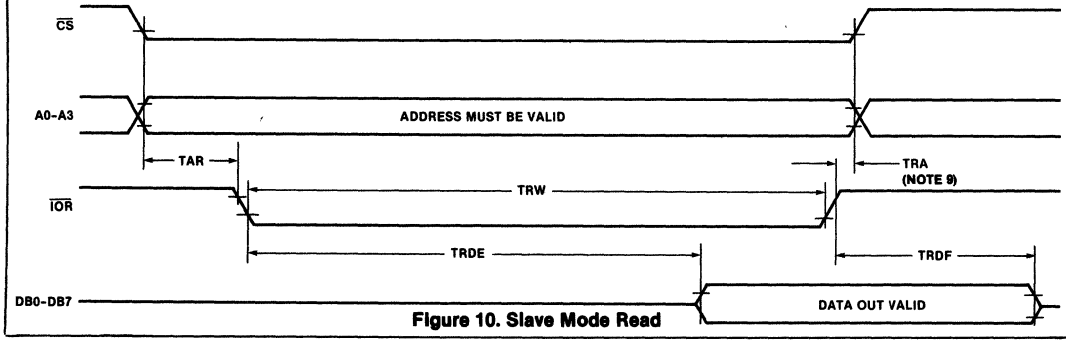


Figure 10. Slave Mode Read

WAVEFORMS (Continued)

DMA TRANSFER TIMING

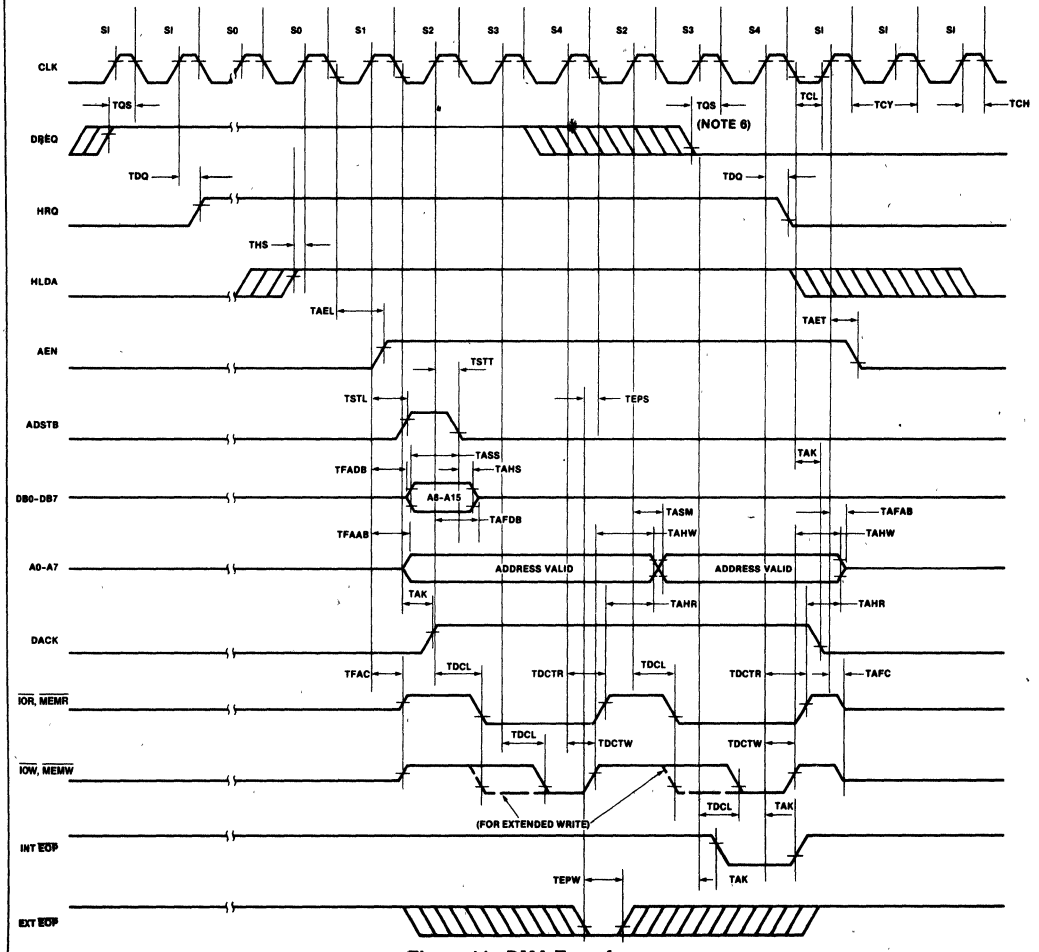


Figure 11. DMA Transfer

WAVEFORMS (Continued)

MEMORY-TO-MEMORY TRANSFER TIMING

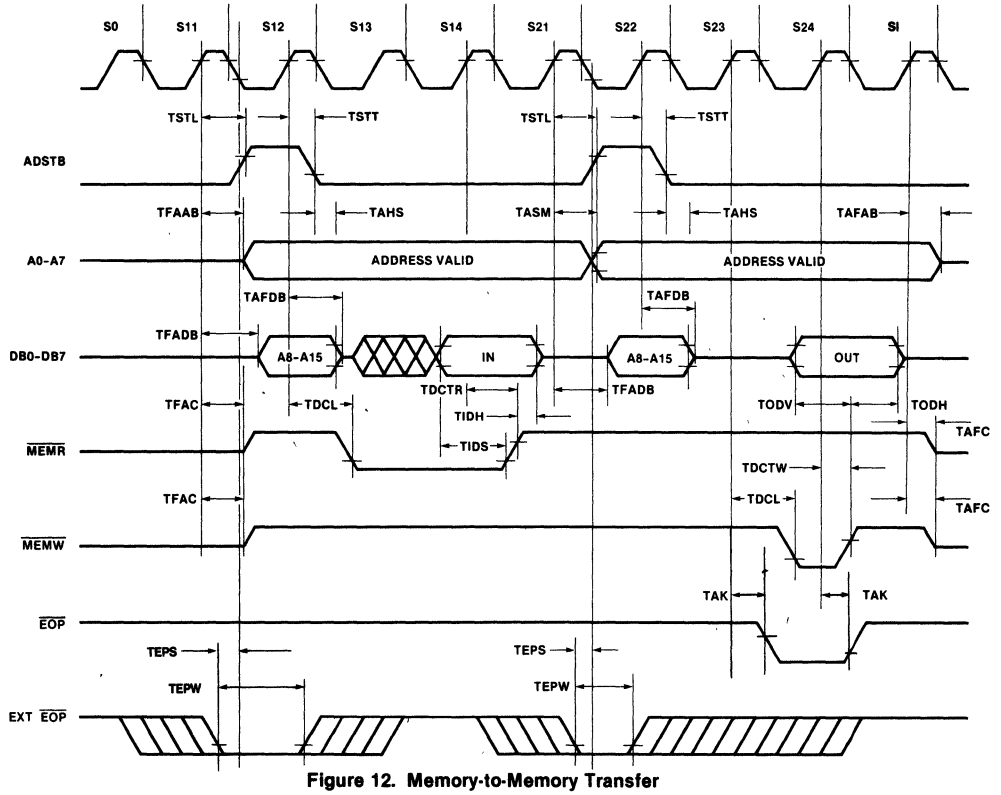


Figure 12. Memory-to-Memory Transfer

READY TIMING

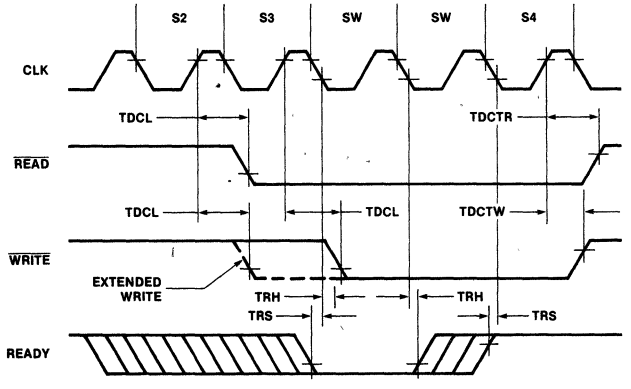


Figure 13. Ready

WAVEFORMS (Continued)

COMPRESSED TRANSFER TIMING

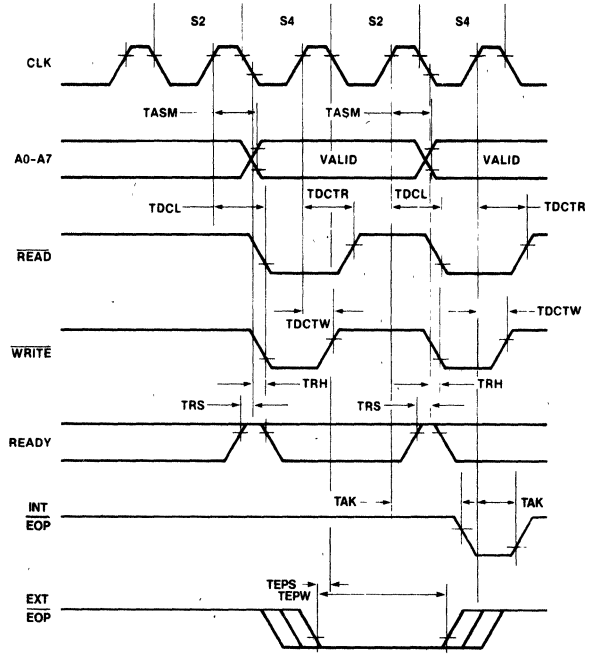


Figure 14. Compressed Transfer

RESET TIMING

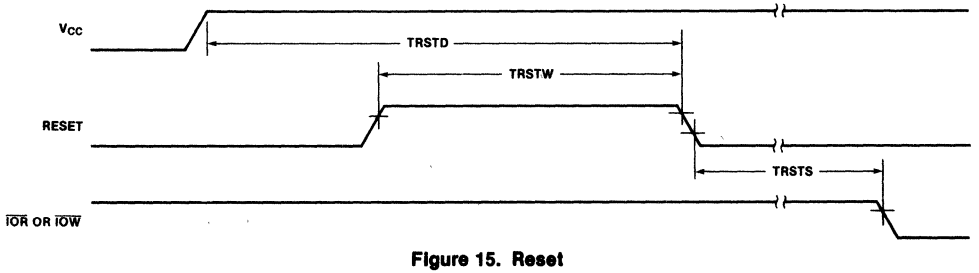


Figure 15. Reset

8257/8257-5
PROGRAMMABLE DMA CONTROLLER

- **MCS-85® Compatible 8257-5**
- **4-Channel DMA Controller**
- **Priority DMA Request Logic**
- **Channel Inhibit Logic**
- **Terminal Count and Modulo 128 Outputs**
- **Single TTL Clock**
- **Single +5V Supply**
- **Auto Load Mode**
- **Available in EXPRESS**
  - Standard Temperature Range

The Intel® 8257 is a 4-channel direct memory access (DMA) controller. It is specifically designed to simplify the transfer of data at high speeds for the Intel® microcomputer systems. Its primary function is to generate, upon a peripheral request, a sequential memory address which will allow the peripheral to read or write data directly to or from memory. Acquisition of the system bus in accomplished via the CPU's hold function. The 8257 has priority logic that resolves the peripherals requests and issues a composite hold request to the CPU. It maintains the DMA cycle count for each channel and outputs a control signal to notify the peripheral that the programmed number of DMA cycles is complete. Other output control signals simplify sectored data transfers. The 8257 represents a significant savings in component count for DMA-based microcomputer systems and greatly simplifies the transfer of data at high speed between peripherals and memories.

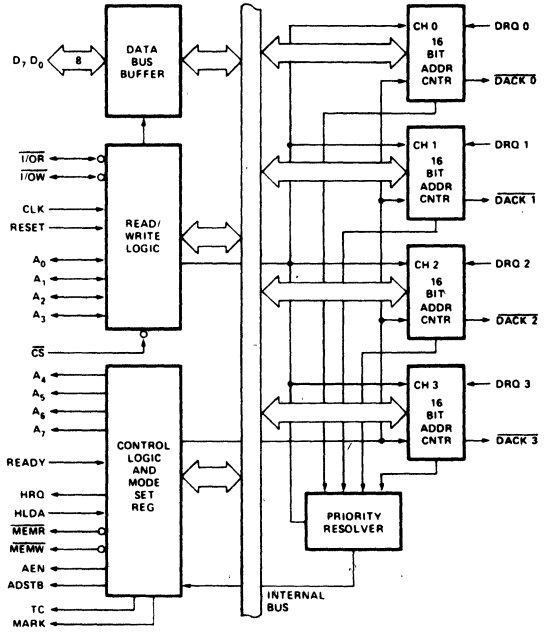


Figure 1. Block Diagram

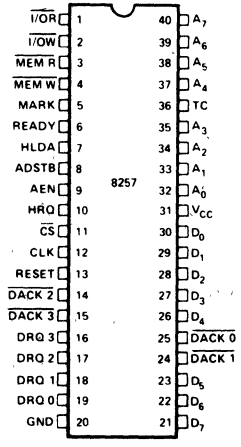


Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

General

The 8257 is a programmable, Direct Memory Access (DMA) device which, when coupled with a single Intel® 8212 I/O port device, provides a complete four-channel DMA controller for use in Intel® microcomputer systems. After being initialized by software, the 8257 can transfer a block of data, containing up to 16,384 bytes, between memory and a peripheral device directly, without further intervention required of the CPU. Upon receiving a DMA transfer request from an enabled peripheral, the 8257:

1. Acquires control of the system bus.
2. Acknowledges that requesting peripheral which is connected to the highest priority channel.
3. Outputs the least significant eight bits of the memory address onto system address lines A_0-A_7, outputs the most significant eight bits of the memory address to the 8212 I/O port via the data bus (the 8212 places these address bits on lines A_8-A_{15}), and
4. Generates the appropriate memory and I/O read/write control signals that cause the peripheral to receive or deposit a data byte directly from or to the addressed location in memory.

The 8257 will retain control of the system bus and repeat the transfer sequence, as long as a peripheral maintains its DMA request. Thus, the 8257 can transfer a block of data to/from a high speed peripheral (e.g., a sector of data on a floppy disk) in a single "burst". When the specified number of data bytes have been transferred, the 8257 activates its Terminal Count (TC) output, informing the CPU that the operation is complete.

The 8257 offers three different modes of operation: (1) DMA read, which causes data to be transferred from memory to a peripheral; (2) DMA write, which causes data to be transferred from a peripheral to memory; and (3) DMA verify, which does not actually involve the transfer of data. When an 8257 channel is in the DMA verify mode, it will respond the same as described for transfer operations, except that no memory or I/O read/write control signals will be generated, thus preventing the transfer of data. The 8257, however, will gain control of the system bus and will acknowledge the peripheral's DMA request for each DMA cycle. The peripheral can use these acknowledge signals to enable an internal access of each byte of a data block in order to execute some verification procedure, such as the accumulation of a CRC (Cyclic Redundancy Code) checkword. For example, a block of DMA verify cycles might follow a block of DMA read cycles (memory to peripheral) to allow the peripheral to verify its newly acquired data.

Block Diagram Description

1. DMA Channels

The 8257 provides four separate DMA channels (labeled CH-0 to CH-3). Each channel includes two sixteen-bit registers: (1) a DMA address register, and (2) a terminal count register. Both registers must be initialized before a channel is enabled. The DMA address register is loaded with the address of the first memory location to be accessed. The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output to be active in the first DMA cycle for that channel. In general, if N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register. The most significant two bits of the terminal count register specify the type of DMA operation for that channel.

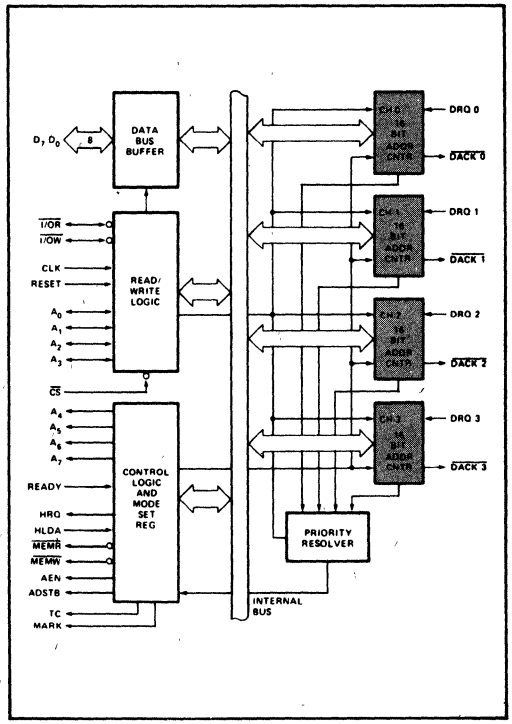


Figure 3. 8257 Block Diagram Showing DMA Channels

These two bits are not modified during a DMA cycle, but can be changed between DMA blocks.

Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output.

(DRQ 0-DRQ 3)

DMA Request: These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle. If not in the rotating priority mode then DRQ 0 has the highest priority and DRQ 3 has the lowest. A request can be generated by raising the request line and holding it high until DMA acknowledge. For multiple DMA cycles (Burst Mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

($\overline{\text{DACK 0}}$ - $\overline{\text{DACK 3}}$)

DMA Acknowledge: An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle. The $\overline{\text{DACK}}$ output acts as a "chip select" for the peripheral device requesting service. This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

2. Data Bus Buffer

This three-state, bi-directional, eight bit buffer interfaces the 8257 to the system data bus.

(D_0-D_7)

Data Bus Lines: These are bi-directional three-state lines. When the 8257 is being programmed by the CPU, eight-bits of data for a DMA address register, a terminal count register or the Mode Set register are received on the data bus. When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus. During DMA cycles (when the 8257 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the 8212 latch via the data bus. These address bits will be transferred at the beginning of the DMA cycle; the bus will then be released to handle the memory data transfer during the balance of the DMA cycle.

BIT 15	BIT 14	TYPE OF DMA OPERATION
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

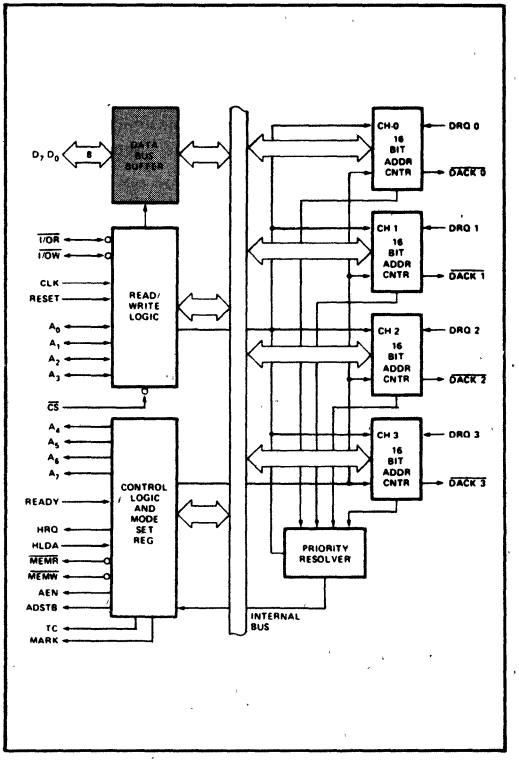


Figure 4. 8257 Block Diagram Showing Data Bus Buffer

3. Read/Write Logic

When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave" device on the system bus), the Read/Write Logic accepts the I/O Read ($\overline{I/OR}$) or I/O Write ($\overline{I/OW}$) signal, decodes the least significant four address bits, (A_0-A_3), and either writes the contents of the data bus into the addressed register (if $\overline{I/OW}$ is true) or places the contents of the addressed register onto the data bus (if $\overline{I/OR}$ is true).

During DMA cycles (i.e., when the 8257 is the bus "master"), the Read/Write Logic generates the I/O read and memory write (DMA write cycle) or I/O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.

Note that during DMA transfers Non-DMA I/O devices should be de-selected (disabled) using "AEN" signal to inhibit I/O device decoding of the memory address as an erroneous device address.

($\overline{I/OR}$)

I/O Read: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the 8-bit status register or the upper/lower byte of a 16-bit DMA address register or terminal count register to be read. In the "master" mode, $\overline{I/OR}$ is a control output which is used to access data from a peripheral during the DMA write cycle.

($\overline{I/OW}$)

I/O Write: An active-low, bi-directional three-state line. In the "slave" mode, it is an input which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 16-bit DMA address register or terminal count register. In the "master" mode, $\overline{I/OW}$ is a control output which allows data to be output to a peripheral during a DMA read cycle.

(CLK)

Clock Input: Generally from an Intel® 8224 Clock Generator device. (ϕ2 TTL) or Intel® 8085A CLK output

(RESET)

Reset: An asynchronous input (generally from an 8224 or 8085 device) which disables all DMA channels by clearing the mode register and 3-states all control lines.

(A_0-A_3)

Address Lines: These least significant four address lines are bi-directional. In the "slave" mode they are inputs which select one of the registers to be read or programmed. In the "master" mode, they are outputs which constitute the least significant four bits of the 16-bit memory address generated by the 8257.

($\overline{CS}$)

Chip Select: An active-low input which enables the I/O Read or I/O Write input when the 8257 is being read or programmed in the "slave" mode. In the "master" mode, $\overline{CS}$ is automatically disabled to prevent the chip from selecting itself while performing the DMA function.

4. Control Logic

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed

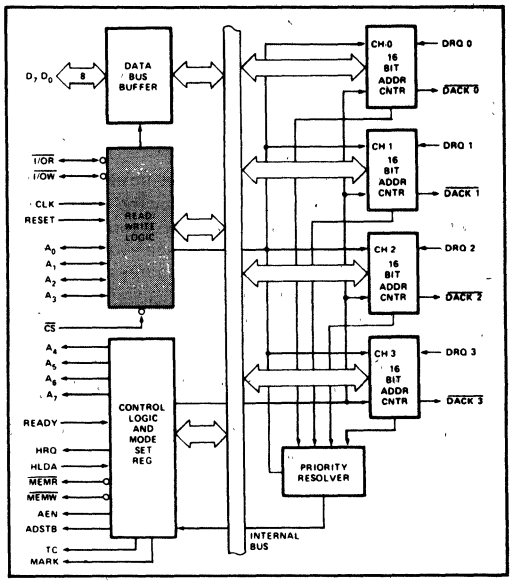


Figure 5. 8257 Block Diagram Showing Read/Write Logic Function

(A_4-A_7)

Address Lines: These four address lines are three-state outputs which constitute bits 4 through 7 of the 16-bit memory address generated by the 8257 during all DMA cycles.

(READY)

Ready: This asynchronous input is used to elongate the memory read and write cycles in the 8257 with wait states if the selected memory requires longer cycles. READY must conform to specified setup and hold times.

(HRQ)

Hold Request: This output requests control of the system bus. In systems with only one 8257, HRQ will normally be applied to the HOLD input on the CPU. HRQ must conform to specified setup and hold times.

(HLDA)

Hold Acknowledge: This input from the CPU indicates that the 8257 has acquired control of the system bus.

($\overline{\text{MEMR}}$)

Memory Read: This active-low three-state output is used to read data from the addressed memory location during DMA Read cycles.

($\overline{\text{MEMW}}$)

Memory Write: This active-low three-state output is used to write data into the addressed memory location during DMA Write cycles.

(ADSTB)

Address Strobe. This output strobes the most significant byte of the memory address into the 8212 device from the data bus.

(AEN)

Address Enable. This output is used to disable (float) the System Data Bus and the System Control Bus. It may also be used to disable (float) the System Address Bus by use of an enable on the Address Bus drivers in systems to inhibit non-DMA devices from responding during DMA cycles. It may be further used to isolate the 8257 data bus from the System Data Bus to facilitate the transfer of the 8 most significant DMA address bits over the 8257 data I/O pins without subjecting the System Data Bus to any timing constraints for the transfer. When the 8257 is used in an I/O device structure (as opposed to memory mapped), this AEN output should be used to disable the selection of an I/O device when the DMA address is on the address bus. The I/O device selection should be determined by the DMA acknowledge outputs for the 4 channels.

(TC)

Terminal Count: This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block. If the TC STOP bit in the Mode Set register is set, the selected channel will be automatically disabled at the end of that DMA cycle. TC is activated when the 14-bit value in the selected channel's terminal count register equals zero. Recall that the low-order 14-bits of the terminal count register should be loaded with the values (n-1), where n = the desired number of the DMA cycles.

(MARK)

Modulo 128 Mark: This output notifies the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. MARK always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisable by 128 (and the terminal count register was loaded with n-1), will MARK occur at 128 (and each succeeding multiple of 128) cycles from the beginning of the data block.

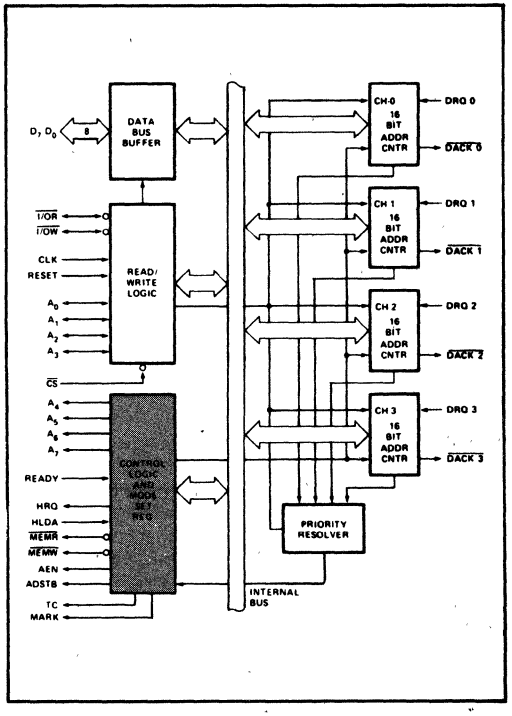
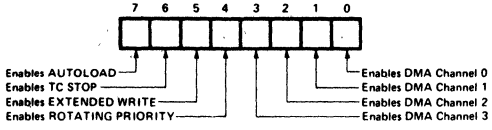


Figure 6. 8257 Block Diagram Showing Control Logic and Mode Set Register

5. Mode Set Register

When set, the various bits in the Mode Set register enable each of the four DMA channels, and allow four different options for the 8257:

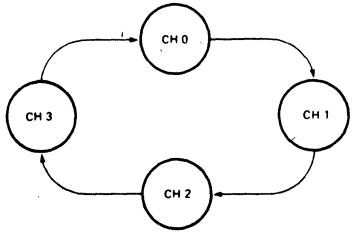


The Mode Set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized. The Mode Set Register is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up. A channel should not be left enabled unless its DMA address and terminal count registers contain valid values; otherwise, an inadvertent DMA request (DRQn) from a peripheral could initiate a DMA cycle that would destroy memory data.

The various options which can be enabled by bits in the Mode Set register are explained below.

Rotating Priority Bit 4

In the Rotating Priority Mode, the priority of the channels has a circular sequence After each DMA cycle, the priority of each channel changes. The channel which had just been serviced will have the lowest priority



If the ROTATING PRIORITY bit is not set (set to a zero), each DMA channel has a fixed priority In the fixed priority mode, Channel 0 has the highest priority and Channel 3 has the lowest priority If the ROTATING PRIORITY bit is set to a one, the priority of each channel changes after each DMA cycle (not each DMA request). Each channel moves up to the next highest priority assignment, while the channel which has just been serviced moves to the lowest priority assignment

	CHANNEL→ JUST SERVICED	CH-0	CH-1	CH-2	CH-3
Priority → Assignments	Highest	CH-1	CH-2	CH-3	CH-0
	↑	CH-2	CH-3	CH-0	CH-1
	↓	CH-3	CH-0	CH-1	CH-2
	Lowest	CH-0	CH-1	CH-2	CH-3

Note that rotating priority will prevent any one channel from monopolizing the DMA mode; consecutive DMA cycles will service different channels if more than one channel is enabled and requesting service. There is no overhead penalty associated with this mode of operation. All DMA operations began with Channel 0 initially assigned to the highest priority for the first DMA cycle.

Extended Write Bit 5

If the EXTENDED WRITE bit is set, the duration of both the $\overline{MEMW}$ and $\overline{I/OW}$ signals is extended by activating them earlier in the DMA cycle. Data transfers within microcomputer systems proceed asynchronously to allow use of various types of memory and I/O devices with different access times. If a device cannot be accessed within a specific amount of time it returns a "not ready" indication to the 8257 that causes the 8257 to insert one or more wait states in its internal sequencing. Some devices are fast enough to be accessed without the use of wait states, but if they generate their READY response with the leading edge of the $\overline{I/OW}$ or $\overline{MEMW}$ signal (which generally occurs late in the transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive READY in time. For systems with these types of devices, the Extended Write option provides alternative timing for the I/O and memory write signals which allows the devices to return an early READY and prevents the unnecessary occurrence of wait states in the 8257, thus increasing system throughput

TC Stop Bit 6

If the TC STOP bit is set, a channel is disabled (i.e., its enable bit is reset) after the Terminal Count (TC) output goes true, thus automatically preventing further DMA operation on that channel. The enable bit for that channel must be re-programmed to continue or begin another DMA operation If the TC STOP bit is not set, the occurrence of the TC output has no effect on the channel enable bits. In this case, it is generally the responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation

Auto Load Bit 7

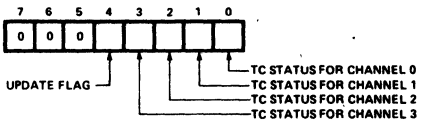
The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks Channel 2 registers are initialized as usual for the first data block, Channel 3 registers, however, are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode). After the first block of DMA cycles is executed by Channel 2 (i.e., after the TC output goes true), the parameters stored in the Channel 3 registers are transferred to Channel 2 during an "update" cycle Note that the TC STOP feature, described above, has no effect on Channel 2 when the Auto Load bit is set

If the Auto Load bit is set, the initial parameters for Channel 2 are automatically duplicated in the Channel 3 registers when Channel 2 is programmed. This permits repeat block operations to be set up with the programming of a single channel. Repeat block operations can be used in applications such as CRT refreshing. Channels 2 and 3 can still be loaded with separate values if Channel 2 is loaded before loading Channel 3. Note that in the Auto Load mode, Channel 3 is still available to the user if the Channel 3 enable bit is set, but use of this channel will change the values to be auto loaded into Channel 2 at update time. All that is necessary to use the Auto Load feature for chaining operations is to reload Channel 3 registers at the conclusion of each update cycle with the new parameters for the next data block transfer.

Each time that the 8257 enters an update cycle, the update flag in the status register is set and parameters in Channel 3 are transferred to Channel 2, non-destructively for Channel 3. The actual re-initialization of Channel 2 occurs at the beginning of the next channel 2 DMA cycle after the TC cycle. This will be the first DMA cycle of the new data block for Channel 2. The update flag is cleared at the conclusion of this DMA cycle. For chaining operations, the update flag in the status register can be monitored by the CPU to determine when the re-initialization process has been completed so that the next block parameters can be safely loaded into Channel 3.

6. Status Register

The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously.



The TC status bits are set when the Terminal Count (TC) output is activated for that channel. These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, however, is not affected by a status register read operation. The UPDATE FLAG can be cleared by resetting the 8257, by changing to the non-auto load mode (i.e., by resetting the AUTO LOAD bit in the Mode Set register) or it can be left to clear itself at the completion of the update cycle. The purpose of the UPDATE FLAG is to prevent the CPU from inadvertently skipping a data block by overwriting a starting address or terminal count in the Channel 3 registers before those parameters are properly auto-loaded into Channel 2.

The user is cautioned against reading the TC status register and using this information to reenable channels that have not completed operation. Unless the DMA channels are inhibited a channel could reach terminal count (TC) between the status read and the mode write. DMA can be inhibited by a hardware gate on the HRQ line or by disabling channels with a mode word before reading the TC status.

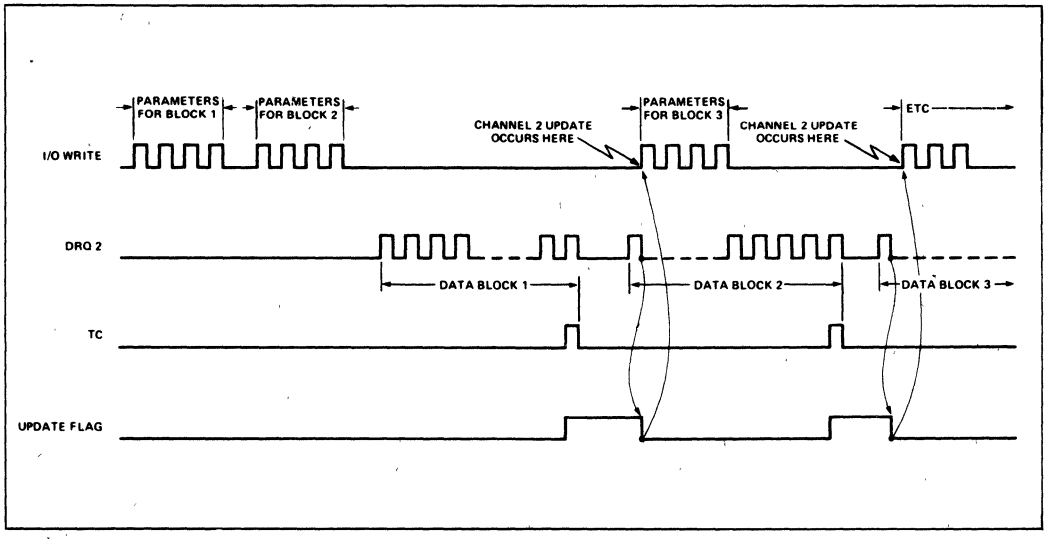


Figure 7. Autoload Timing

OPERATIONAL SUMMARY

Programming and Reading the 8257 Registers

There are four pairs of "channel registers": each pair consisting of a 16-bit DMA address register and a 16-bit terminal count register (one pair for each channel). The 8257 also includes two "general registers": one 8-bit Mode Set register and one 8-bit Status register. The registers are loaded or read when the CPU executes a write or read instruction that addresses the 8257 device and the appropriate register within the 8257. The 8228 generates the appropriate read or write control signal (generally I/OR or I/OW while the CPU places a 16-bit address on the system address bus, and either outputs the data to be written onto the system data bus or accepts the data being read from the data bus. All or some of the most significant 12 address bits A_4-A_{15} (depending on the systems memory, I/O configuration) are usually decoded to produce the chip select ($\overline{CS}$) input to the 8257. An I/O Write input (or Memory Write in memory mapped I/O configurations, described below) specifies that the addressed register is to be programmed, while an I/O Read input (or Memory Read) specifies that the addressed register is to be read. Address bit 3 specifies whether a "channel register" ($A_3 = 0$) or the Mode Set (program only)/Status (read only) register ($A_3 = 1$) is to be accessed.

The least significant three address bits, A_0-A_2, indicate the specific register to be accessed. When accessing the Mode Set or Status register, A_0-A_2 are all zero. When accessing a channel register bit A_0 differentiates between the DMA address register ($A_0 = 0$) and the terminal count register ($A_0 = 1$), while bits A_1 and A_2 specify one of the four channels. Because the "channel registers" are 16-bits, two program instruction cycles are required to load or read an entire register. The 8257 contains a first/last (F/L) flip flop which toggles at the completion of each channel program or read operation. The F/L flip flop determines whether the upper or lower byte of the register is to be accessed. The F/L flip flop is reset by the RESET input and whenever the Mode Set register is loaded. To maintain proper synchronization when accessing the "channel registers" all channel command instruction operations should occur in pairs, with the lower byte of a register always being accessed first. Do not allow $\overline{CS}$ to clock while either $\overline{I/OR}$ or $\overline{I/OW}$ is active, as this will cause an erroneous F/L flip flop state. In systems utilizing an interrupt structure, interrupts should be disabled prior to any paired programming operations to prevent an interrupt from splitting them. The result of such a split would leave the F/L F/F in the wrong state. This problem is particularly obvious when other DMA channels are programmed by an interrupt structure.

CONTROL INPUT	$\overline{CS}$	$\overline{I/OW}$	$\overline{I/OR}$	A_3
Program Half of a Channel Register	0	0	1	0
Read Half of a Channel Register	0	1	0	0
Program Mode Set Register	0	0	1	1
Read Status Register	0	1	0	1

8257 Register Selection

REGISTER	BYTE	ADDRESS INPUTS				F/L	*BI-DIRECTIONAL DATA BUS							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	Same as Channel 0							
	MSB	0	0	1	0	1								
CH-1 Terminal Count	LSB	0	0	1	1	0								
	MSB	0	0	1	1	1								
CH-2 DMA Address	LSB	0	1	0	0	0	Same as Channel 0							
	MSB	0	1	0	0	1								
CH-2 Terminal Count	LSB	0	1	0	1	0								
	MSB	0	1	0	1	1								
CH-3 DMA Address	LSB	0	1	1	0	0	Same as Channel 0							
	MSB	0	1	1	0	1								
CH-3 Terminal Count	LSB	0	1	1	1	0								
	MSB	0	1	1	1	1								
MODE SET (Program only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

*A_0-A_{15}: DMA Starting Address, C_0-C_{13}: Terminal Count value (N-1), Rd and Wr: DMA Verify (00), Write (01) or Read (10) cycle selection, AL: Auto Load, TCS: TC STOP, EW: EXTENDED WRITE, RP: ROTATING PRIORITY, EN3-EN0: CHANNEL ENABLE MASK, UP: UPDATE FLAG, TC3-TC0: TERMINAL COUNT STATUS BITS.

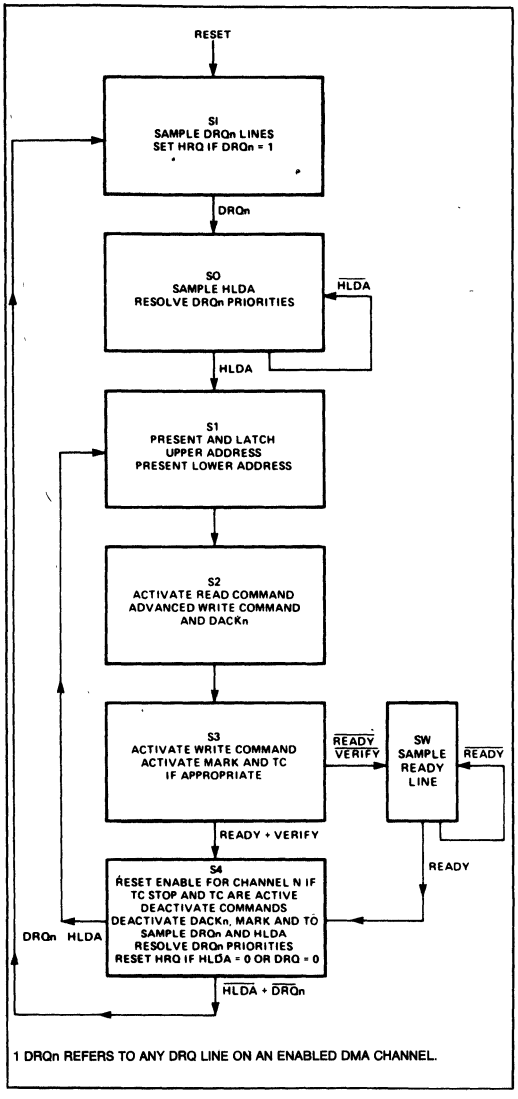


Figure 8. DMA Operation State Diagram

DMA OPERATION

Single Byte Transfers

A single byte transfer is initiated by the I/O device raising the DRQ line of one channel of the 8257. If the channel is enabled, the 8257 will output a HRQ to the CPU. The 8257 now waits until a HLDA is received insuring that the system bus is free for its use. Once HLDA is received the $\overline{DACK}$ line for the requesting channel is activated (LOW). The $\overline{DACK}$ line acts as a chip select for the requesting I/O device. The 8257 then generates the read and write commands and byte transfer occurs between the selected I/O device and memory. After the transfer is complete, the $\overline{DACK}$ line is set HIGH and the HRQ line is set LOW to indicate to the CPU that the bus is now free for use. DRQ must remain HIGH until $\overline{DACK}$ is issued to be recognized and must go LOW before S4 of the transfer sequence to prevent another transfer from occuring. (See timing diagram.)

Consecutive Transfers

If more than one channel requests service simultaneously, the transfer will occur in the same way a burst does. No overhead is incurred by switching from one channel to another. In each S4 the DRQ lines are sampled and the highest priority request is recognized during the next transfer. A burst mode transfer in a lower priority channel will be overridden by a higher priority request. Once the high priority transfer has completed control will return to the lower priority channel if its DRQ is still active. No extra cycles are needed to execute this sequence and the HRQ line remains active until all DRQ lines go LOW.

Control Override

The continuous DMA transfer mode described above can be interrupted by an external device by lowering the HLDA line. After each DMA transfer the 8257 samples the HLDA line to insure that it is still active. If it is not active, the 8257 completes the current transfer, releases the HRQ line (LOW) and returns to the idle state. If DRQ lines are still active the 8257 will raise the HRQ line in the third cycle and proceed normally. (See timing diagram.)

Not Ready

The 8257 has a Ready input similar to the 8080A and the 8085A. The Ready line is sampled in State 3. If Ready is LOW the 8257 enters a wait state. Ready is sampled during every wait state. When Ready returns HIGH the 8257 proceeds to State 4 to complete the transfer. Ready is used to interface memory or I/O devices that cannot meet the bus set up times required by the 8257.

Speed

The 8257 uses four clock cycles to transfer a byte of data. No cycles are lost in the master to master transfer maximizing bus efficiency. A 2MHz clock input will allow the 8257 to transfer at a rate of 500K bytes/second.

Memory Mapped I/O Configurations

The 8257 can be connected to the system bus as a memory device instead of as an I/O device for memory mapped I/O configurations by connecting the system memory control lines to the 8257's I/O control lines and the system I/O control lines to the 8257's memory control lines

This configuration permits use of the 8080's considerably larger repertoire of memory instructions when reading or loading the 8257's registers Note that with this connection, the programming of the Read (bit 15) and Write (bit 14) bits in the terminal count register will have a different meaning

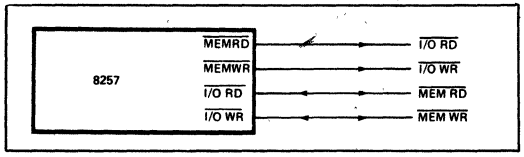


Figure 9. System Interface for Memory Mapped I/O

BIT 15 READ	BIT 14 WRITE	
0	0	DMA Verify Cycle
0	1	DMA Read Cycle
1	0	DMA Write Cycle
1	1	Illegal

Figure 10. TC Register for Memory Mapped I/O Only

SYSTEM APPLICATION EXAMPLES

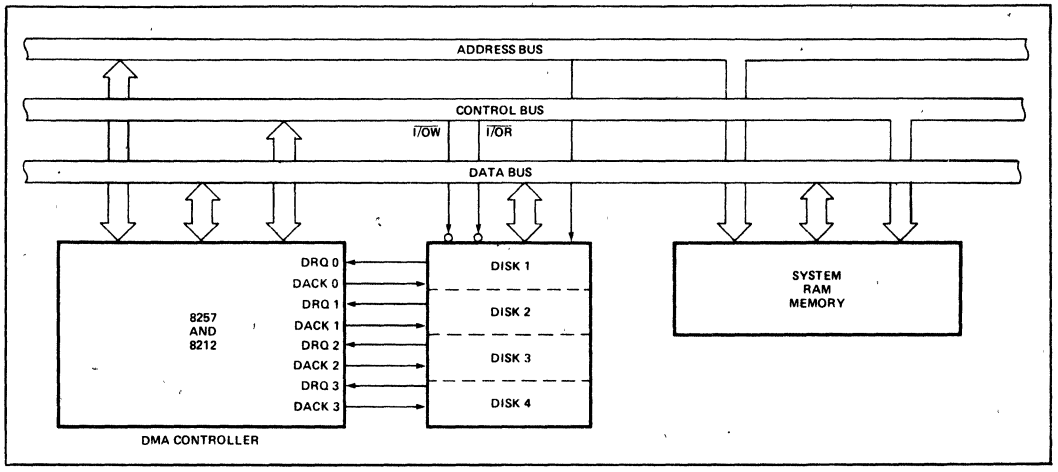


Figure 11. Floppy Disk Controller (4 Drives)

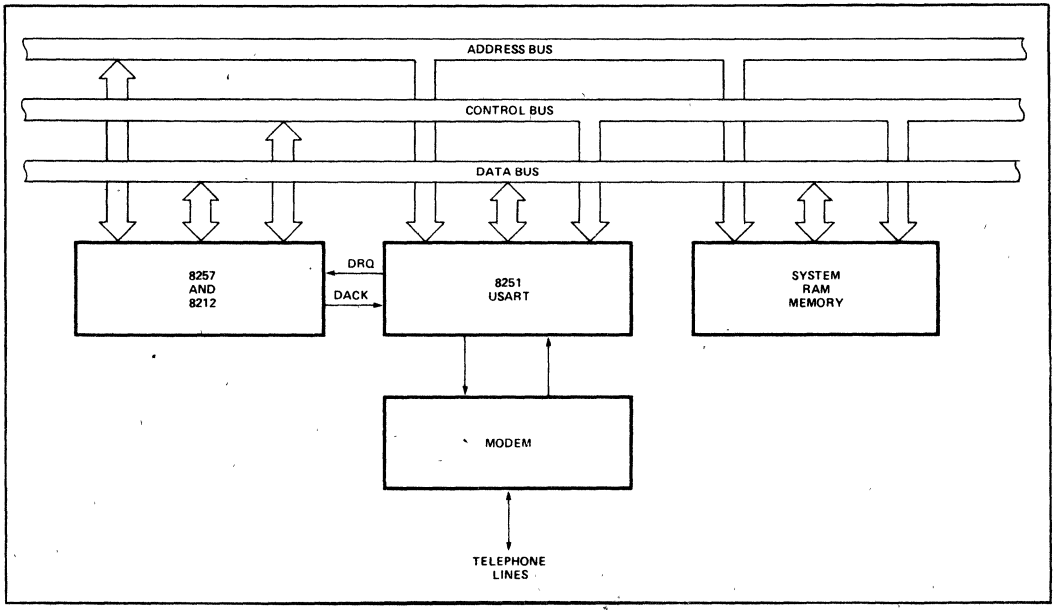
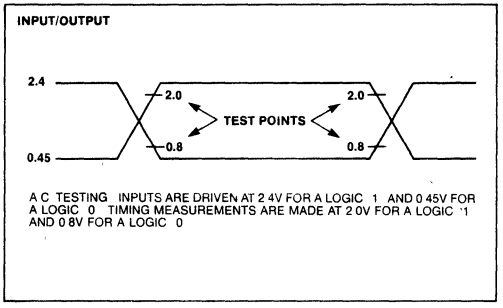
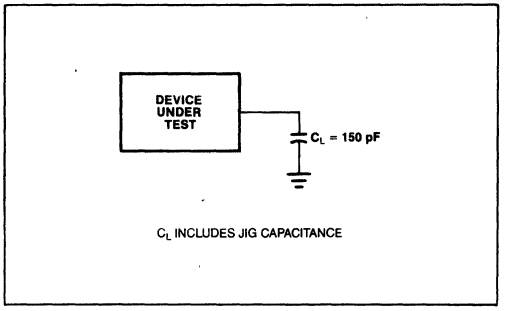


Figure 12. High-Speed Communication Controller

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



Tracking Parameters

Signals labeled as Tracking Parameters (footnotes 1 and 5-7 under A.C. Specifications) are signals that follow similar paths through the silicon die. The propagation speed of these signals varies in the manufacturing process but the relationship between all these parameters is constant. The variation is less than or equal to 50 ns.

Suppose the following timing equation is being evaluated,

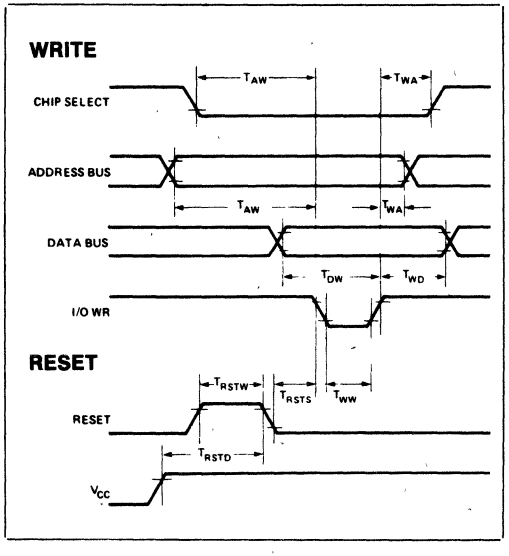
$$T_{A(MIN)} + T_{B(MAX)} \leq 150 \text{ ns}$$

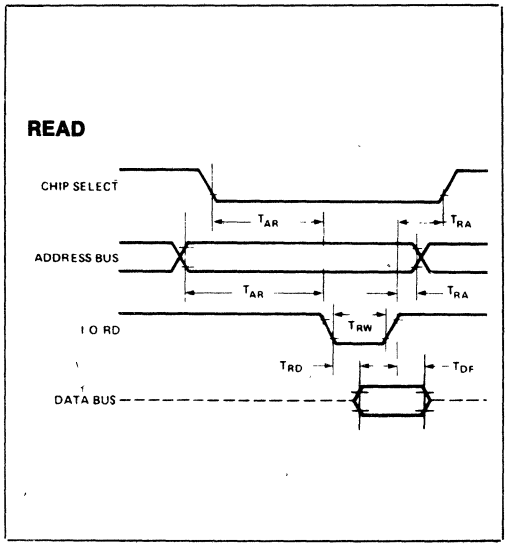
and only minimum specifications exist for T_A and T_B. If $T_{A(MIN)}$ is used, and if T_A and T_B are tracking parameters, $T_{B(MAX)}$ can be taken as $T_{B(MIN)}$ + 50 ns.

$$T_{A(MIN)} + (T_{B(MIN)}^{*} + 50 \text{ ns}) \leq 150 \text{ ns}$$

*if T_A and T_B are tracking parameters

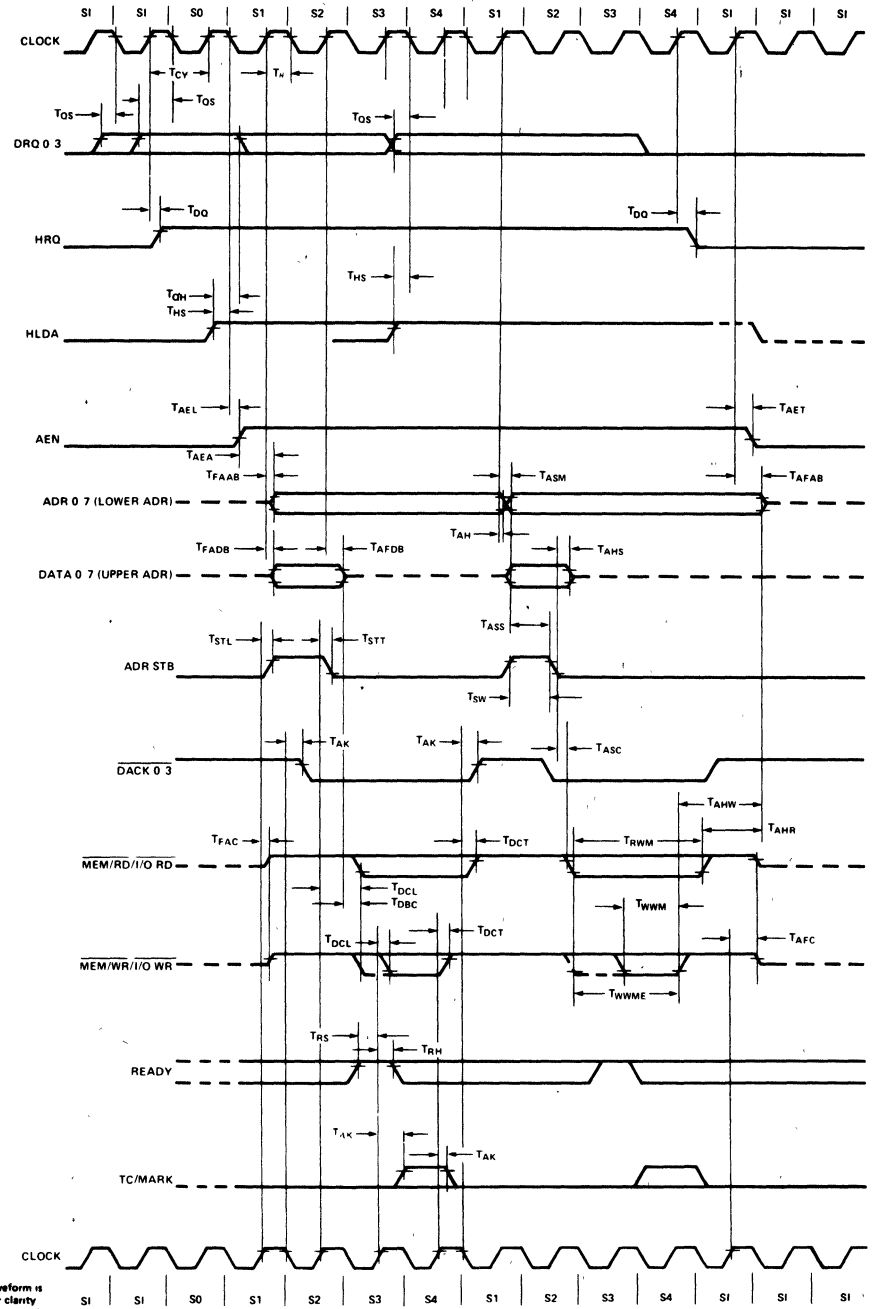
WAVEFORMS—PERIPHERAL MODE





WAVEFORMS—DMA

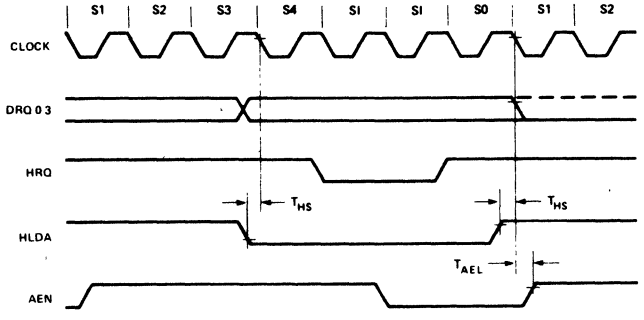
CONSECUTIVE CYCLES AND BURST MODE SEQUENCE



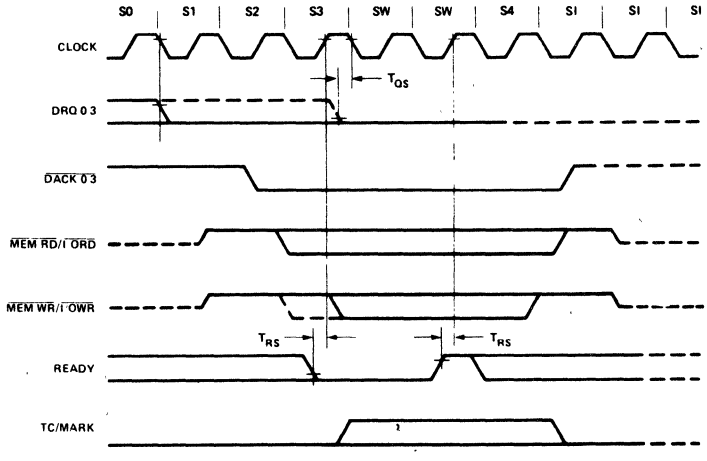
NOTE The clock waveform is duplicated for clarity The 8257 requires only one clock input

WAVEFORMS (Continued)

CONTROL OVERRIDE SEQUENCE



NOT READY SEQUENCE



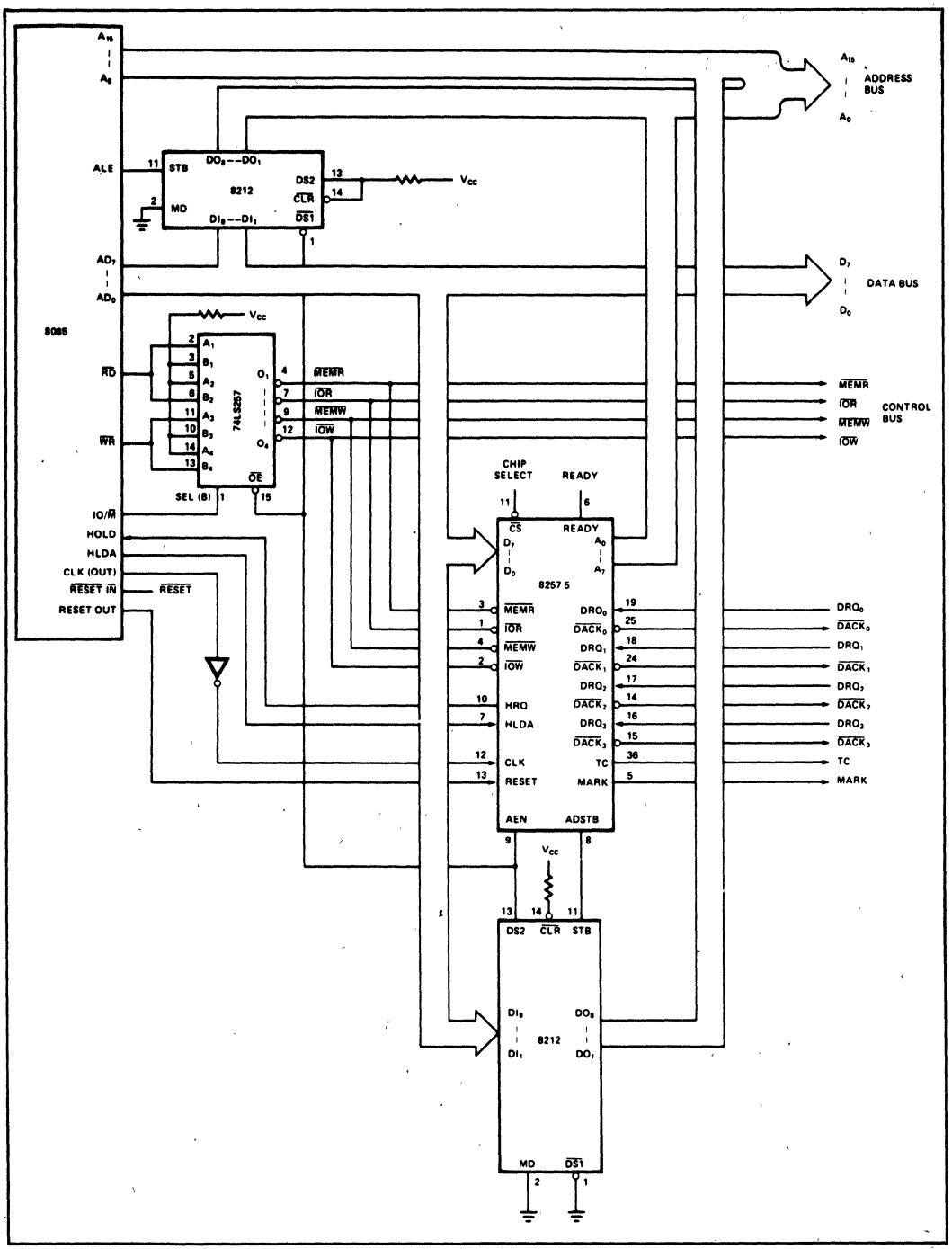


Figure 13. Detailed System Interface Schematic

AFN-01840D

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	Volts	
V_{IH}	Input High Voltage	2.0	V_{CC}+.5	Volts	
V_{OL}	Output Low Voltage		0.45	Volts	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage	2.4	V_{CC}	Volts	I_{OH}=-150μA for AB, DB and AEN I_{OH}=-80μA for others
V_{HH}	HRQ Output High Voltage	3.3	V_{CC}	Volts	I_{OH} = -80μA
I_{CC}	V_{CC} Current Drain		120	mA	
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{OFL}	Output Leakage During Float		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$

CAPACITANCE

(T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

8080 Bus Parameters

READ CYCLE

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{AR}	Adr or $\overline{CS}\downarrow$ Setup to $\overline{RD}\downarrow$	0		0		ns	
T_{RA}	Adr or $\overline{CS}\uparrow$ Hold from $\overline{RD}\uparrow$	0		0		ns	
T_{RD}	Data Access from $\overline{RD}\downarrow$	0	300	0	220	ns	
T_{DF}	DB→Float Delay from $\overline{RD}\uparrow$	20	150	20	120	ns	
T_{RR}	$\overline{RD}$ Width	250		250		ns	

WRITE CYCLE

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{AW}	Adr Setup to $\overline{WR}\downarrow$	20		20		ns	
T_{WA}	Adr Hold from $\overline{WR}\uparrow$	0		0		ns	
T_{DW}	Data Setup to $\overline{WR}\uparrow$	200		200		ns	
T_{WD}	Data Hold from $\overline{WR}\uparrow$	10		10		ns	
T_{WW}	$\overline{WR}$ Width	200		200		ns	

OTHER TIMING

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit	Test Conditions
T_{RSTW}	Reset Pulse Width	300		300		ns	
T_{RSTD}	Power Supply↑ (V_{CC}) Setup to Reset↓	500		500		μs	
T_r	Signal Rise Time		20		20	ns	
T_f	Signal Fall Time		20		20	ns	
T_{RSTS}	Reset to First $\overline{I/O}\overline{WR}$	2		2		t_{CY}	

A.C. CHARACTERISTICS—DMA (MASTER) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

TIMING REQUIREMENTS

Symbol	Parameter	8257 Min.	8257 Max.	8257-5 Min.	8257-5 Max.	Unit
T_{CY}	Cycle Time (Period)	0.320	4	0.320	4	μs
T_θ	Clock Active (High)	120	$.8T_{CY}$	80	$.8T_{CY}$	ns
T_{QS}	DRQ↑ Setup to CLK↓ (SI, S4)	120		120		ns
T_{QH}	DRQ↓ Hold from HLDA↑[1]	0		0		ns
T_{HS}	HLDA↑ or ↓Setup to CLK↓(SI, S4)	100		100		ns
T_{RS}	READY Setup Time to CLK↓(S3, Sw)	30		30		ns
T_{RH}	READY Hold Time from CLK↓(S3, Sw)	30		30		ns

A.C. CHARACTERISTICS—DMA (MASTER) MODE

(8257: T_A = 0°C to 70°C, V_{CC} = 5.0V ±5%, GND = 0V)
(8257-5: T_A = 0°C to 70°C, V_{CC} = 5.0V ±10%, GND = 0V)

TIMING RESPONSES

Symbol	Parameter	8257		8257-5		Unit
		Min.	Max.	Min.	Max.	
T_{DQ}	HRQ↑ or ↓Delay from CLK↑ (SI, S4) (measured at 2.0V)		160		160	ns
T_{DQ1}	HRQ↑ or ↓Delay from CLK↑ (SI, S4) (measured at 3.3V)[3]		250		250	ns
T_{AEL}	AEN↑ Delay from CLK↓ (S1)		300		300	ns
T_{AET}	AEN↓ Delay from CLK↑ (SI)		200		200	ns
T_{AEA}	Adr (AB) (Active) Delay from AEN↑ (S1)[1]	20		20		ns
T_{FAAB}	Adr (AB) (Active) Delay from CLK↑ (S1)[2]		250		250	ns
T_{AFAB}	Adr (AB) (Float) Delay from CLK↑ (SI)[2]		150		150	ns
T_{ASM}	Adr (AB) (Stable) Delay from CLK↑ (S1)[2]		250		250	ns
T_{AH}	Adr (AB) (Stable) Hold from CLK↑ (S1)[2]	$T_{ASM}-50$		$T_{ASM}-50$		ns
T_{AHR}	Adr (AB) (Valid) Hold from $\overline{RD}$↑ (S1, SI)[1]	60		60		ns
T_{AHW}	Adr (AB) (Valid) Hold from $\overline{Wr}$↑ (S1, SI)[1]	300		300		ns
T_{FADB}	Adr (DB) (Active) Delay from CLK↑ (S1)[2]		300		300	ns
T_{AFDB}	Adr (DB) (Float) Delay from CLK↑ (S2)[2]	$T_{STT}+20$	250	$T_{STT}+20$	170	ns
T_{ASS}	Adr (DB) Setup to Adr Stb↓ (S1-S2)[1]	100		100		ns
T_{AHS}	Adr (DB) (Valid) Hold from Adr Stb↓ (S2)[1]	20		20		ns
T_{STL}	Adr Stb↑ Delay from CLK↑ (S1)		200		200	ns
T_{STT}	Adr Stb↓ Delay from CLK↑ (S2)		140		140	ns
T_{SW}	Adr Stb Width (S1-S2)[1]	$T_{CY}-100$		$T_{CY}-100$		ns
T_{ASC}	$\overline{Rd}$↓ or $\overline{Wr}$(Ext)↓ Delay from Adr Stb↓ (S2)[1]	70		70		ns
T_{DBC}	$\overline{RD}$↓ or $\overline{WR}$(Ext)↓ Delay from Adr (DB) (Float) (S2)[1]	20		20		ns
T_{AK}	DACK↑ or ↓Delay from CLK↓ (S2, S1) and TC/Mark↑ Delay from CLK↑ (S3) and TC/Mark↓ Delay from CLK↑ (S4)[4]		250		250	ns
T_{DCL}	$\overline{RD}$↓ or $\overline{Wr}$(Ext)↓ Delay from CLK↑ (S2) and Wr↓ Delay from CLK↑ (S3)[2,5]		200		200	ns
T_{DCT}	Rd↑ Delay from CLK↓ (S1, SI) and Wr↑ Delay from CLK↑ (S4)[2,6]		200		200	ns
T_{FAC}	$\overline{Rd}$ or Wr (Active) from CLK↑ (S1)[2]		300		300	ns
T_{AFC}	$\overline{Rd}$ or $\overline{Wr}$ (Active) from CLK↑ (S1)[2]		150		150	ns
T_{RWM}	$\overline{Rd}$ Width (S2-S1 or SI)[1]	$2T_{CY}+T_\theta-50$		$2T_{CY}+T_\theta-50$		ns
T_{WWM}	$\overline{Wr}$ Width (S3-S4)[1]	$T_{CY}-50$		$T_{CY}-50$		ns
T_{WWME}	$\overline{WR}$(Ext) Width (S2-S4)[1]	$2T_{CY}-50$		$2T_{CY}-50$		ns

NOTES:
1. Tracking Parameter.
2. Load = + 50 pF.
3. Load = V_{OH} = 3.3V.
4. $\Delta T_{AK} < 50$ ns.
5. $\Delta T_{DCL} < 50$ ns.
6. $\Delta T_{DCT} < 50$ ns.

8259A/8259A-2/8259A-8
PROGRAMMABLE INTERRUPT CONTROLLER

- **IAPX 86, IAPX 88 Compatible**
- **MCS-80®, MCS-85® Compatible**
- **Eight-Level Priority Controller**
- **Expandable to 64 Levels**
- **Programmable Interrupt Modes**
- **Individual Request Mask Capability**
- **Single +5V Supply (No Clocks)**
- **28-Pin Dual-In-Line Package**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

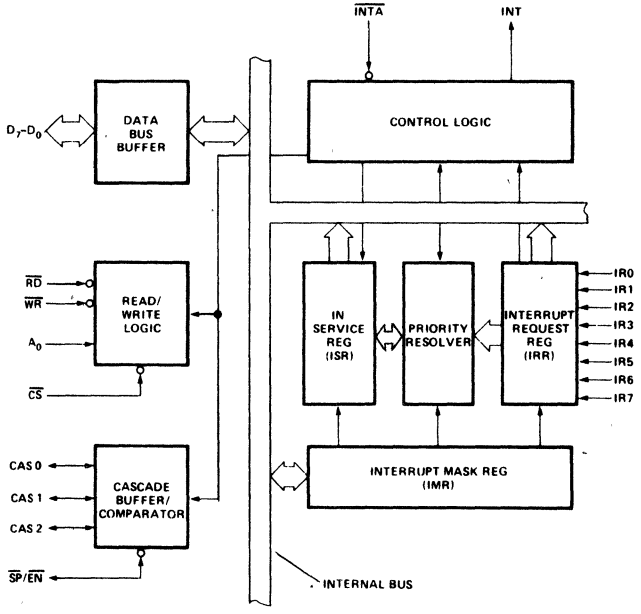


Figure 1. Block Diagram

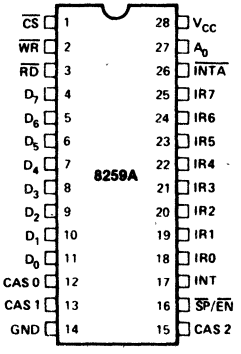


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	**Supply:** +5V Supply.
GND	14	I	**Ground.**
$\overline{CS}$	1	I	**Chip Select:** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA functions are independent of CS.
$\overline{WR}$	2	I	**Write:** A low on this pin when CS is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	**Read:** A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.
D_7–D_0	4–11	I/O	**Bidirectional Data Bus:** Control, status and interrupt-vector information is transferred via this bus.
CAS_0–CAS_2	12, 13, 15	I/O	**Cascade Lines:** The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP}/\overline{EN}$	16	I/O	**Slave Program/Enable Buffer:** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	**Interrupt:** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0–IR_7	18–25	I	**Interrupt Requests:** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
$\overline{INTA}$	26	I	**Interrupt Acknowledge:** This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**A0 Address Line:** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for iAPX 86, 88).

FUNCTIONAL DESCRIPTION

Interrupts in Microcomputer Systems

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polled* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

The 8259A

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels or requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

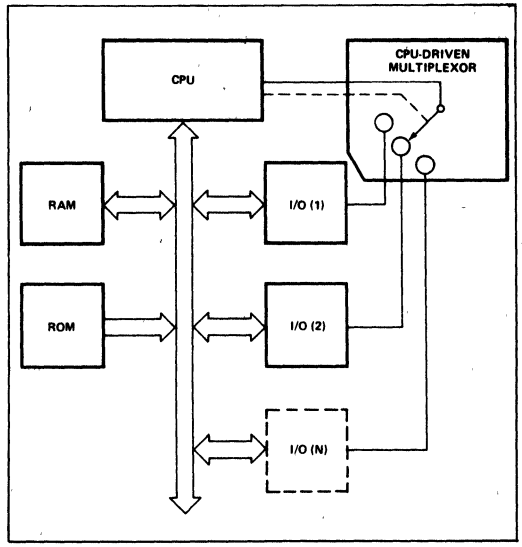


Figure 3a. Polled Method

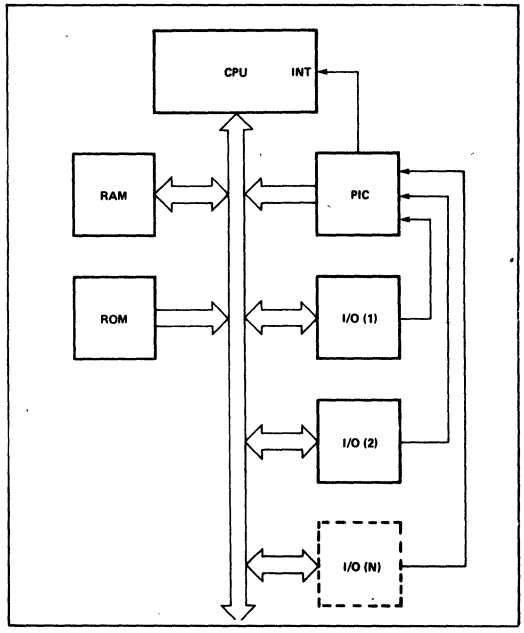


Figure 3b. Interrupt Method

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during $\overline{INTA}$ pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V_{OH} level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

$\overline{INTA}$ (INTERRUPT ACKNOWLEDGE)

$\overline{INTA}$ pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (μPM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

$\overline{CS}$ (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

$\overline{WR}$ (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

$\overline{RD}$ (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

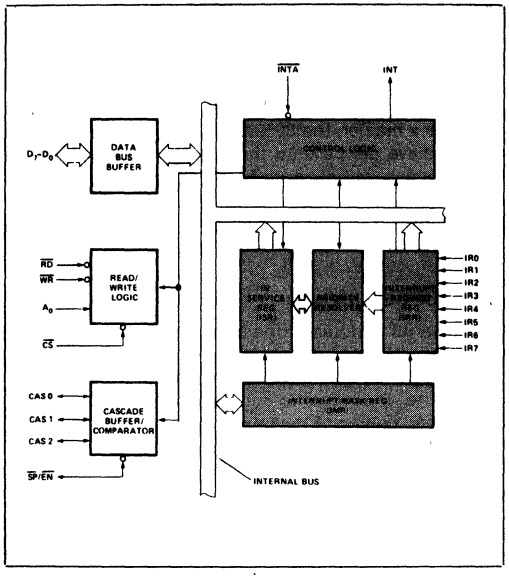


Figure 4a. 8259A Block Diagram

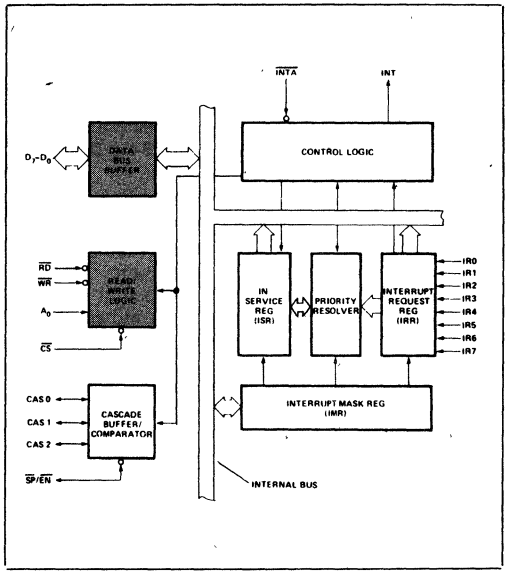


Figure 4b. 8259A Block Diagram

A_0

This input signal is used in conjunction with $\overline{WR}$ and $\overline{RD}$ signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0-2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive $\overline{INTA}$ pulses. (See section "Cascading the 8259A".)

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7-0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an $\overline{INTA}$ pulse.
4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.
5. This CALL instruction will initiate two more $\overline{INTA}$ pulses to be sent to the 8259A from the CPU group.
6. These two $\overline{INTA}$ pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first $\overline{INTA}$ pulse and and the higher 8-bit address is released at the second $\overline{INTA}$ pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an iAPX 86 system are the same until step 4.

4. Upon receiving an $\overline{INTA}$ from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.
5. The iAPX 86/10 will initiate a second $\overline{INTA}$ pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second $\overline{INTA}$ pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

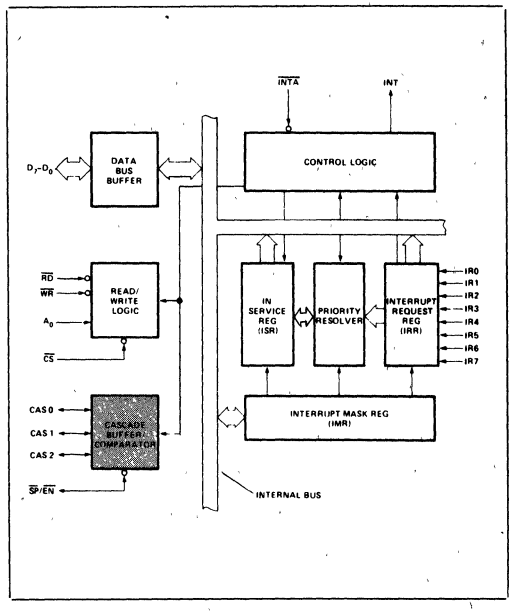


Figure 4c. 8259A Block Diagram

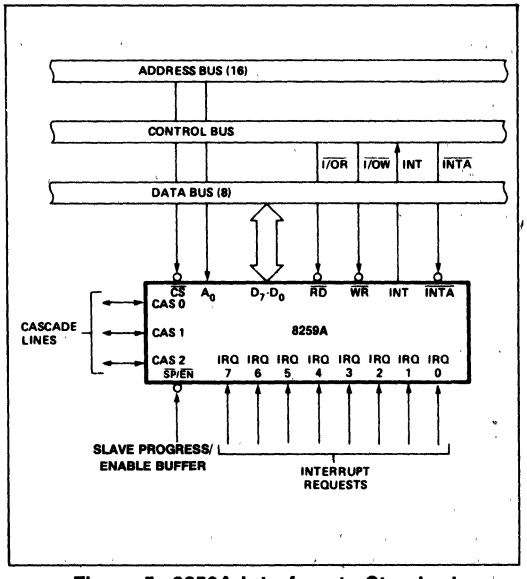


Figure 5. 8259A Interface to Standard System Bus

INTERRUPT SEQUENCE OUTPUTS

MCS-80®, MCS-85®

This sequence is timed by three $\overline{INTA}$ pulses. During the first $\overline{INTA}$ pulse the CALL opcode is enabled onto the data bus.

Content of First Interrupt Vector Byte

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second $\overline{INTA}$ pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A_5-A_7 are programmed, while A_0-A_4 are automatically inserted by the 8259A. When Interval = 8 only A_6 and A_7 are programmed, while A_0-A_5 are automatically inserted.

Content of Second Interrupt Vector Byte

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third INTA pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence (A_8-A_{15}), is enabled onto the bus.

Content of Third Interrupt Vector Byte

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

iAPX 86, iAPX 88

iAPX 86 mode is similar to MCS-80 mode except that only two Interrupt Acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80, 85 systems in that the 8259A uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the INTA pulse. On this first cycle it does not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in iAPX 86 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the ADI mode control is ignored and A_5-A_{11} are unused in iAPX 86 mode):

Content of Interrupt Vector Byte for iAPX 86 System Mode

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. *Initialization Command Words (ICWs):* Before normal operation can begin, each 8259A in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by $\overline{WR}$ pulses.
2. *Operation Command Words (OCWs):* These are the command words which command the 8259A to operate in various interrupt modes. These modes are:
   a. Fully nested mode
   b. Rotating priority mode
   c. Special mask mode
   d. Polled mode

The OCWs can be written into the 8259A anytime after initialization.

INITIALIZATION COMMAND WORDS (ICWS)

GENERAL

Whenever a command is issued with A0 = 0 and D4 = 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
b. The Interrupt Mask Register is cleared.
c. IR7 input is assigned priority 7.
d. The slave mode address is set to 7.
e. Special Mask Mode is cleared and Status Read is set to IRR.
f. If IC4=0, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

*Note: Master/Slave in ICW4 is only used in the buffered mode

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A_5-A_{15}: *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A_0-A_{15}). When the routine interval is 4, A_0-A_4 are automatically inserted by the 8259A, while A_5-A_{15} are programmed externally. When the routine interval is 8, A_0-A_5 are automatically inserted by the 8259A, while A_6-A_{15} are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an iAPX 86 system A_{15}-A_{11} are inserted in the five most significant bits of the vectoring byte and the 8259A sets the three least significant bits according to the interrupt level. A_{10}-A_5 are ignored and ADI (Address interval) has no effect.

LTIM: If LTIM = 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address interval. ADI = 1 then interval = 4; ADI = 0 then interval = 8.

SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

a. In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for iAPX 86 only byte 2) through the cascade lines.

b. In the slave mode (either when $\overline{SP}$ = 0, or if BUF = 1 and M/S = 0 in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for iAPX 86 are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP}/\overline{EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 8259A is programmed to be a master, M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μPM: Microprocessor mode: μPM = 0 sets the 8259A for MCS-80, 85 system operation, μPM = 1 sets the 8259A for iAPX 86 system operation.

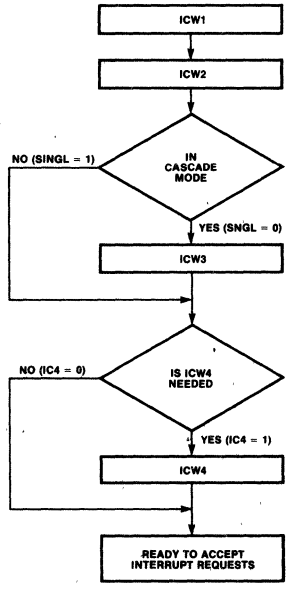
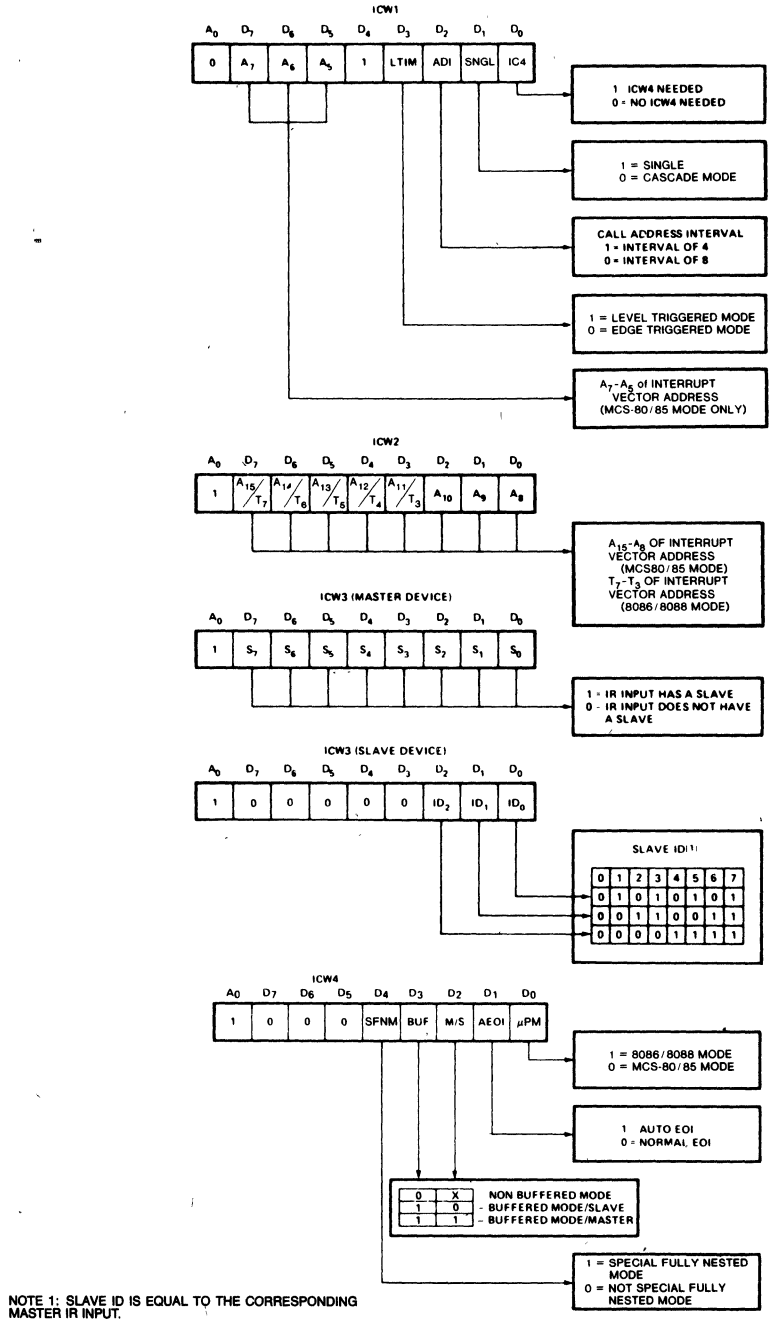


Figure 6. Initialization Sequence



NOTE 1: SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT.

Figure 7. Initialization Command Word Format

OPERATION COMMAND WORDS (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)

OCW1

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

OCW2

0	R	SL	EOI	0	0	L2	L1	L0

OCW3

0	0	ESMM	SMM	0	1	P	RR	RIS

OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). $M_7 - M_0$ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

R, SL, EOI — These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L_2, L_1, L_0—These bits determine the interrupt level acted upon when the SL bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.

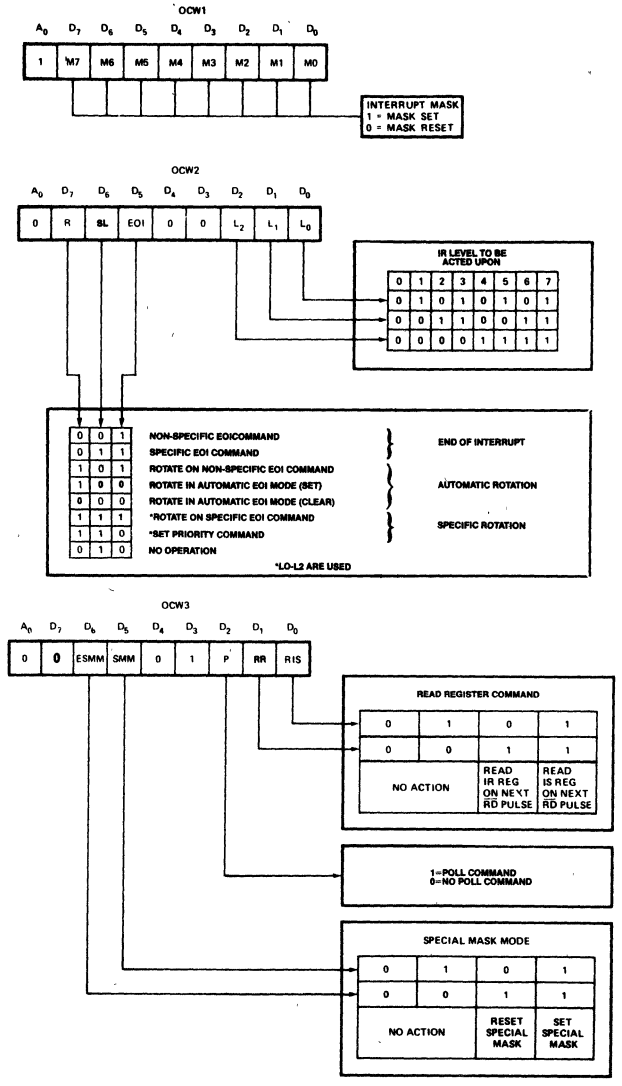


Figure 8. Operation Command Word Format

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, in the rotating priority mode.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence $\overline{\text{INTA}}$ pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

There are two forms of EOI command: Specific and Non-Specific. When the 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 8259A will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest IS level was necessarily the last level acknowledged and serviced. A non-specific EOI can be issued with OCW2 (EOI = 1, SL = 0, R = 0).

When a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset. A specific EOI can be issued with OCW2 (EOI = 1, SL = 1, R = 0, and LO-L2 is the binary level of the IS bit to be reset).

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 8259A is in the Special Mask Mode.

AUTOMATIC END OF INTERRUPT (AEOI) MODE

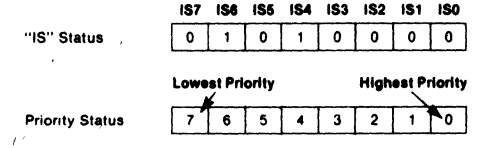
If AEOI = 1 in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85, second in iAPX 86). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 8259A.

The AEOI mode can only be used in a master 8259A and not a slave.

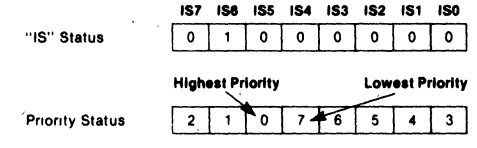
AUTOMATIC ROTATION (Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most *once*. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)



After Rotate (IR4 was serviced, all other priorities rotated correspondingly)



There are two ways to accomplish Automatic Rotation using OCW2, the Rotation on Non-Specific EOI Command (R = 1, SL = 0, EOI = 1) and the Rotate in Automatic EOI Mode which is set by (R = 1, SL = 0, EOI = 0) and cleared by (R = 0, SL = 0, EOI = 0).

SPECIFIC ROTATION (Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: R = 1, SL = 1; LO–L2 is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 (R = 1, SL = 1, EOI = 1 and LO–L2 = IR level to receive bottom priority).

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level *and enables* interrupts from *all other* levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SSMM = 1, SMM = 1, and cleared where SSMM = 1, SMM = 0.

POLL COMMAND

In this mode the INT output is not used or the microprocessor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by software using a Poll command.

The Poll command is issued by setting P = "1" in OCW3. The 8259A treats the next $\overline{RD}$ pulse to the 8259A (i.e., $\overline{RD} = 0$, $\overline{CS} = 0$) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from $\overline{WR}$ to $\overline{RD}$.

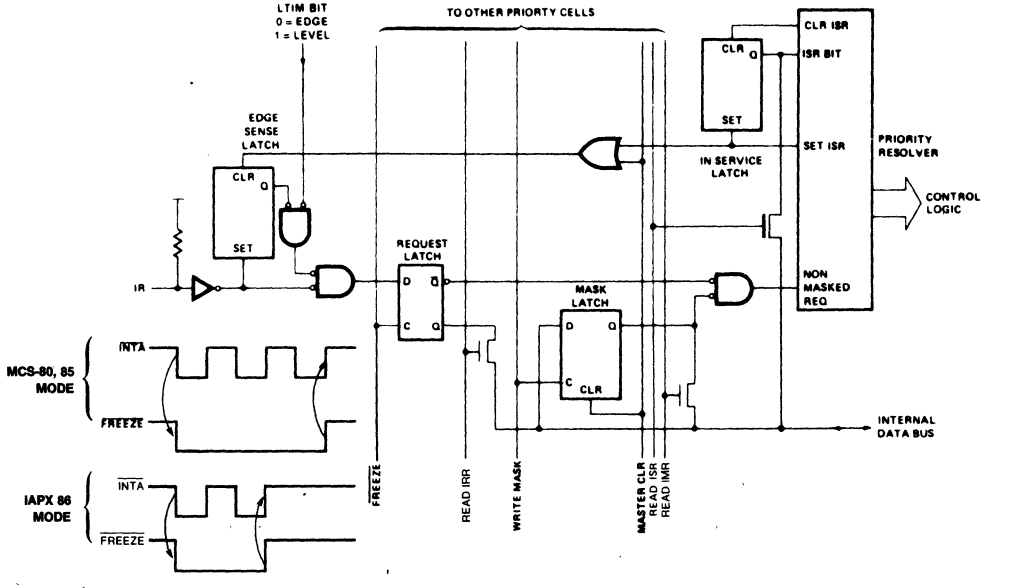
The word enabled onto the data bus during $\overline{RD}$ is:

D7	D6	D5	D4	D3	D2	D1	D0
I	–	–	–	–	W2	W1	W0

W0-W2: Binary code of the highest priority level requesting service.

I: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the $\overline{INTA}$ sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.



NOTES

1 MASTER CLEAR ACTIVE ONLY DURING ICW1

2 FREEZE/ IS ACTIVE DURING $\overline{INTA}$/ AND POLL SEQUENCES ONLY

3 TRUTH TABLE FOR D LATCH

C	D	Q	OPERATION
1	D_I	D_I	FOLLOW
0	X	Q_{n-1}	HOLD

Figure 9. Priority Cell—Simplified Logic Diagram

READING THE 8259A STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 [IMR]).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR.)

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt Command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0.)

The ISR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1).

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 8259A "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 8259A is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever $\overline{RD}$ is active and AO = 1 (OCW1).

Polling overrides status read when P = 1, RR = 1 in OCW3.

EDGE AND LEVEL TRIGGERED MODES

This mode is programmed using bit 3 in ICW1.

If LTIM = '0', an interrupt request will be recognized by a low to high transition on an IR input. The IR input can remain high without generating another interrupt.

If LTIM = '1', an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The priority cell diagram shows a conceptual circuit of the level sensitive and edge sensitive input circuitry of the 8259A. Be sure to note that the request latch is a transparent D type latch.

In both the edge and level triggered modes the IR inputs must remain high until after the falling edge of the first INTA. If the IR input goes low before this time a DEFAULT IR7 will occur when the CPU acknowledges the interrupt. This can be a useful safeguard for detecting interrupts caused by spurious noise glitches on the IR inputs. To implement this feature the IR7 routine is used for "clean up" simply executing a return instruction, thus ignoring the interrupt. If IR7 is needed for other purposes a default IR7 can still be detected by reading the ISR. A normal IR7 interrupt will set the corresponding ISR bit, a default IR7 won't. If a default IR7 routine occurs during a normal IR7 routine, however, the ISR will remain set. In this case it is necessary to keep track of whether or not the IR7 routine was previously entered. If another IR7 occurs it is a default.

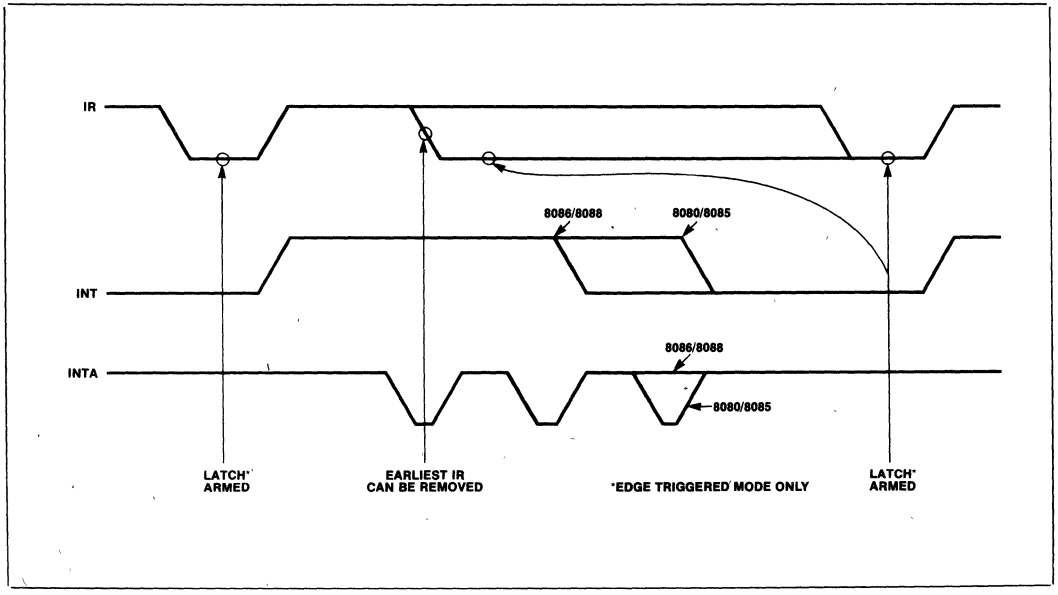


Figure 10. IR Triggering Timing Requirements

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal nested mode with the following exceptions:

a. When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)

b. When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on $\overline{SP}/\overline{EN}$ to enable the buffers. In this mode, whenever the 8259A's data bus outputs are enabled, the $\overline{SP}/\overline{EN}$ output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

CASCADE MODE

The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the $\overline{INTA}$ sequence.

In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address during bytes 2 and 3 of INTA. (Byte 2 only for 8086/8088).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first INTA pulse to the trailing edge of the third pulse. Each 8259A in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (CS) input of each 8259A.

The cascade lines of the Master 8259A are activated only for slave inputs, non slave inputs leave the cascade line inactive (low).

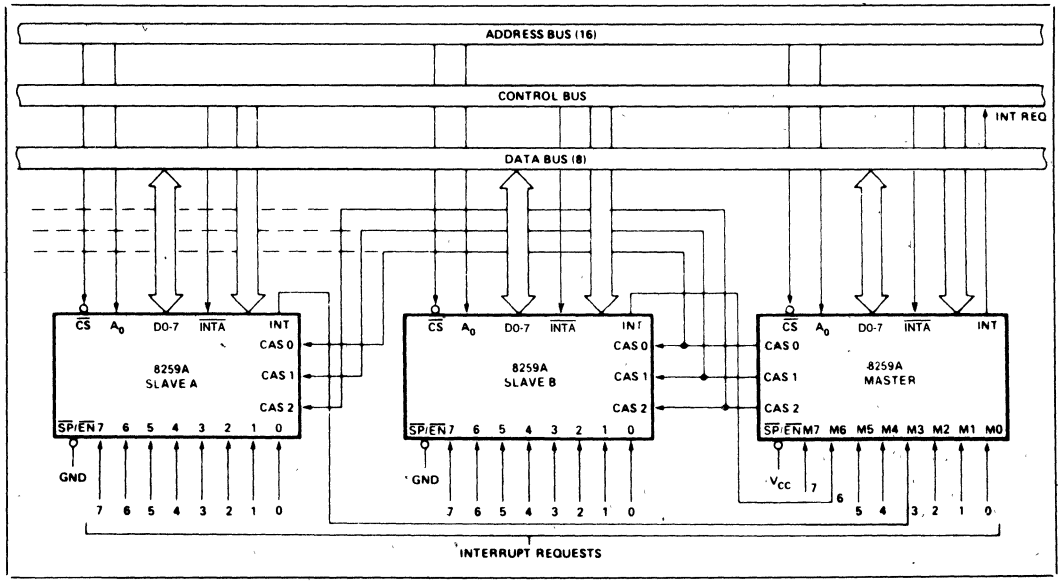


Figure 11. Cascading the 8259A

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin
with Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ±10% (8259A, 8259A-2)]

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0*	V_{CC} +0.5V	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
$V_{OH(INT)}$	Interrupt Output High Voltage	3.5		V	I_{OH} = −100μA
		2.4		V	I_{OH} = −400μA
I_{LI}	Input Load Current	−10	+10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LOL}	Output Leakage Current	−10	+10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		85	mA	
I_{LIR}	IR Input Load Current		−300	μA	V_{IN} = 0
			10	μA	V_{IN} = V_{CC}

*Note: For Extended Temperature EXPRESS V_{IH} = 2.3V.

CAPACITANCE (T_A = 25°C; V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1 MHZ
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to V_{SS}

A.C. CHARACTERISTICS [T_A = 0°C to 70°C, V_{CC} = 5V ±5% (8259A-8), V_{CC} = 5V ± 10% (8259A, 8259A-2)]

TIMING REQUIREMENTS

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TAHRL	AO/$\overline{CS}$ Setup to $\overline{RD}$/$\overline{INTA}$↓	50		0		0		ns	
TRHAX	AO/$\overline{CS}$ Hold after $\overline{RD}$/$\overline{INTA}$↑	5		0		0		ns	
TRLRH	$\overline{RD}$ Pulse Width	420		235		160		ns	
TAHWL	AO/$\overline{CS}$ Setup to $\overline{WR}$↓	50		0		0		ns	
TWHAX	AO/$\overline{CS}$ Hold after $\overline{WR}$↑	20		0		0		ns	
TWLWH	$\overline{WR}$ Pulse Width	400		290		190		ns	
TDVWH	Data Setup to $\overline{WR}$↑	300		240		160		ns	
TWHDX	Data Hold after $\overline{WR}$↑	40		0		0		ns	
TJLJH	Interrupt Request Width (Low)	100		100		100		ns	See Note 1
TCVIAL	Cascade Setup to Second or Third $\overline{INTA}$↓ (Slave Only)	55		55		40		ns	
TRHRL	End of $\overline{RD}$ to next $\overline{RD}$ End of $\overline{INTA}$ to next $\overline{INTA}$ within an $\overline{INTA}$ sequence only	160		160		160		ns	
TWHWL	End of $\overline{WR}$ to next $\overline{WR}$	190		190		190		ns	

A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
*TCHCL	End of Command to next Command (Not same command type)	500		500		500		ns	
	End of $\overline{INTA}$ sequence to next $\overline{INTA}$ sequence.								

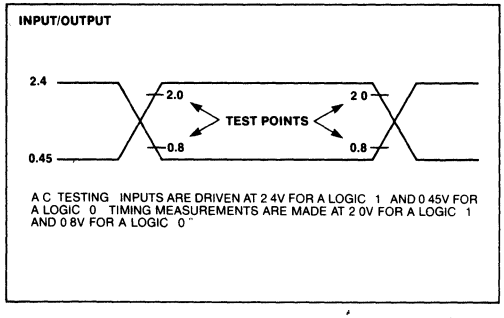
*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 500 ns (i.e. 8085A = 1.6μs, 8085A-2 = 1μs, 8086 = 1μs, 8086-2 = 625 ns)

NOTE: This is the low time required to clear the input latch in the edge triggered mode.

TIMING RESPONSES

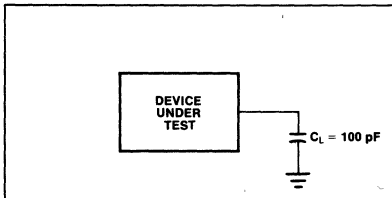
Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TRLDV	Data Valid from $\overline{RD}$/$\overline{INTA}$↓		300		200		120	ns	C of Data Bus = 100 pF
TRHDZ	Data Float after $\overline{RD}$/$\overline{INTA}$↑	10	200	10	100	10	85	ns	C of Data Bus Max text C = 100 pF Min. test C = 15 pF
TJHIH	Interrupt Output Delay		400		350		300	ns	
TIALCV	Cascade Valid from First $\overline{INTA}$↓ (Master Only)		565		565		360	ns	C_{INT} = 100 pF
TRLEL	Enable Active from $\overline{RD}$↓ or $\overline{INTA}$↓		160		125		100	ns	$C_{CASCADE}$ = 100 pF
TRHEH	Enable Inactive from $\overline{RD}$↑ or $\overline{INTA}$↑		325		150		150	ns	
TAHDV	Data Valid from Stable Address		350		200		200	ns	
TCVDV	Cascade Valid to Valid Data		300		300		200	ns	

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

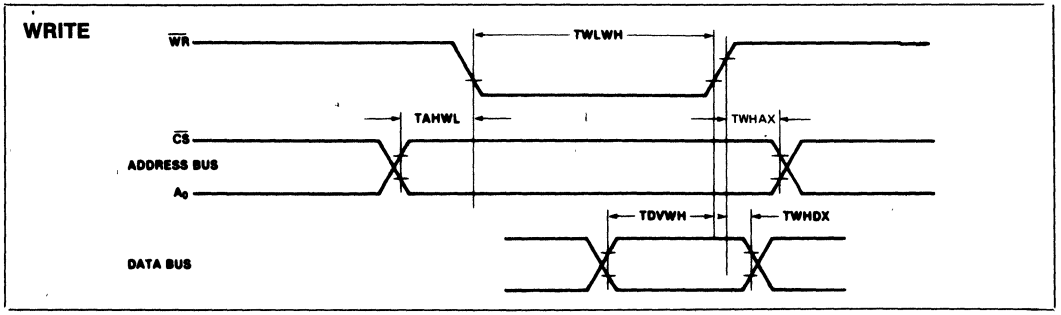
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

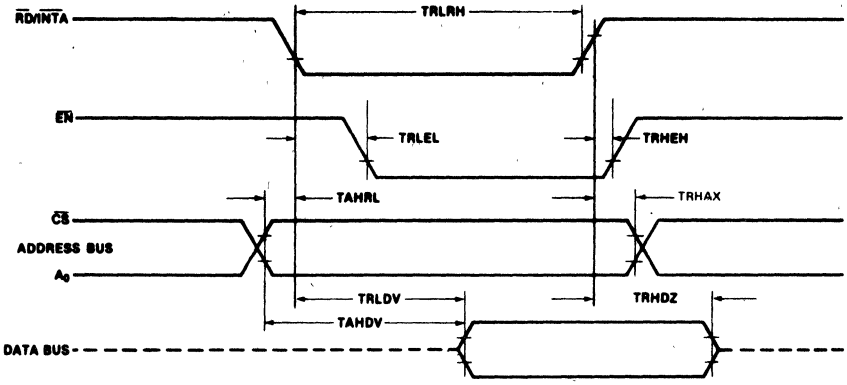
WAVEFORMS

WRITE

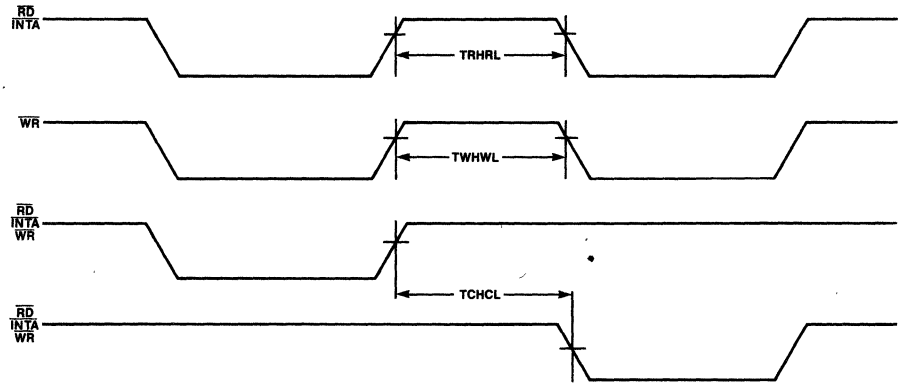


WAVEFORMS (Continued)

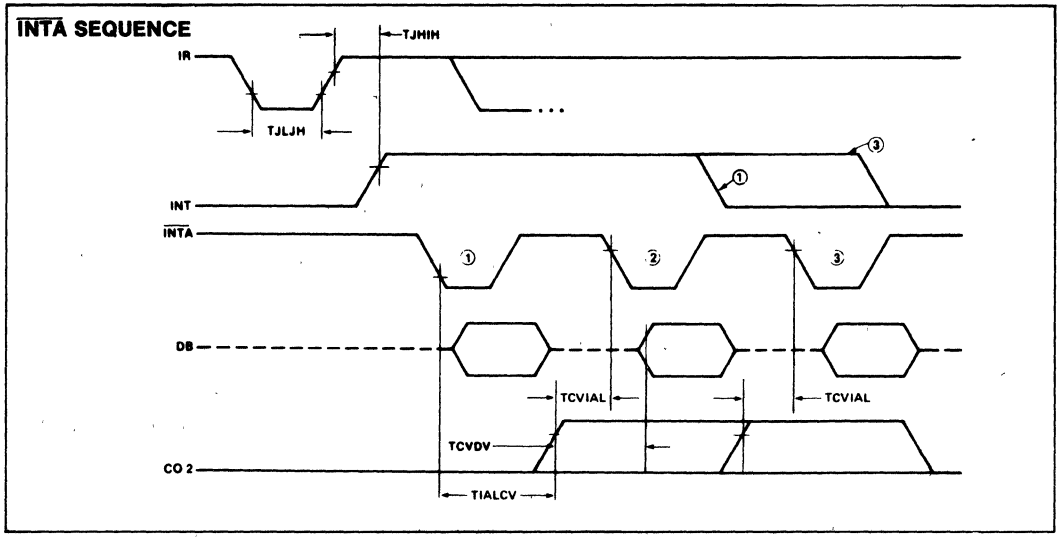
READ/INTA



OTHER TIMING



WAVEFORMS (Continued)



NOTES: Interrupt output must remain HIGH at least until leading edge of first INTA.
1. Cycle 1 in iAPX 86, iAPX 88 systems, the Data Bus is not active.

8355/8355-2
16,384-BIT ROM WITH I/O

- **2048 Words x 8 Bits**
- **Single +5V Power Supply**
- **Directly Compatible with 8085A and iAPX 88 Microprocessors**
- **2 General Purpose 8-Bit I/O Ports**
- **Each I/O Port Line Individually Programmable as Input or Output**
- **Multiplexed Address and Data Bus**
- **Internal Address Latch**
- **40-Pin DIP**

The Intel® 8355 is a ROM and I/O chip to be used in the 8085A and iAPX 88 microprocessor systems. The ROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in the 8085A CPU.

The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 lines and each I/O port line is individually programmable as input or output.

The 8355-2 has a 300 ns access time for compatibility with the 8085A-2 and 5 MHz iAPX 88 microprocessors.

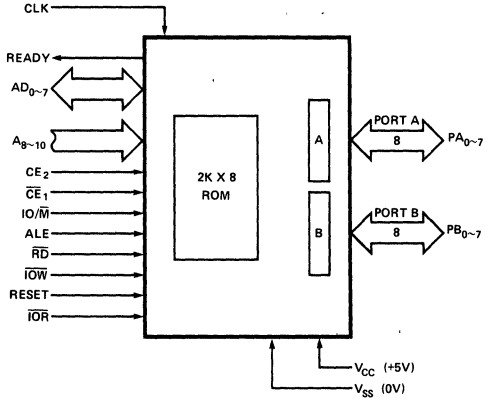
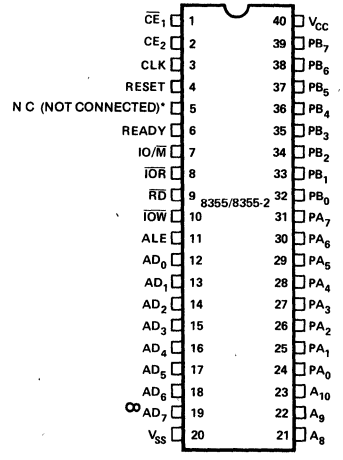


Figure 1. Block Diagram



*For 8755A compatibility, pin 5 should be directly tied to VCC.

Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
ALE	I	**Address Latch Enable:** When high, AD_{0-7}, IO/$\overline{M}$, A_{8-10}, CE_2, and $\overline{CE_1}$ enter the address latches. The signals (AD, I/O$\overline{M}$, A_{8-10}, CE_2, $\overline{CE_1}$) are latched in at the trailing edge of ALE.
AD_{0-7}	I	**Address/Data Bus (Bidirectional):** The lower 8-bits of the ROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B is selected based on the latched value of AD_0. If $\overline{RD}$ or $\overline{IOR}$ is low when the latched chip enables are active, the output buffers present data on the bus.
A_{8-10}	I	**Address Bus:** High order bits of the ROM address. They do not affect I/O operations.
$\overline{CE_1}$ CE_2	I	**Chip Enable Inputs:** $\overline{CE_1}$ is active low and CE_2 is active *high*. The 8355 can be accessed only when *BOTH* Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD_{0-7} and READY outputs will be in a high impedance state.
IO/$\overline{M}$	I	**I/O Memory:** If the latched IO/$\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low, the output data comes from the ROM.
$\overline{RD}$	I	**Read:** If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD_{0-7} output buffers are enabled and output either the selected ROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD_{0-7} output buffers are 3-stated.
$\overline{IOW}$	I	**I/O Write:** If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD_0 to be written with the data on AD_{0-7}. The state of IO/$\overline{M}$ is ignored.
CLK	I	**Clock:** Used to force the READY into its high impedance state after it has been forced low by $\overline{CE_1}$ low, CE_2 high and ALE high.
READY	O	**READY:** A 3-state output controlled by $\overline{CE_1}$, CE_2, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK.
PA_{0-7}	I/O	**Port A:** General purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD_0, AD_1. Read operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD_0 and AD_1 low, *or* IO/$\overline{M}$ high, $\overline{RD}$ low, active chip enables, and AD_0 and AD_1, LOW.
PB_{0-7}	I/O	**Port B:** This general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD_0 and a 0 from AD_1.
RESET	I	**Reset:** An input high causes all pins in Port A and B to assume input mode. (Clear DER Register).
$\overline{IOR}$	I	**I/O Read:** When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination IO/$\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V_{CC} ("1").
V_{CC}		**Voltage:** +5 volt supply.
V_{SS}		**Ground:** Ground Reference.

FUNCTIONAL DESCRIPTION

ROM Section

The 8355 contains an 8-bit address latch which allows it to interface directly to MCS-48, MCS-85, and iAPX 88/10 Microcomputers without additional hardware.

The ROM section of the chip is addressed by an 11-bit address and the Chip Enables. The address and levels on the Chip Enable pins are latched into the address latches on the falling edge of ALE. If the latched Chip Enables are active and IO/$\overline{M}$ is low when $\overline{RD}$ goes low, the contents of the ROM location addressed by the latched address are put out through AD_{0-7} output buffers.

I/O Section

The I/O section of the chip is addressed by the latched value of AD_{0-1}. Two 8-bit Data Direction Registers (DDR) in 8355 determine the input/output status of each pin in the corresponding ports. A "0" in a particular bit position of a DDR signifies that the corresponding I/O port bit is in the input mode. A "1" in a particular bit position signifies that the corresponding I/O port bit is in the output mode. In this manner the I/O ports of the 8355 are bit-by-bit programmable as inputs or outputs. The table summarizes port and DDR designation. DDR's cannot be read.

AD_1	AD_0	Selection
0	0	Port A
0	1	Port B
1	0	Port A Data Direction Register (DDR A)
1	1	Port B Data Direction Register (DDR B)

When $\overline{IOW}$ goes low and the Chip Enables are active, the data on the AD_{0-7} is written into I/O port selected by the latched value of AD_{0-1}. During this operation all I/O bits of the selected port are affected, regardless of their I/O mode and the state of IO/$\overline{M}$. The actual output level does not change until $\overline{IOW}$ returns high (glitch free output).

A port can be read out when the latched Chip Enables are active and either $\overline{RD}$ goes low with IO/$\overline{M}$ high, or $\overline{IOR}$ goes low. Both input and output mode bits of a selected port will appear on lines AD_{0-7}.

To clarify the function of the I/O ports and Data Direction Registers, the following diagram shows the configuration of one bit of PORT A and DDR A. The same logic applies to PORT B and DDR B

Note that hardware RESET or writing a zero to the DDR latch will cause the output latch's output buffer to be disabled, preventing the data in the output latch from being passed through to the pin. This is equivalent to putting the port in the input mode. Note also that the data can be written to the Output Latch even though the Output Buffer has been disabled. This enables a port to be initialized with a value prior to enabling the output.

The diagram also shows that the contents of PORT A and PORT B can be read even when the ports are configured as outputs.

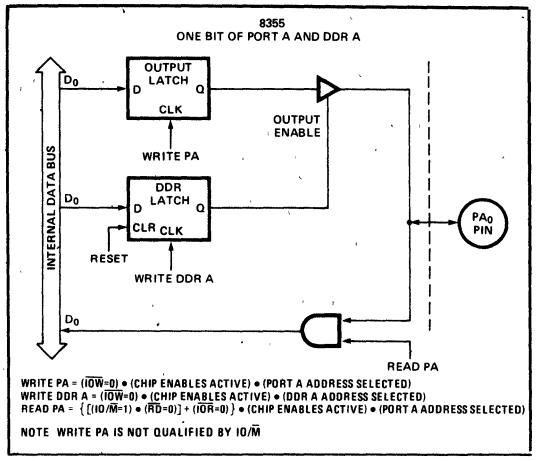


Figure 3. 8355 One Bit of Port A and DDR A

SYSTEM APPLICATIONS

System Interface with 8085A and iAPX 88

A system using the 8355 can use either one of the two I/O Interface techniques.

- Standard I/O
- Memory Mapped I/O

If a standard I/O technique is used, the system can use the feature of both CE_2 and $\overline{CE}_1$. By using a combination of unused address lines A_{11-15} and the Chip Enable inputs, the system can use up to 5 each 8355's without requiring a CE decoder. See Figure 5a and 5b.

If a memory mapped I/O approach is used the 8355 will be selected by the combination of both the Chip Enables and IO/$\overline{M}$ using AD_{8-15} address lines. See Figure 4.

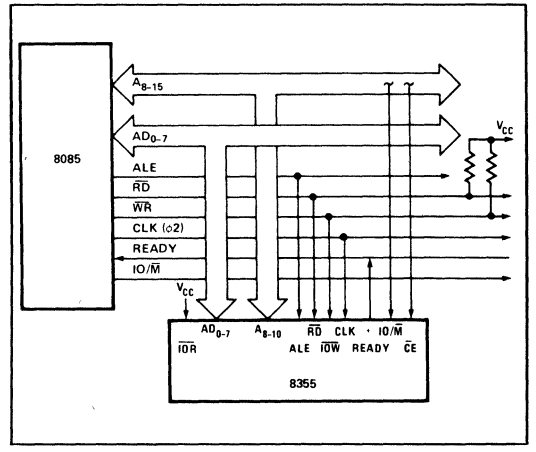


Figure 4. 8355 in 8085A System (Memory-Mapped I/O)

iAPX 88 FIVE CHIP SYSTEM:

- 1.25 K Bytes RAM
- 2 K Bytes ROM
- 38 I/O Pins
- 1 Internal Timer
- 2 Interrupt Levels

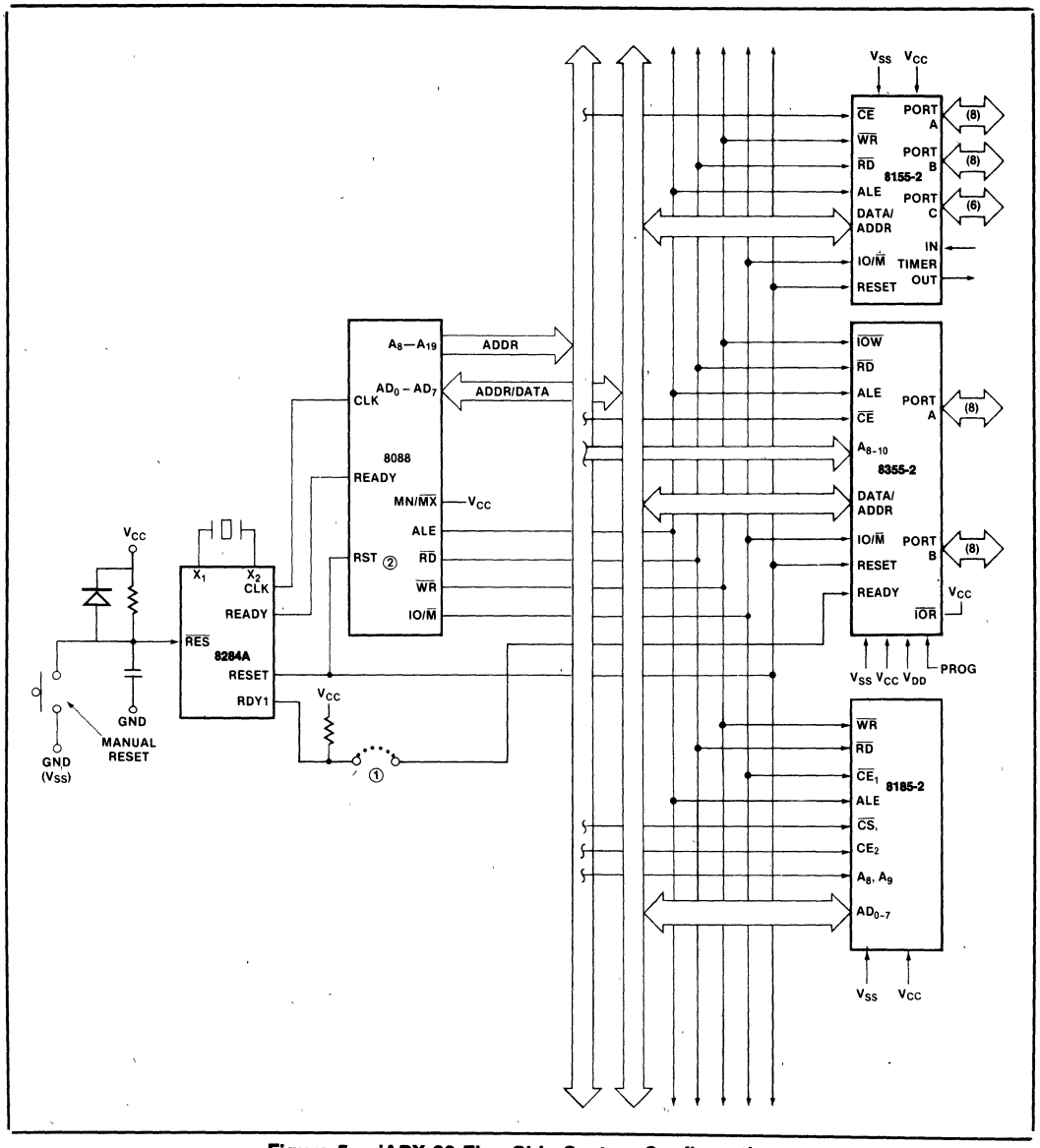
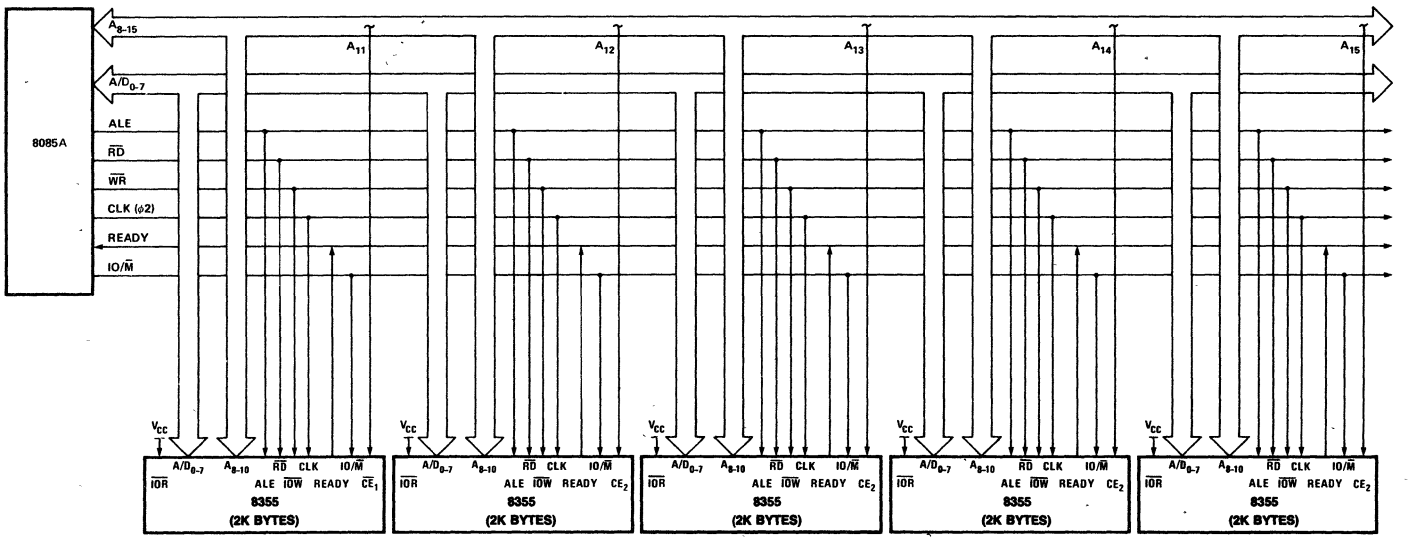


Figure 5a. iAPX 88 Five Chip System Configuration



NOTE: Use $\overline{CE_1}$ for the first 8355 in the system, and CE_2 for the other 8355's. Permits up to 5-8355's in a system without CE decoder.

Figure 5b. 8355 in 8085A System (Standard I/O)

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to +70°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin
With Respect to Ground ..... -0.5V to +7V
Power Dissipation 1.5W

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	V_{CC} = 5.0V
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leqslant V_{OUT} \leqslant V_{CC}$
I_{CC}	V_{CC} Supply Current		180	mA	

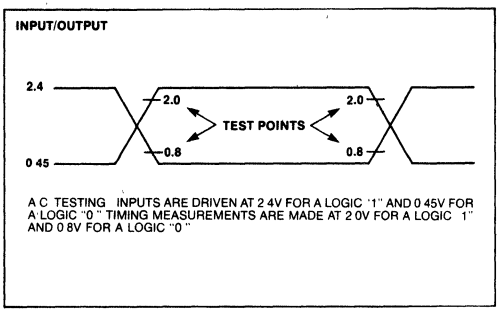
A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%)

Symbol	Parameter	8355		8355-2		Units
		Min.	Max.	Min.	Max.	
t_{CYC}	Clock Cycle Time	320		200		ns
T_1	CLK Pulse Width	80		40		ns
T_2	CLK Pulse Width	120		70		ns
t_f, t_r	CLK Rise and Fall Time		30		30	ns
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		45		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control*		170		140	ns
t_{AD}	Address Stable to Data Out Valid**		450		300	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float after READ	0	100	0	85	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to Write Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	30		10		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Set Up Time	50		50		ns
t_{RP}	Port Input Hold Time	50		50		ns
t_{RYH}	READY HOLD Time	0	160	0	160	ns
t_{ARY}	ADDRESS (CE) to READY		160		160	ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{RDE}	READ Control to Data Bus Enable	10		10		ns

*Or T_{AD}-(T_{AL} + T_{LC}), whichever is greater.

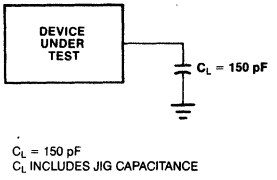
**Defines ALE to Data out Valid in conjunction with T_{AL}.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC '1' AND 0 45V FOR A LOGIC "0 " TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0 "

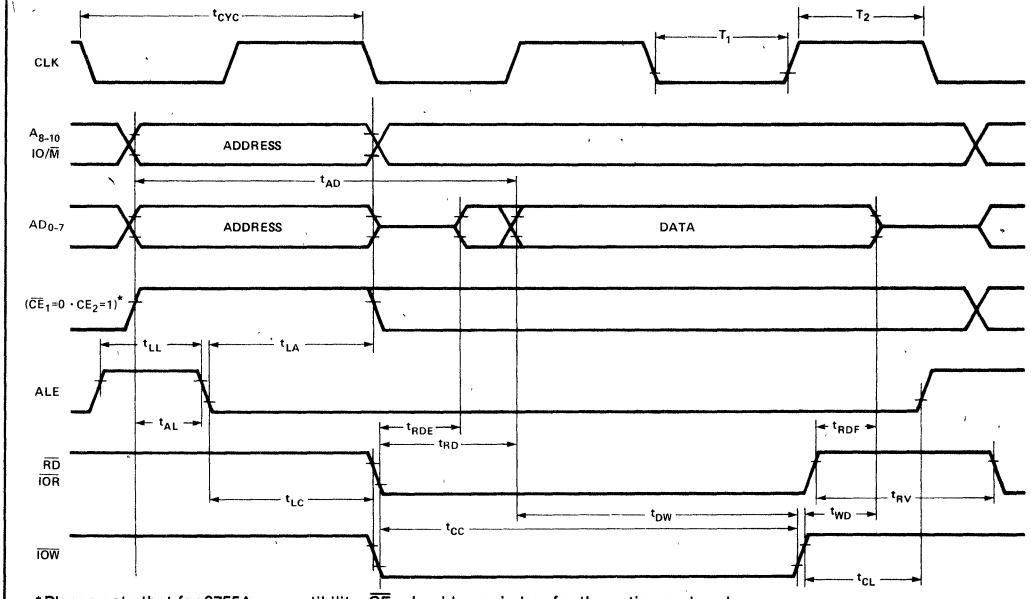
A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

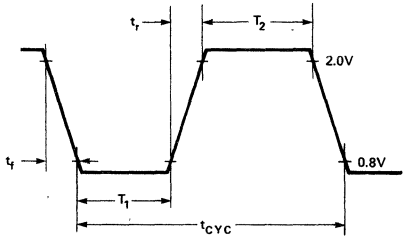
WAVEFORMS

ROM READ AND I/O READ AND WRITE



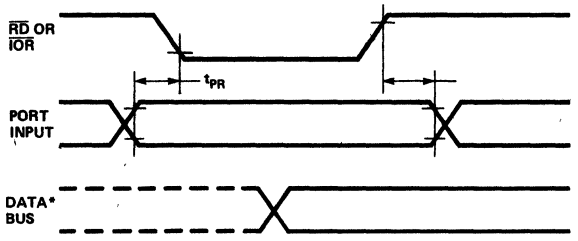
*Please note that for 8755A compatibility, $\overline{CE}_1$ should remain low for the entire read cycle.

8355 CLOCK SPECIFICATIONS

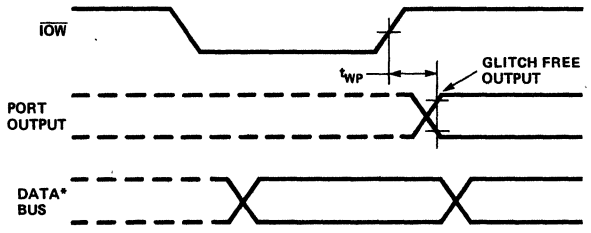


WAVEFORMS (Continued)

INPUT MODE

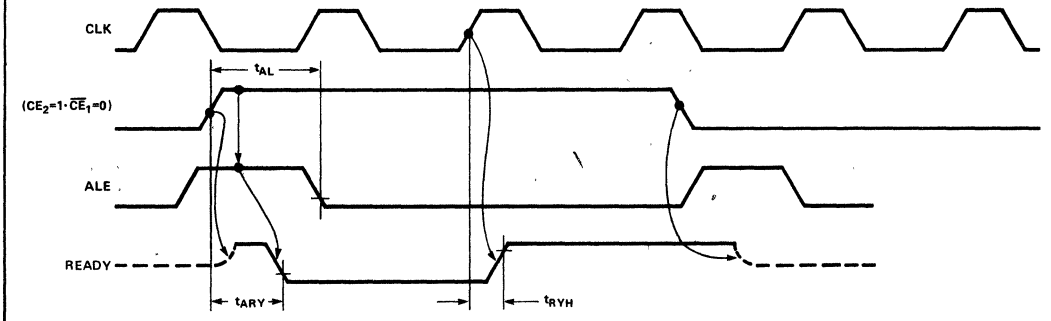


OUTPUT MODE



*DATA BUS TIMING IS SHOWN IN FIGURE 4.

WAIT STATE



NOTE: Ready = 0.

8755A/8755A-2
16,384-BIT EPROM WITH I/O

- 2048 Words x 8 Bits
- Single +5V Power Supply (V_{CC})
- Directly Compatible with 8085A and 8088 Microprocessors
- U.V. Erasable and Electrically Reprogrammable
- Internal Address Latch
- 2 General Purpose 8-Bit I/O Ports
- Each I/O Port Line Individually Programmable as Input or Output
- Multiplexed Address and Data Bus
- 40-Pin DIP
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® 8755A is an erasable and electrically reprogrammable ROM (EPROM) and I/O chip to be used in the 8085A and iAPX 88 microprocessor systems. The EPROM portion is organized as 2048 words by 8 bits. It has a maximum access time of 450 ns to permit use with no wait states in an 8085A CPU.

The I/O portion consists of 2 general purpose I/O ports. Each I/O port has 8 port lines, and each I/O port line is individually programmable as input or output.

The 8755A-2 is a high speed selected version of the 8755A compatible with the 5 MHz 8085A-2 and the 5 MHz iAPX 88 microprocessor.

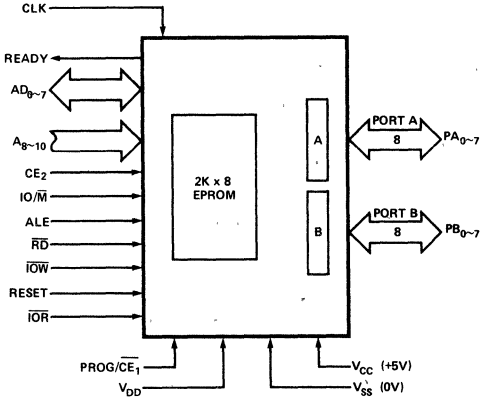


Figure 1. Block Diagram

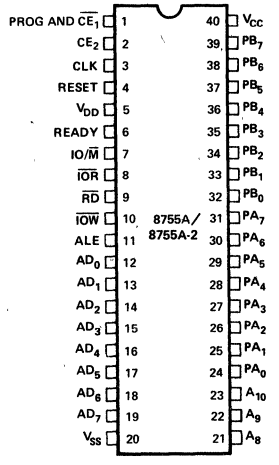


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
ALE	I	**Address Latch Enable:** When Address Latch Enable goes *high*, AD_{0-7}, IO/$\overline{M}$, A_{8-10}, CE_2, and $\overline{CE_1}$ enter the address latches. The signals (AD, IO/$\overline{M}$ AD_{8-10}, CE_2, $\overline{CE_1}$) are latched in at the trailing edge of ALE.
AD_{0-7}	I	**Bidirectional Address/Data Bus:** The lower 8-bits of the PROM or I/O address are applied to the bus lines when ALE is high. During an I/O cycle, Port A or B is selected based on the latched value of AD_0. IF $\overline{RD}$ or $\overline{IOR}$ is low when the latched Chip Enables are active, the output buffers present data on the bus.
A_{8-10}	I	**Address Bus:** These are the high order bits of the PROM address. They do not affect I/O operations.
PROG/$\overline{CE_1}$ CE_2	I	**Chip Enable Inputs:** $\overline{CE_1}$ is active low and CE_2 is active high. The 8755A can be accessed only when *both* Chip Enables are active at the time the ALE signal latches them up. If either Chip Enable input is not active, the AD_{0-7} and READY outputs will be in a high impedance state. $\overline{CE_1}$ is also used as a programming pin. (See section on programming.)
IO/$\overline{M}$	I	**I/O Memory:** If the latched IO/$\overline{M}$ is high when $\overline{RD}$ is low, the output data comes from an I/O port. If it is low the output data comes from the PROM.
$\overline{RD}$	I	**Read:** If the latched Chip Enables are active when $\overline{RD}$ goes low, the AD_{0-7} output buffers are enabled and output either the selected PROM location or I/O port. When both $\overline{RD}$ and $\overline{IOR}$ are high, the AD_{0-7} output buffers are 3-stated.
$\overline{IOW}$	I	**I/O Write:** If the latched Chip Enables are active, a low on $\overline{IOW}$ causes the output port pointed to by the latched value of AD_0 to be written with the data on AD_{0-7}. The state of IO/$\overline{M}$ is ignored.
CLK	I	**Clock:** The CLK is used to force the READY into its high impedance state after it has been forced low by $\overline{CE_1}$ low, CE_2 high, and ALE high.

Symbol	Type	Name and Function
READY	O	**Ready** is a 3-state output controlled by $\overline{CE_1}$, CE_2, ALE and CLK. READY is forced low when the Chip Enables are active during the time ALE is high, and remains low until the rising edge of the next CLK. (See Figure 6c.)
PA_{0-7}	I/O	**Port A:** These are general purpose I/O pins. Their input/output direction is determined by the contents of Data Direction Register (DDR). Port A is selected for write operations when the Chip Enables are active and $\overline{IOW}$ is low and a 0 was previously latched from AD_0, AD_1. Read Operation is selected by either $\overline{IOR}$ low and active Chip Enables and AD_0 and AD_1 low, *or* IO/$\overline{M}$ high, $\overline{RD}$ low, active Chip Enables, and AD_0 and AD_1 low.
PB_{0-7}	I/O	**Port B:** This general purpose I/O port is identical to Port A except that it is selected by a 1 latched from AD_0 and a 0 from AD_1.
RESET	I	**Reset:** In normal operation, an input high on RESET causes all pins in Ports A and B to assume input mode (clear DDR register).
$\overline{IOR}$	I	**I/O Read:** When the Chip Enables are active, a low on $\overline{IOR}$ will output the selected I/O port onto the AD bus. $\overline{IOR}$ low performs the same function as the combination of IO/$\overline{M}$ high and $\overline{RD}$ low. When $\overline{IOR}$ is not used in a system, $\overline{IOR}$ should be tied to V_{CC} ("1").
V_{CC}		**Power:** +5 volt supply.
V_{SS}		**Ground:** Reference.
V_{DD}		**Power Supply:** V_{DD} is a programming voltage, and must be tied to V_{CC} when the 8755A is being read. For programming, a high voltage is supplied with V_{DD} = 25V, typical. (See section on programming.)

FUNCTIONAL DESCRIPTION

PROM Section

The 8755A contains an 8-bit address latch which allows it to interface directly to MCS-48, MCS-85 and iAPX 88/10 Microcomputers without additional hardware.

The PROM section of the chip is addressed by the 11-bit address and the Chip Enables. The address, $\overline{CE}_1$ and CE_2 are latched into the address latches on the falling edge of ALE. If the latched Chip Enables are active and IO/$\overline{M}$ is low when $\overline{RD}$ goes low, the contents of the PROM location addressed by the latched address are put out on the AD_{0-7} lines (provided that V_{DD} is tied to V_{CC}.)

I/O Section

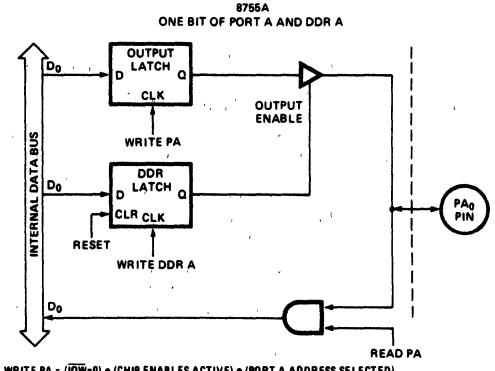
The I/O section of the chip is addressed by the latched value of AD_{0-1}. Two 8-bit Data Direction Registers (DDR) in 8755A determine the input/output status of each pin in the corresponding ports. A "0" in a particular bit position of a DDR signifies that the corresponding I/O port bit is in the input mode. A "1" in a particular bit position signifies that the corresponding I/O port bit is in the output mode. In this manner the I/O ports of the 8755A are bit-by-bit programmable as inputs or outputs. The table summarizes port and DDR designation. DDR's cannot be read.

AD_1	AD_0	Selection
0	0	Port A
0	1	Port B
1	0	Port A Data Direction Register (DDR A)
1	1	Port B Data Direction Register (DDR B)

When $\overline{IOW}$ goes low and the Chip Enables are active, the data on the AD_{0-7} is written into I/O port selected by the latched value of AD_{0-1}. During this operation all I/O bits of the selected port are affected, regardless of their I/O mode and the state of IO/$\overline{M}$. The actual output level does not change until $\overline{IOW}$ returns high. (glitch free output)

A port can be read out when the latched Chip Enables are active and either $\overline{RD}$ goes low with IO/$\overline{M}$ high, or $\overline{IOR}$ goes low. Both input and output mode bits of a selected port will appear on lines AD_{0-7}.

To clarify the function of the I/O Ports and Data Direction Registers, the following diagram shows the configuration of one bit of PORT A and DDR A. The same logic applies to PORT B and DDR B.



WRITE PA = ($\overline{IOW}$=0) • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)
WRITE DDR A = ($\overline{IOW}$=0) • (CHIP ENABLES ACTIVE) • (DDR A ADDRESS SELECTED)
READ PA = {[(IO/$\overline{M}$=1) • ($\overline{RD}$=0)] + ($\overline{IOR}$=0)} • (CHIP ENABLES ACTIVE) • (PORT A ADDRESS SELECTED)

NOTE: WRITE PA IS NOT QUALIFIED BY IO/$\overline{M}$.

Note that hardware RESET or writing a zero to the DDR latch will cause the output latch's output buffer to be disabled, preventing the data in the Output Latch from being passed through to the pin. This is equivalent to putting the port in the input mode. Note also that the data can be written to the Output Latch even though the Output Buffer has been disabled. This enables a port to be initialized with a value prior to enabling the output.

The diagram also shows that the contents of PORT A and PORT B can be read even when the ports are configured as outputs.

TABLE 1. 8755A PROGRAMMING MODULE CROSS REFERENCE

MODULE NAME	USE WITH
UPP 955	UPP(4)
UPP UP2(2)	UPP 855
PROMPT 975	PROMPT 80/85(3)
PROMPT 475	PROMPT 48(1)

NOTES:
1. Described on p. 13-34 of 1978 Data Catalog.
2. Special adaptor socket.
3. Described on p. 13-39 of 1978 Data Catalog.
4. Described on p. 13-71 of 1978 Data Catalog.

ERASURE CHARACTERISTICS

The erasure characteristics of the 8755A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8755A in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8755A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 8755 window to prevent unintentional erasure.

The recommended erasure procedure for the 8755A is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm^2 power rating. The 8755A should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter on their tubes and this filter should be removed before erasure.

PROGRAMMING

Initially, and after each erasure, all bits of the EPROM portions of the 8755A are in the "1" state. Information is introduced by selectively programming "0" into the desired bit locations. A programmed "0" can only be changed to a "1" by UV erasure.

The 8755A can be programmed on the Intel® Universal PROM Programmer (UPP), and the PROMPT™ 80/85 and PROMPT-48™ design aids. The appropriate programming modules and adapters for use in programming both 8755A's and 8755's are shown in Table 1.

The program mode itself consists of programming a single address at a time, giving a single 50 msec pulse for every address. Generally, it is desirable to have a verify cycle after a program cycle for the same address as shown in the attached timing diagram. In the verify cycle (i.e., normal memory read cycle) 'V_{DD}' should be at +5V.

Preliminary timing diagrams and parameter values pertaining to the 8755A programming operation are contained in Figure 7.

SYSTEM APPLICATIONS

System Interface with 8085A and iAPX 88

A system using the 8755A can use either one of the two I/O Interface techniques:

- Standard I/O
- Memory Mapped I/O

If a standard I/O technique is used, the system can use the feature of both CE_2 and $\overline{CE_1}$. By using a combination of unused address lines A_{11-15} and the Chip Enable inputs, the 8085A system can use up to 5 each 8755A's without requiring a CE decoder. See Figure 4a and 4b.

If a memory mapped I/O approach is used the 8755A will be selected by the combination of both the Chip Enables and IO/$\overline{M}$ using AD_{8-15} address lines. See Figure 3.

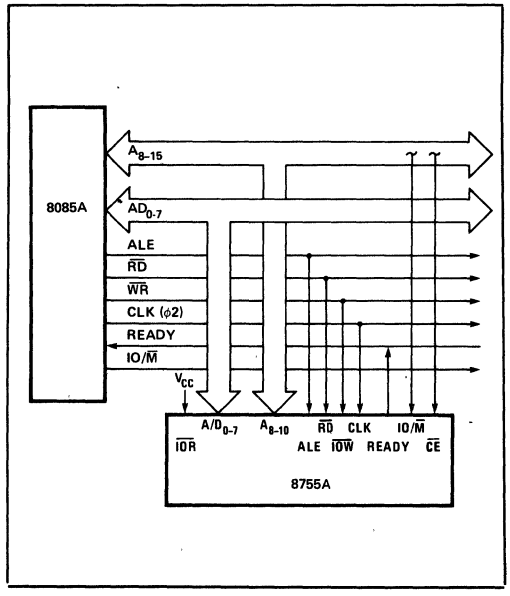


Figure 3. 8755A in 8085A System (Memory-Mapped I/O)

iAPX 88 FIVE CHIP SYSTEM

Figure 4 shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes ROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

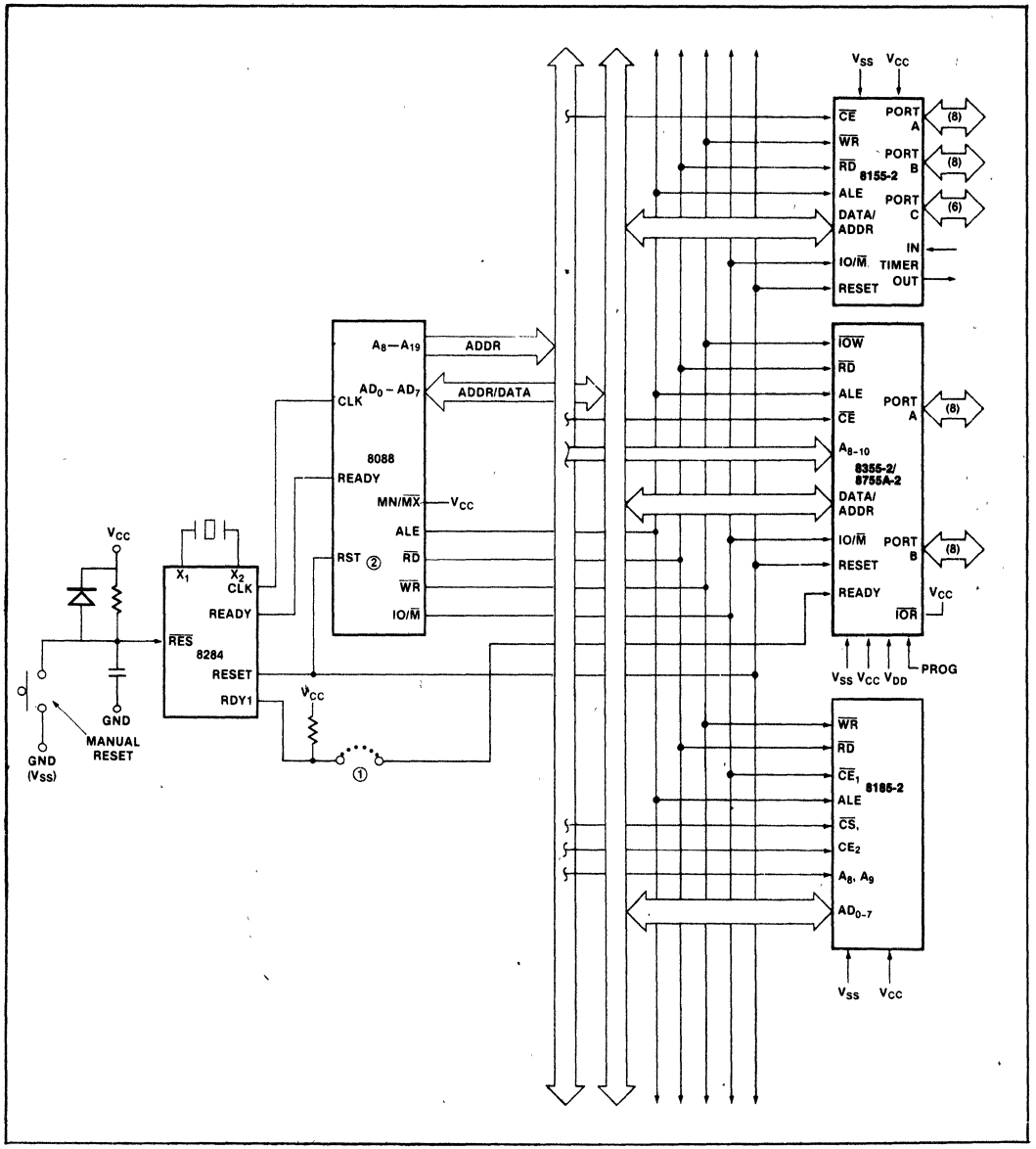


Figure 4a. iAPX 88 Five Chip System Configuration

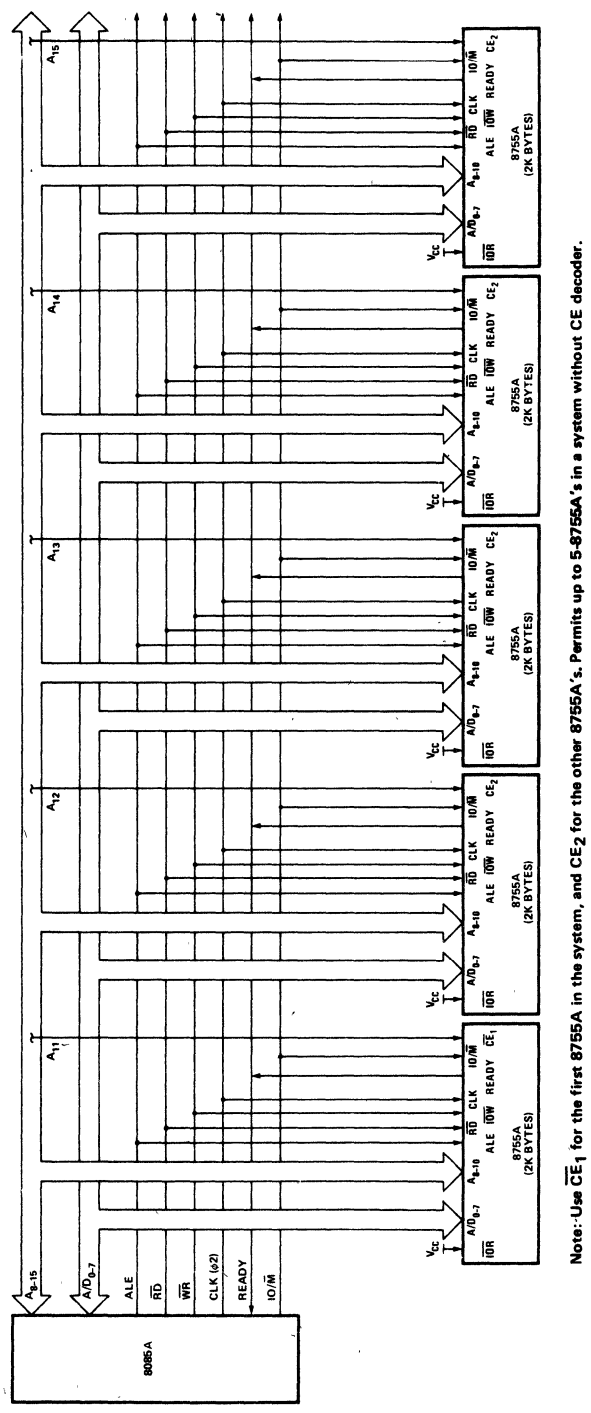


Figure 4b. 8755A in 8085A System (Standard I/O)

AFN-00843D

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias	0°C to +70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin With Respect to Ground	-0.5V to +7V
Power Dissipation	1.5W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(T_A = 0°C to 70°, V_{CC} = V_{DD} = 5V ± 5%; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

SYMBOL	PARAMETER	MIN.	MAX.	UNITS	TEST CONDITIONS
V_{IL}	Input Low Voltage	-0.5	0.8	V	V_{CC} = 5.0V
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	V_{CC} = 5.0V
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current		180	mA	
I_{DD}	V_{DD} Supply Current		30	mA	V_{DD} = V_{CC}
C_{IN}	Capacitance of Input Buffer		10	pF	f_C = 1μHz
$C_{I/O}$	Capacitance of I/O Buffer		15	pF	f_C = 1μHz

D.C. CHARACTERISTICS — PROGRAMMING

(T_A = 0°C to 70°, V_{CC} = 5V ± 5%, V_{SS} = 0V, V_{DD} = 25V ±1V; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{DD}	Programming Voltage (during Write to EPROM)	24	25	26	V
I_{DD}	Prog Supply Current		15	30	mA

A.C. CHARACTERISTICS (T_A = 0°C to 70°, V_{CC} = 5V ± 5%; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

Symbol	Parameter	8755A		8755A-2 (Preliminary)		Units
		Min.	Max.	Min.	Max.	
t_{CYC}	Clock Cycle Time	320		200		ns
T_1	CLK Pulse Width	80		40		ns
T_2	CLK Pulse Width	120		70		ns
t_f,t_r	CLK Rise and Fall Time		30		30	ns
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		45		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control*		170		140	ns
t_{AD}	Address Stable to Data Out Valid**		450		300	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float after READ	0	100	0	85	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to Write Set Up Time	150		150		ns
t_{WD}	Data In Hold Time After WRITE	30		10		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Set Up Time	50		50		ns
t_{RP}	Port Input Hold Time to Control	50		50		ns
t_{RYH}	READY HOLD Time to Control	0	160	0	160	ns
t_{ARY}	ADDRESS (CE) to READY		160		160	ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{RDE}	READ Control to Data Bus Enable	10		10		ns

NOTE:

C_{LOAD} = 150pF.

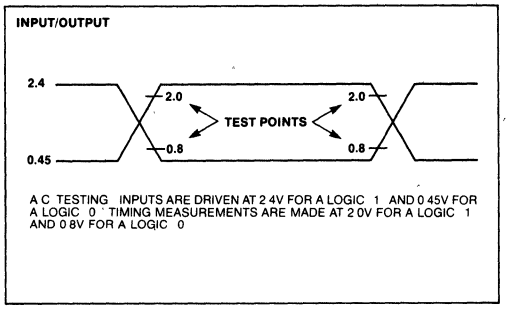
*Or $T_{AD} - (T_{AL} + T_{LC})$, whichever is greater.

**Defines ALE to Data Out Valid in conjunction with T_{AL}.

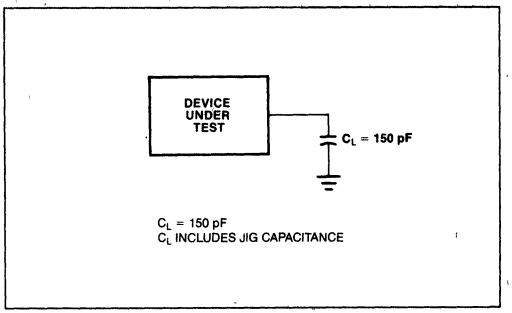
A.C. CHARACTERISTICS—PROGRAMMING (T_A = 0°C to 70°, V_{CC} = 5V ± 5%, V_{SS} = 0V, V_{DD} = 25V ±1V; V_{CC} = V_{DD} = 5V ±10% for 8755A-2)

Symbol	Parameter	Min.	Typ.	Max.	Unit
t_{PS}	Data Setup Time	10			ns
t_{PD}	Data Hold Time	0			ns
t_S	Prog Pulse Setup Time	2			μs
t_H	Prog Pulse Hold Time	2			μs
t_{PR}	Prog Pulse Rise Time	0.01	2		μs
t_{PF}	Prog Pulse Fall Time	0.01	2		μs
t_{PRG}	Prog Pulse Width	45	50		msec

A.C. TESTING INPUT, OUTPUT WAVEFORM

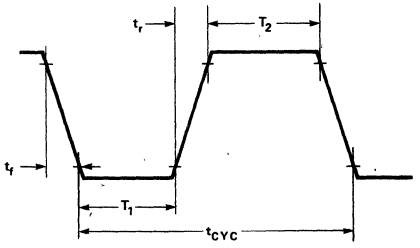


A.C. TESTING LOAD CIRCUIT

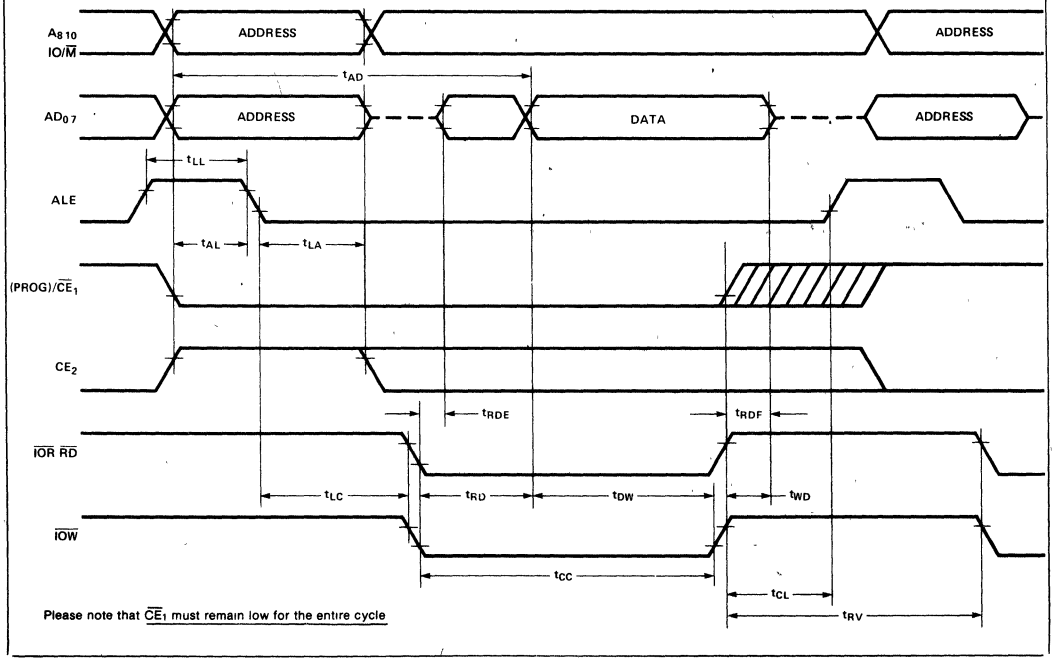


WAVEFORMS

CLOCK SPECIFICATION FOR 8755A



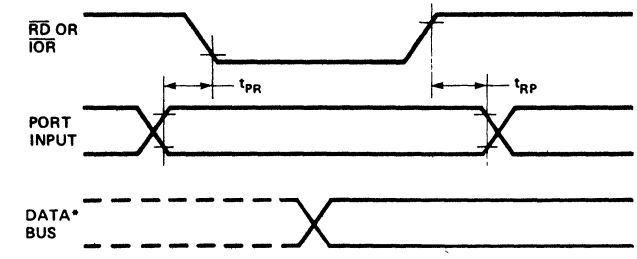
PROM READ, I/O READ AND WRITE



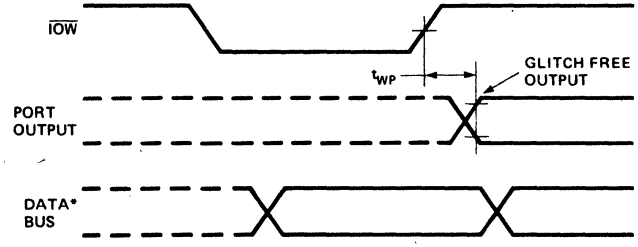
WAVEFORMS (Continued)

I/O PORT

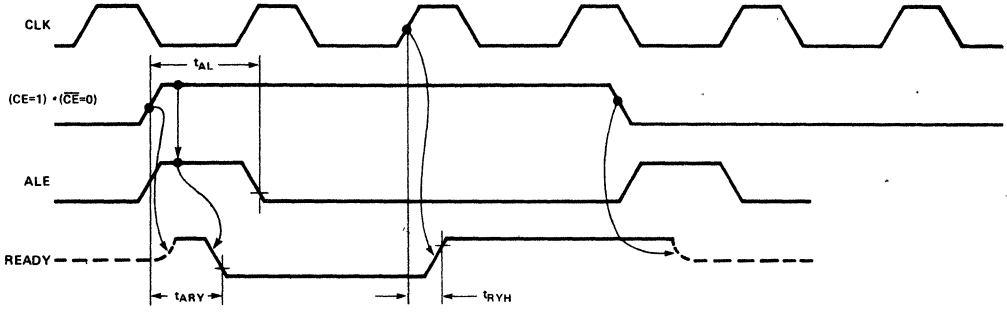
A. INPUT MODE



B. OUTPUT MODE

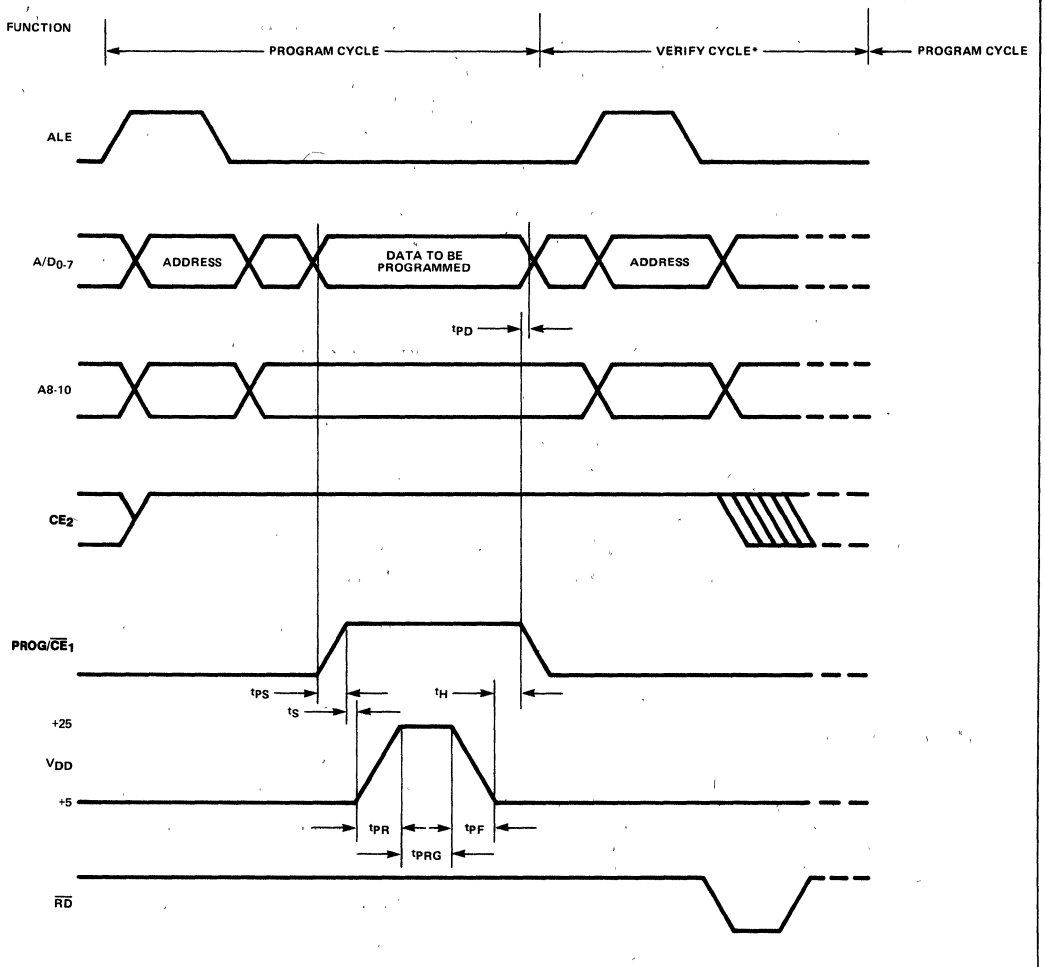


WAIT STATE (READY = 0)



WAVEFORMS (Continued)

8755A PROGRAM MODE



*VERIFY CYCLE IS A REGULAR MEMORY READ CYCLE (WITH V_{DD} = +5V FOR 8755A)

iAPX 86, 88, 186, 188 Microprocessors

3

Microprocessors Section

intel

APPLICATION NOTE

AP-113

February 1981

Getting Started With the Numeric Data Processor

Bill Rash
Microcomputer Applications

INTRODUCTION

This is an application note on using numerics in Intel's iAPX 86 or iAPX 88 microprocessor family. The numerics implemented in the family provide instruction level support for high-precision integer and floating point data types with arithmetic operations like add, subtract, multiply, divide, square root, power, log and trigonometrics. These features are provided by members of the iAPX 86 or iAPX 88 family called numeric data processors.

Rather than concentrate on a narrow, specific application, the topics covered in this application note were chosen for generality across many applications. The goal is to provide sufficient background information so that software and hardware engineers can quickly move beyond needs specific to the numeric data processor and concentrate on the special needs of their application. The material is structured to allow quick identification of relevant material without reading all the material leading up to that point. Everyone should read the introduction to establish terminology and a basic background.

IAPX 86,88 BASE

The numeric data processor is based on an 8088 or 8086 microprocessor. The 8086 and 8088 are general purpose microprocessors, designed for general data processing applications. General applications need fast, efficient data movement and program control instructions. Actual arithmetic on data values is simple in general applications. The 8086 and 8088 fulfill these needs in a low cost, effective manner.

However, some applications need more powerful arithmetic instructions and data types than a general purpose data processor provides. The real world deals in fractional values and requires arithmetic operations like square root, sine, and logarithms. Integer data types and their operations like add, subtract, multiply, and divide may not meet the needs for accuracy, speed, and ease of use.

Such functions are not simple or inexpensive. The general data processor does not provide these features due to their cost to other less-complex applications that do not need such features. A special processor is required, one which is easy to use and has a high level of support in hardware and software.

The numeric data processor provides these features. It supports the data types and operations needed and allows use of all the current hardware and software support for the iAPX 86/10 and 88/10 microprocessors.

The iAPX 86 and iAPX 88 provide two implementations of a numeric data processor. Each offers different tradeoffs in performance, memory size, and cost.

One alternative uses a special hardware component, the 8087 numeric processor extension, while the other is based on software, the 8087 emulator. Both component and software emulator add the extra numerics data types and operations to the 8086 or 8088.

The component and its software emulator are completely compatible.

Nomenclature

Table one shows several possible configurations of the iAPX 86 and iAPX 88 microprocessor family. The choice of configuration will be decided by the needs of the application for cost and performance in the areas of general data processing, numerics, and I/O processing. The combination of an 8086 or 8088 with an 8087 is called an iAPX 86/20 or 88/20 numeric data processor. For applications requiring high I/O bandwidths and numeric performance, a combination of 8086, 8087 and 8089 is an iAPX 86/21 numerics and I/O data processor. The same system with an 8088 CPU for smaller size and lower cost, due to the smaller 8-bit wide system data bus, is referred to as an iAPX 88/21. Each 8089 in the system is designated in the units digit of the system designation. The term 86/2X or 88/2X refers to a numeric data processor with any number of 8089s.

Throughout this application note, I will use the terms NDP, numeric data processor, 86/2X, and 88/2X synonymously. Numeric processor extension and NPX are also synonymous for the functions of either the 8087 component or 8087 emulator. The term numeric instruction or numeric data type refers to an instruction or data type made available by the NPX. The term host will refer to either the 8086 or 8088 microprocessor.

Table 1. Components Used in iAPX 86,88 Configurations

System Name	8086	8087	8088	8089
iAPX 86/10	1			
iAPX 86/11	1			1
iAPX 86/12	1			2
iAPX 86/20	1	1		
iAPX 86/21	1	1		1
iAPX 86/22	1	1		2
iAPX 88/10			1	
iAPX 88/11			1	1
iAPX 88/12			1	2
iAPX 88/20		1	1	
iAPX 88/21		1	1	1
iAPX 88/22		1	1	2

NPX OVERVIEW

The 8087 is a coprocessor extension available to iAPX 86/1X or iAPX 88/1X maximum mode microprocessor systems. (See page 7). The 8087 adds hardware support for floating point and extended precision integer data types, registers, and instructions. Figure 1 shows the register set available to the NDP. On the next page, the seven data types available to numeric instructions are listed (Fig 2). Each data type has a load and store instruction. Independent of whether an 8087 or its emulator are used, the registers and data types all appear the same to the programmer.

All the numeric instructions and data types of the NPX are used by the programmer in the same manner as the general data types and instructions of the host.

The numeric data formats and arithmetic operations provided by the 8087 conform to the proposed IEEE Microprocessor Floating Point Standard. All the proposed IEEE floating point standard algorithms, exception detection, exception handling, infinity arithmetic and rounding controls are implemented.[1]

The numeric registers of the NPX are provided for fast, easy reference to values needed in numeric calculations. All numeric values kept in the NPX register file are held in the 80-bit temporary real floating point format which is the same as the 80-bit temporary real data type.

All data types are converted to the 80-bit register file format when used by the NPX. Load and store instructions automatically convert between the memory operand data type and the register file format for all numeric data types. The numeric load instruction specifies the format in which the memory operand is expected and which addressing mode to use.

All host base registers, index registers, segment registers, and addressing modes are available for locating numeric operands. In the same manner, the store instruction also specifies which data type to use and where the value is located when stored into memory.

Selecting Numeric Data Types

As figure 2 shows, the numeric data types are of different lengths and domains (real or integer). Each numeric data type is provided for a specific function, they are:

16-bit word integers	—Index values, loop counts, and small program control values
32-bit short integers	—Large integer general computation
64-bit long integers	—Extended range integer computation
18-digit packed decimal	—Commercial and decimal conversion arithmetic
32-bit short real	—Reduced range and accuracy is traded for reduced memory requirements
64-bit long real	—Recommended floating point variable type
80-bit temporary real	—Format for intermediate or high precision calculations

Referencing memory data types in the NDP is not restricted to load and store instructions. Some arithmetic operations can specify a memory operand in one of four possible data types. The numeric instructions compare, add, subtract, subtract reversed, multiply, divide, and divide reversed can specify a memory operand to be either a 16-bit integer, 32-bit integer, 32-bit real, or 64-bit real value. As with the load and store operations, the arithmetic instruction specifies the address and expected format of the memory operand.

The remaining arithmetic operations: square root, modulus, tangent, arctangent, logarithm, exponentiate, scale power, and extract power use only register operands.

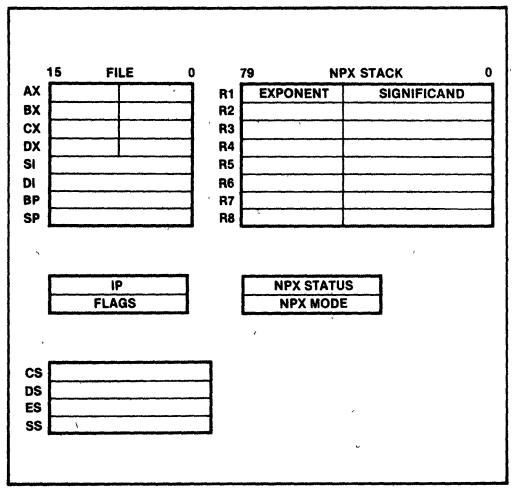


Figure 1. NDP Register Set for iAPX 86/20, 88/20

[1] "An Implementation Guide to a Proposed Standard for Floating Point" by Jerome Coonen in *Computer*, Jan. 1980 or the Oct. 1979 issue of *ACM SIGNUM*, for more information on the standard.

The register set of the host and 8087 are in separate components. Direct transfer of values between the two register sets in one instruction is not possible. To transfer values between the host and numeric register sets, the value must first pass through memory. The memory format of a 16-bit short integer used by the NPX is identical to that of the host, ensuring fast, easy transfers.

Since an 8086 or 8088 does not provide single instruction support for the remaining numeric data types, host programs reading or writing these data types must conform to the bit and byte ordering established by the NPX.

Writing programs using numeric instructions is as simple as with the host's instructions. The numeric instructions are simply placed in line with the host's instructions. They are executed in the same order as they appear in the instruction stream. Numeric instructions follow the same form as the host instructions. Figure 2 shows the ASM 86/88 representations for different numeric instructions and their similarity to host instructions.

8087 EMULATOR OVERVIEW

The NDP has two basic implementations, an 8087 component or with its software emulator (E8087). The decision to use the emulator or component has no effect on programs at the source level. At the source level, all instructions, data types, and features are used the same way.

The emulator requires all numeric instruction opcodes to be replaced with an interrupt instruction. This replacement is performed by the LINK86 program. Interrupt vectors in the host's interrupt vector table will point to numeric instruction emulation routines in the 8087 software emulator.

When using the 8087 emulator, the linker changes all the 2-byte wait-escape, nop-escape, wait-segment override, or nop-segment override sequences generated by an assembler or compiler for the 8087 component with a 2-byte interrupt instruction. Any remaining bytes of the numeric instruction are left unchanged.

```
FILD        VALUE
FIADD       TABLE [BX]
FADD        ST,ST(1)
```

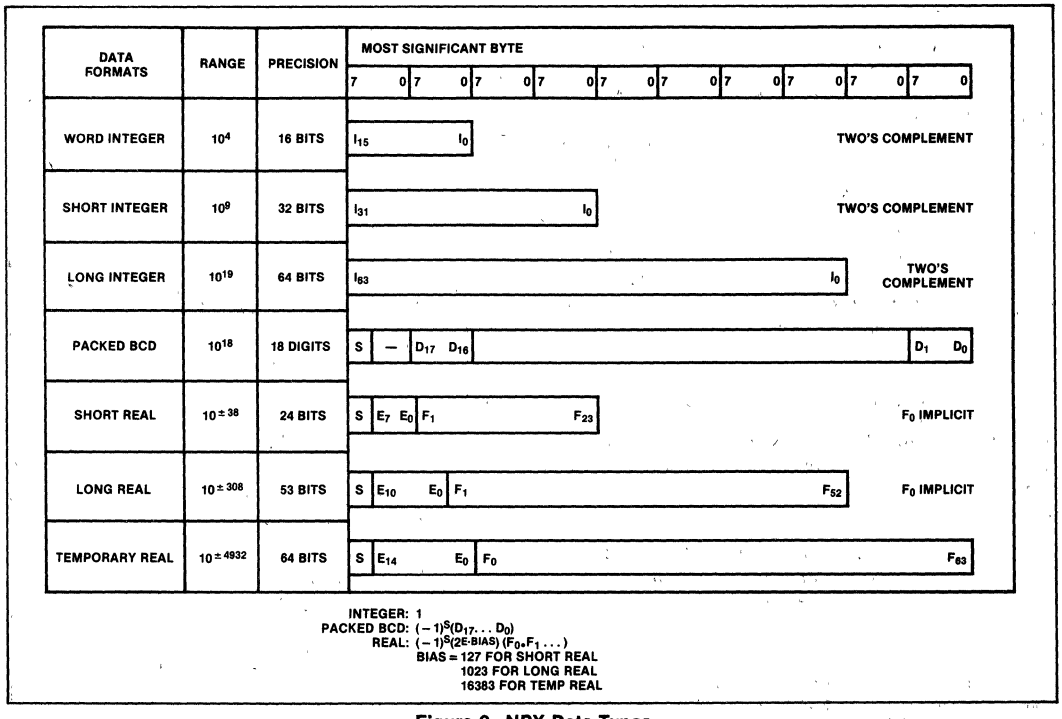


Figure 2. NPX Data Types

When the host encounters numeric and emulated instruction, it will execute the software interrupt instruction formed by the linker. The interrupt vector table will direct the host to the proper entry point in the 8087 emulator. Using the interrupt return address and CPU register set, the host will decode any remaining part of the numeric instruction, perform the indicated operation, then return to the next instruction following the emulated numeric instruction.

One copy of the 8087 emulator can be shared by all programs in the host.

The decision to use the 8087 or software emulator is made at link time, when all software modules are brought together. Depending on whether an 8087 or its software emulator is used, a different group of library modules are included for linking with the program.

If the 8087 component is used, the libraries do not add any code to the program, they just satisfy external references made by the assembler or compiler. Using the emulator will not increase the size of individual modules; however, other modules requiring about 16K bytes that implement the emulator will be automatically added.

Selecting between the emulator or the 8087 can be very easy. Different versions of submit files performing the link operation can be used to specify the different set of library modules needed. Figure 3 shows an example of two different submit files for the same program using the NPX with an 8087 or the 8087 emulator.

```
8087 BASED LINK/LOCATE COMMANDS

LINK86 :F1:PROG.OBJ, IO.LIB, 8087.LIB TO
       :F1:PROG.LNK
LOC86  :F1:PROG.LNK TO :F1:PROG


SOFTWARE EMULATOR BASED
LINK/LOCATE COMMANDS

LINK86 :F1:PROG.OBJ,  IO.LIB, E8087.LIB,
       E8087 TO :F1:PROG.LNK
LOC86  :F1:PROG.LNK TO :F1:PROG
```

Figure 3. Submit File Example

iSBC 337™ MULTIMODULE™ Overview

The benefits of the NPX are not limited to systems which left board space for the 8087 component or memory space for its software emulator. Any maximum mode iAPX 86/1X or iAPX 88/1X system can be upgraded to a numeric processor. The iSBC 337 MULTIMODULE is designed for just this function. The iSBC 337 provides a socket for the host microprocessor and an 8087. A 40-pin plug is provided on the underside of the 337 to plug into the original host's socket, as shown in Figure 4. Two other pins on the underside of the MULTIMODULE allow easy connection to the 8087 INT and RQ/GT1 pins.

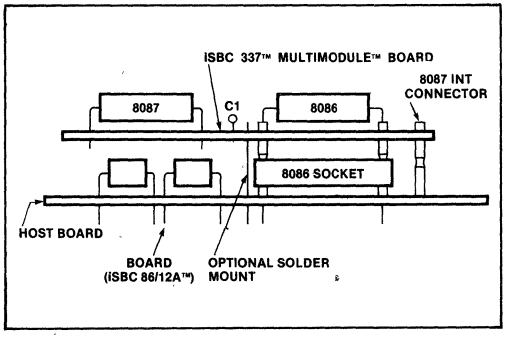


Figure 4. MULTIMODULE™ Math Mounting Scheme

CONSTRUCTING AN iAPX 86/2X OR iAPX 88/2X SYSTEM

This section will describe how to design a microprocessor system with the 8087 component. The discussion will center around hardware issues. However, some of the hardware decisions must be made based upon how the software will use the NPX. To better understand how the 8087 operates as a local bus master, we shall cover how the coprocessor interface works later in this section.

Wiring up the 8087

The 8087 can be designed into any 86/1X or 88/1X system operating in maximum mode. Such a system would be designated an 86/2X or 88/2X. Figure 5 shows the local bus interconnections for an iAPX 86/20 (or iAPX 88/20) system. The 8087 shares the maximum mode host's multiplexed address/data bus, status signals, queue status signals, ready status signal, clock and reset signal. Two dedicated signals, BUSY and INT, inform the host of current 8087 status. The 10K pull-down resistor on the BUSY signal ensures the host will always see a "not busy" status if an 8087 is not installed.

Adding the 8087 to your design has a minor effect on hardware timing. The 8087 has the exact same timing and equivalent DC and AC drive characteristics as a host or IOP on the local bus. All the local bus logic, such as clock, ready, and interface logic is shared.

The 8087 adds 15 pF to the total capacitive loading on the shared address/data and status signals. Like the 8086 or 8088, the 8087 can drive a total of 100 pF capacitive load above its own self load and sink 2.0 mA DC current on these pins. This AC and DC drive is sufficient for an 86/21 system with two sets of data transceivers, address latches, and bus controllers for two separate busses, an on-board bus and an off-board MULTIBUS™ using the 8289 bus arbiter.

Later in this section, what to do with the 8087 INT and RQ/GT pins, is covered.

It is possible to leave a prewired 40-pin socket on the board for the 8087. Adding the 8087 to such a system is as easy as just plugging it in. If a program attempts to execute any numeric instructions without the 8087 installed, they will be simply treated as NOP instructions by the host. Software can test for the existence of the 8087 by initializing it and then storing the control word. The program of Figure 6 illustrates this technique.

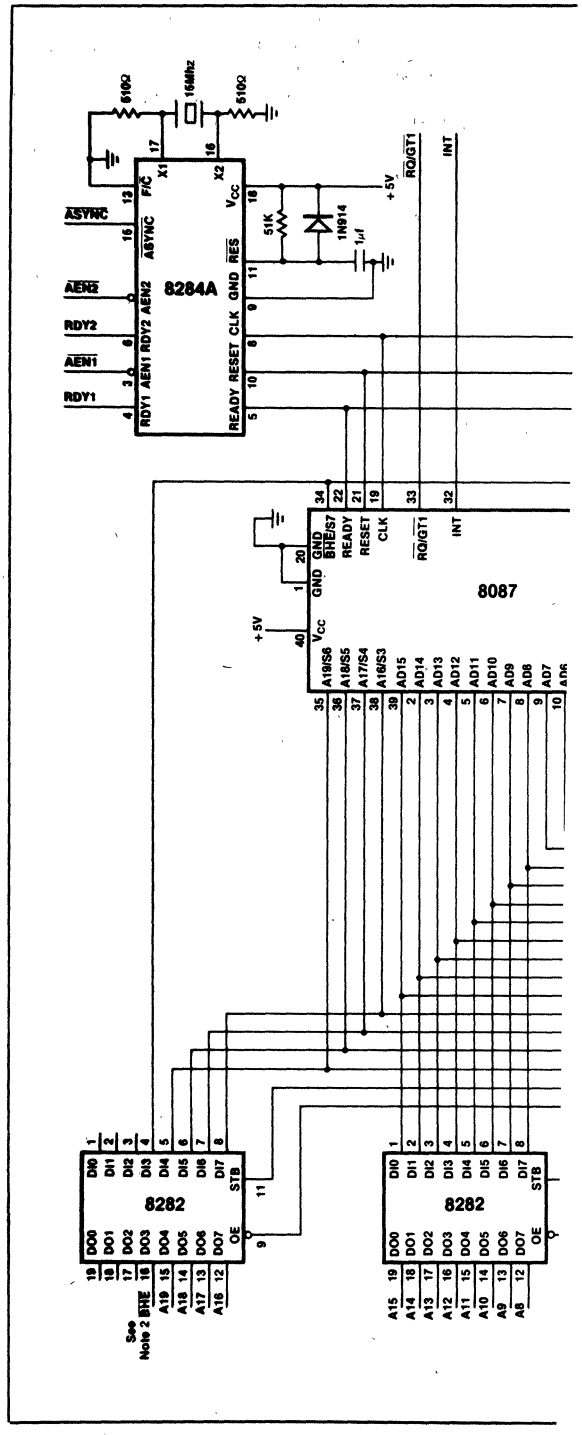
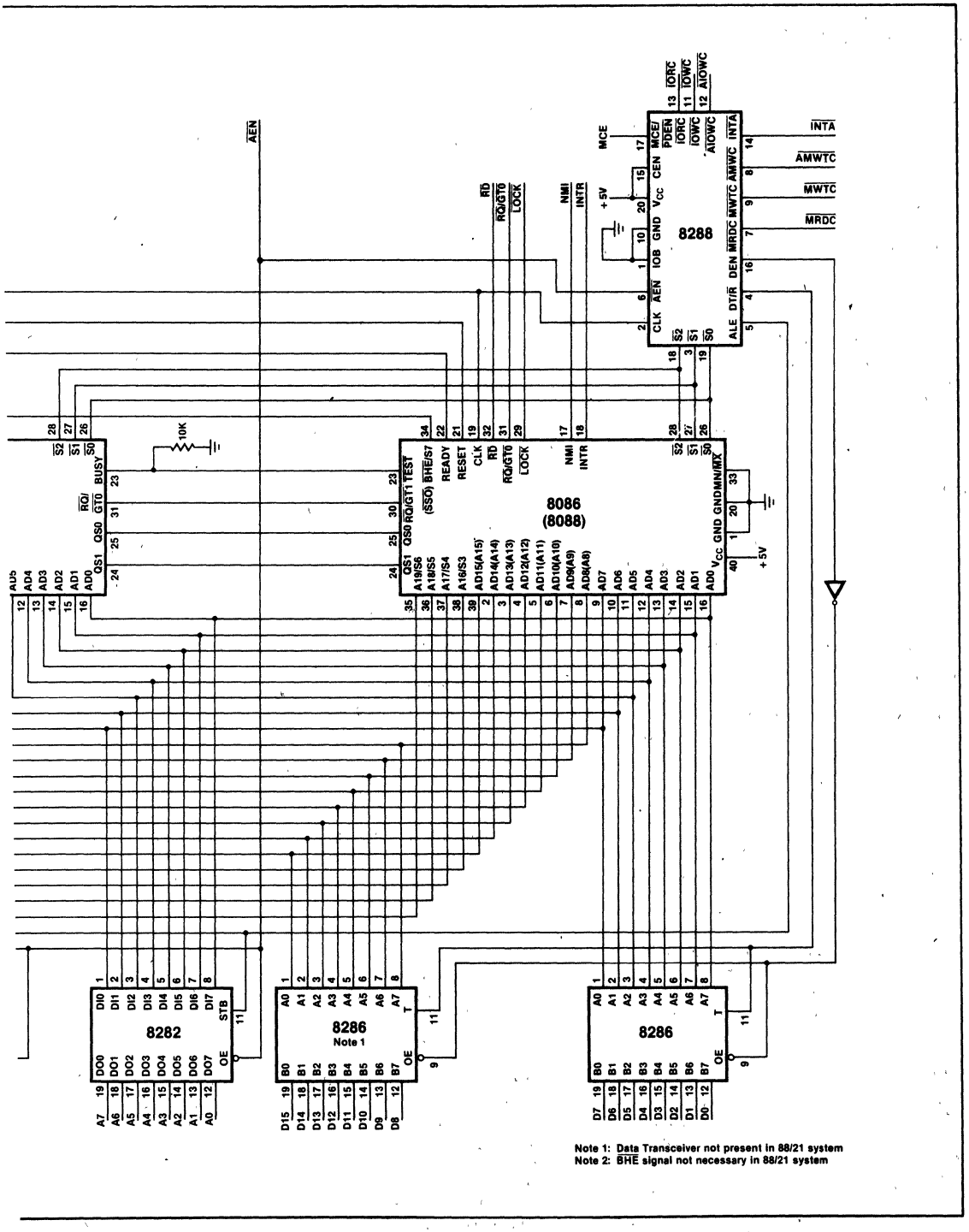


Figure 5. System Diagram


8086
(8088)
8288
8282
8286
Note 1
8286
10K
+ 5V
AEN
MCE
INTA
AMWTC
MWTC
MRDC
Note 1: Data Transceiver not present in 88/21 system
Note 2: BHE signal not necessary in 88/21 system

WHAT IS THE iAPX 86, 88 COPROCESSOR INTERFACE?

The idea of a coprocessor is based on the observation that hardware specially designed for a function is the fastest, smallest, and cheapest implementation. But, it is too expensive to incorporate all desired functions in general purpose hardware. Few applications could use all the functions. To build fast, small, economical systems, we need some way to mix and match components supporting specialized functions.

Purpose of the Coprocessor Interface

The coprocessor interface of the general purpose 8086 or 8088 microprocessor provides a way to attach specialized hardware in a simple, elegant, and efficient manner. Because the coprocessor hardware is specialized, it can perform its job much faster than any general purpose CPU of similar size and cost. The coprocessor interface simply requires connection to the host's local address/data, status, clock, ready, reset, test and request/grant signals. Being attached to the host's local bus gives the coprocessor access to all memory and I/O resources available to the host.

The coprocessor is independent of system configuration. Using the local bus as the connection point to the host isolates the coprocessor from the particular system configuration, since the timing and function of local bus signals are fixed.

Software's View of the Coprocessor

The coprocessor interface allows specialized hardware to appear as an integral part of the host's architecture controlled by the host with special instructions. When the host encounters these special instructions, both the host and coprocessor recognize them and work together to perform the desired function. No status polling loops or command stuffing sequences are required by software to operate the coprocessor.

More information is available to a coprocessor than simply an instruction opcode and a signal to begin execution. The host's coprocessor interface can read a value from memory, or identify a region of memory the coprocessor should use while performing its function. All the addressing modes of the host are available to identify memory based operands to the coprocessor.

Concurrent Execution of Host and Coprocessor

After the coprocessor has started its operation, the host may continue on with the program, executing it in parallel while the coprocessor performs the function started earlier. The parallel operation of the coprocessor does not normally affect that of the host unless the coprocessor must reference memory or I/O-based operands. When the host releases the local bus to the coprocessor, the host may continue to execute from its internal instruction queue. However, the host must stop when it also needs the local bus currently in use by the coprocessor. Except for the stolen memory cycle, the operation of the coprocessor is transparent to the host.

This parallel operation of host and coprocessor is called concurrent execution. Concurrent execution of instructions requires less total time then a strictly sequential execution would. System performance will be higher with concurrent execution of instructions between the host and coprocessor.

SYNCHRONIZATION

In exchange for the higher system performance made available by concurrent execution, programs must provide what is called synchronization between the host and coprocessor. Synchronization is necessary whenever the host and coprocessor must use information available from the other. Synchronization involves either the host or coprocessor waiting for the other to finish an operation currently in progress. Since the host executes the program, and has program control instructions like jumps, it is given responsibility for synchronization. To meet this need, a special host instruction exists to synchronize host operation with a coprocessor.

```
;
; Test for the existence of an 8087 in the system. This code will always recognize an 8087
; independent of the TEST pin usage on the host. No deadlock is possible. Using the 8087
; emulator will not change the function of this code since ESC instructions are used. The word
; variable control is used for communication between the 8087 and the host. Note: if an 8087 is
; present, it will be initialized. Register ax is not transparent across this code.
;
ESC   28, bx          ; FNINIT if 8087 is present . The contents of bx is irrelevant
XOR   ax, ax          ; These two instructions insert delay while the 8087 initializes itself
MOV   control, ax     ; Clear intial control word value
ESC   15, control     ; FNSTCW if 8087 is present
OR    ax, control     ; Control = 03ffh if 8087 present
JZ    no_8087         ; Jump if no 8087 is present
```

Figure 6. Test for Existence of an 8087

The host coprocessor synchronization instruction, called "WAIT", uses the TEST pin of the host. The coprocessor can signal that it is still busy to the host via this pin. Whenever the host executes a wait instruction, it will stop program execution while the TEST input is active. When the TEST pin becomes inactive, the host will resume program execution with the next instruction following the WAIT. While waiting on the TEST pin, the host can be interrupted at 5 clock intervals; however, after the TEST pin becomes inactive, the host will immediately execute the next instruction, ignoring any pending interrupts between the WAIT and following instruction.

COPROCESSOR CONTROL

The host has the responsibility for overall program control. Coprocessor operation is initiated by special instructions encountered by the host. These instructions are called "ESCAPE" instructions. When the host encounters an ESCAPE instruction, the coprocessor is expected to perform the action indicated by the instruction. There are 576 different ESCAPE instructions, allowing the coprocessor to perform many different actions.

The host's coprocessor interface requires the coprocessor to recognize when the host has encountered an ESCAPE instruction. Whenever the host begins executing a new instruction, the coprocessor must look to see if it is an ESCAPE instruction. Since only the host fetches instructions and executes them, the coprocessor must monitor the instructions being executed by the host.

Host Queue Tracking

The host can fetch an instruction at a variable length time before the host executes the instruction. This is a characteristic of the instruction queue of an 8086 or 8088 microprocessor. An instruction queue allows prefetching instructions during times when the local bus would be otherwise idle. The end benefit is faster execution time of host instructions for a given memory bandwidth.

The host does not externally indicate which instruction it is currently executing. Instead, the host indicates when it fetches an instruction and when, some time later, an opcode byte is decoded and executed. To identify the actual instruction the host fetched from its queue, the coprocessor must also maintain an instruction stream identical to the host's.

Instructions can be fetched in byte or word increments, depending on the type of host and the destination address of jump instructions executed by the host. When the host has filled its queue, it stops prefetching instructions. Instructions are removed from the queue a byte at a time for decoding and execution. When a jump occurs, the queue is emptied. The coprocessor follows these actions in the host by monitoring the host's bus status, queue status, and data bus signals. Figure 7 shows how the bus status signals and queue status signals are encoded.

IGNORING I/O PROCESSORS

The host is not the only local bus master capable of fetching instructions. An Intel 8089 IOP can generate instruction fetches on the local bus in the course of executing a channel program in system memory. In this case, the status signals S2, S1, and S0 generated by the IOP are identical to those of the host. The coprocessor must not interpret these instruction prefetches as going to the host's instruction queue. This problem is solved with a status signal called S6. The S6 signal identifies when the local bus is being used by the host. When the host is the local bus master, S6 = 0 during T2 and T3 of the memory cycle. All other bus masters must set S6 = 1 during T2 and T3 of their instruction prefetch cycles. Any coprocessor must ignore activity on the local bus when S6 = 1.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	Function	QS1	QS0	Host Function	Coprocessor Activity
0	0	0	Interrupt Acknowledge	0	0	No Operation	No Queue Activity
0	0	1	Read I/O Port	0	1	First Byte	Decode Opcode Byte
0	1	0	Write I/O Port	1	0	Empty Queue	Empty Queue
0	1	1	Halt	1	1	Subsequent Byte	Flush Byte or if 2nd
1	0	0	Code Fetch				Byte of Escape
1	0	1	Read Data Memory				Decode it
1	1	0	Write Data Memory				
1	1	1	Idle				

Figure 7.

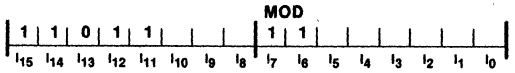
DECODING ESCAPE INSTRUCTIONS

To recognize ESCAPE instructions, the coprocessor must examine all instructions executed by the host. When the host fetches an instruction byte from its internal queue, the coprocessor must do likewise.

The queue status state, fetch opcode byte, identifies when an opcode byte is being examined by the host. At the same time, the coprocessor will check if the byte fetched from its internal instruction queue is an ESCAPE opcode. If the instruction is not an ESCAPE, the coprocessor will ignore it. The queue status signals for fetch subsequent byte and flush queue let the coprocessor track the host's queue without knowledge of the length and function of host instructions and addressing modes.

Escape Instruction Encoding

All ESCAPE instructions start with the high-order 5-bits of the instruction being 11011. They have two basic forms. The non-memory form, listed here, initiates some activity in the coprocessor using the nine available bits of the ESCAPE instruction to indicate which function to perform.



Memory reference forms of the ESCAPE instruction, shown in Figure 8, allow the host to point out a memory operand to the coprocessor using any host memory addressing mode. Six bits are available in the memory reference form to identify what to do with the memory operand. Of course, the coprocessor may not recognize all possible ESCAPE instructions, in which case it will simply ignore them.

Memory reference forms of ESCAPE instructions are identified by bits 7 and 6 of the byte following the ESCAPE opcode. These two bits are the MOD field of the 8086 or 8088 effective address calculation byte. They, together with the R/M field, bits 2 through 0, determine the addressing mode and how many subsequent bytes remain in the instruction.

Host's Response to an Escape Instruction

The host performs one of two possible actions when encountering an ESCAPE instruction: do nothing or calculate an effective address and read a word value beginning at that address. The host ignores the value of the word read. ESCAPE instructions change no registers in the host other than advancing IP. So, if there is no coprocessor, or the coprocessor ignores the ESCAPE instruction, the ESCAPE instruction is effectively a NOP to the host. Other than calculating a memory address and reading a word of memory, the host makes no other assumptions regarding coprocessor activity.

The memory reference ESCAPE instructions have two purposes: identify a memory operand and for certain instructions, transfer a word from memory to the coprocessor.

COPROCESSOR INTERFACE TO MEMORY

The design of a coprocessor is considerably simplified if it only requires reading memory values of 16 bits or less. The host can perform all the reads with the coprocessor latching the value as it appears on the data bus at the end of T3 during the memory read cycle. The coprocessor need never become a local bus master to read or write additional information.

If the coprocessor must write information to memory, or deal with data values longer than one word, then it must save the memory address and be able to become a local bus master. The read operation performed by the host in the course of executing the ESCAPE instruction places the 20-bit physical address of the operand on the address/data pins during T1 of the memory cycle. At this time the coprocessor can latch the address. If the coprocessor instruction also requires reading a value, it will appear on the data bus during T3 of the memory read. All other memory bytes are addressed relative to this starting physical address.

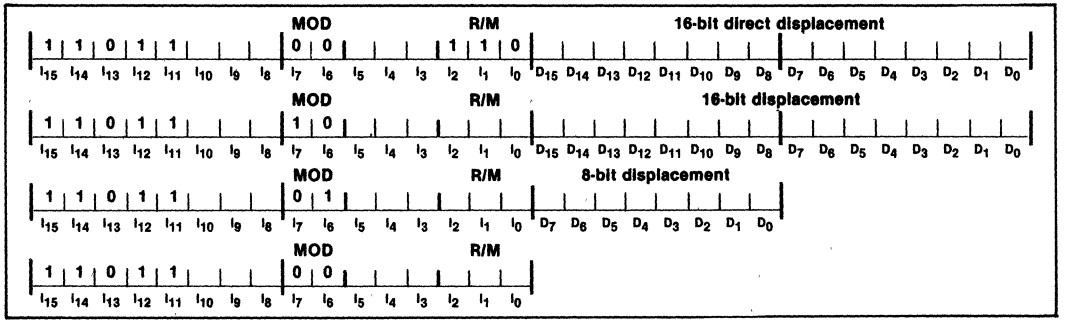


Figure 8. Memory Reference Escape Instruction Forms

Whether the coprocessor becomes a bus master or not, if the coprocessor has memory reference instruction forms, it must be able to identify the memory read performed by the host in the course of executing an ESCAPE instruction.

Identifying the memory read is straightforward, requiring all the following conditions to be met:

1) A MOD value of 00, 01, or 10 in the second byte of the ESCAPE instruction executed by the host.
2) This is the first data read memory cycle performed by the host after it encountered the ESCAPE instruction. In particular, the bus status signals S2–S0 will be 101 and S6 will be 0.

The coprocessor must continue to track the instruction queue of the host while it calculates the memory address and reads the memory value. This is simply a matter of following the fetch subsequent byte status commands occurring on the queue status pins.

HOST PROCESSOR DIFFERENCES

A coprocessor must be aware of the bus characteristics of the host processor. This determines how the host will read the word operand of a memory reference ESCAPE instruction. If the host is an 8088, it will always perform two byte reads at sequential addresses. But if the host is an 8086, it can either perform a single word read or two byte reads to sequential addresses.

The 8086 places no restrictions on the alignment of word operands in memory. It will automatically perform two byte operations for word operands starting at an odd address. The two operations are necessary since the two bytes of the operand exist in two different memory words. The coprocessor should be able to accept the two possible methods of reading a word value on the 8086.

A coprocessor can determine whether the 8086 will perform one or two memory cycles as part of the current ESCAPE instruction execution. The AD0 pin during T1 of the first memory read by the host tells if this is the only read to be performed as part of the ESCAPE instruction. If this pin is a 1 during T1 of the memory cycle, the 8086 will immediately follow this memory read cycle with another one at the next byte address.

Coprocessor Interface Summary

The host ESCAPE instructions, coprocessor interface, and WAIT instruction allow easy extension of the host's architecture with specialized processors. The 8087 is such a processor, extending the host's architecture as seen by the programmer. The specialized hardware provided by the 8087 can greatly improve system performance economically in terms of both hardware and software for numerics applications.

The next section examines how the 8087 uses the coprocessor interface of the 8086 or 8088.

8087 COPROCESSOR OPERATION

The 8086 or 8088 ESCAPE instructions provide 64 memory reference opcodes and 512 non-memory reference opcodes. The 8087 uses 57 of the memory reference forms and 406 of the non-memory reference forms. Figure 9 shows the ESCAPE instructions not used by the 8087.

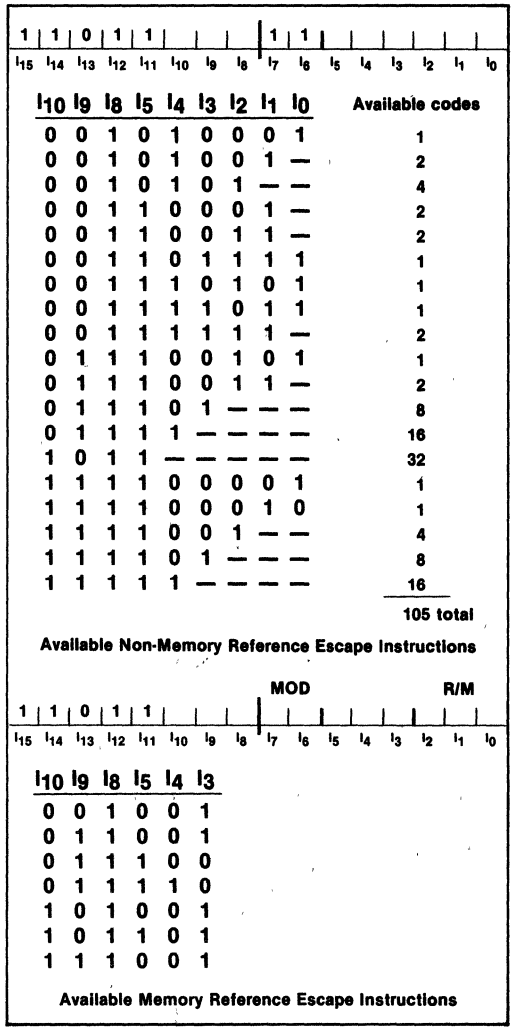


Figure 9.

Using the 8087 With Custom Coprocessors

Custom coprocessors, a designer may care to develop, should limit their use of ESCAPE instructions to those not used by the 8087 to prevent ambiguity about whether any one ESCAPE instruction is intended for a numerics or other custom coprocessor. Using any escape instruction for a custom coprocessor may conflict with opcodes chosen for future Intel coprocessors.

Operation of an 8087 together with other custom coprocessors is possible under the following constraints:

1) All 8087 errors are masked. The 8087 will update its opcode and instruction address registers for the unused opcodes. Unused memory reference instructions will also update the operand address value. Such changes in the 8087 make software-defined error handling impossible.
2) If the coprocessors provide a BUSY signal, they must be ORed together for connection to the host TEST pin. When the host executes a WAIT instruction, it does not know which coprocessor will be affected by the following ESCAPE instruction. In general, all coprocessors must be idle before executing the ESCAPE instruction.

Operand Addressing by the 8087

The 8087 has seven different memory operand formats. Six of them are longer than one word. All are an even number of bytes in length and are addressed by the host at the lowest address word.

When the host executes a memory reference ESCAPE instruction intended to cause a read operation by the 8087, the host always reads the low-order word of any 8087 memory operand. The 8087 will save the address and data read. To read any subsequent words of the operand, the 8087 must become a local bus master.

When the 8087 has the local bus, it increments the 20-bit physical address it saved to address the remaining words of the operand.

When the ESCAPE instruction is intended to cause a write operation by the 8087, the 8087 will save the address but ignore the data read. Eventually, it will get control of the local bus, then perform successive write, increment address operations writing the entire data value.

8087 OPERATION IN IAPX 86,88 SYSTEMS

The 8087 will work with either an 8086 or 8088 host. The identity of the host determines the width of the local bus path. The 8087 will identify the host and adjust its use of the data bus accordingly; 8 bits for an 8088 or 16 bits for an 8086. No strapping options are required by the 8087; host identification is automatic.

The 8087 identifies the host each time the host and 8087 are reset via the RESET pin. After the reset signal goes inactive, the host will begin instruction execution at memory address $FFFF0_{16}$.

If the host is an 8086 it will perform a word read at that address; an 8088 will perform a byte read.

The 8087 monitors pin 34 on the first memory cycle after power up. If an 8086 host is used, pin 34 will be the BHE signal, which will be low for that memory cycle. For an 8088 host, pin 34 will be the SS0 signal, which will be high during T1 of the first memory cycle. Based on this signal, the 8087 will then configure its data bus width to match that of the host local bus.

For 88/2X systems, pin 34 of the 8087 may be tied to V_{CC} if not connected to the 8088 SS0 pin.

The width of the data bus and alignment of data operands has no effect, except for execution time and number of memory cycles performed, on 8087 instructions. A numeric program will always produce the same results on an 86/2X or 88/2X with any operand alignment. All numeric operands have the same relative byte orderings independent of the host and starting address.

The byte alignment of memory operands can affect the performance of programs executing on an 86/2X. If a word operand, or any numeric operand, starts on an odd-byte address, more memory cycles are required to access the operand than if the operand started on an even address. The extra memory cycles will lower system performance.

The 86/2X will attempt to minimize the number of extra memory cycles required for odd-aligned operands. In these cases, the 8087 will perform first a byte operation, then a series of word operations, and finally a byte operation.

88/2X instruction timings are independent of operand alignment, since byte operations are always performed. However, it is recommended to align numeric operands on even boundaries for maximum performance in case the program is transported to an 86/2X.

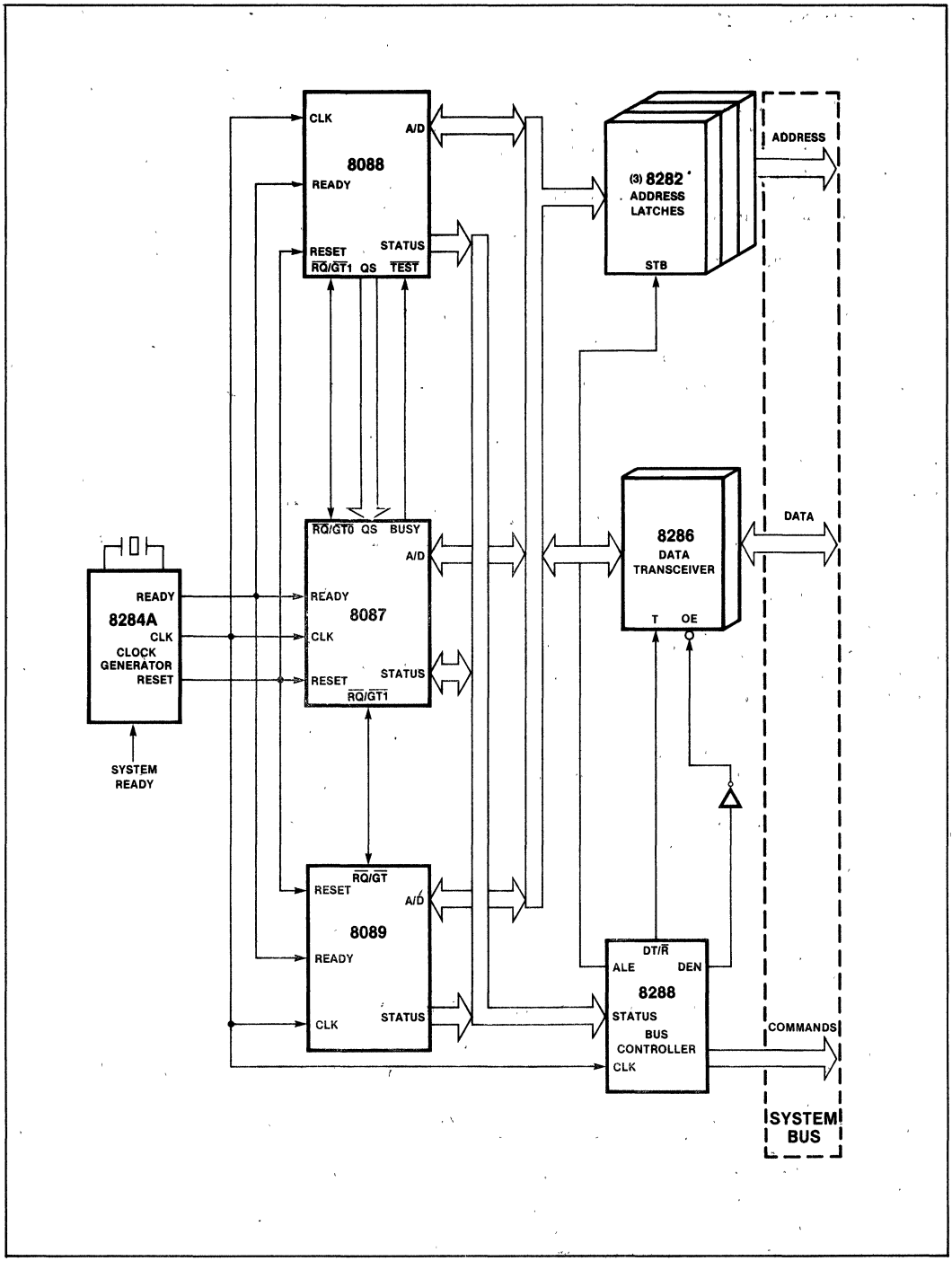


Figure 10. iAPX 88/21

RQ/GT CONNECTION

Two decisions must be made when connecting the 8087 to a system. The first is how to interconnect the RQ/GT signals of all local bus masters. The RQ/GT decision affects the response time to service local bus requests from other local bus masters, such as an 8089 IOP or other coprocessor. The interrupt connection affects the response time to service an interrupt request and how user-interrupt handlers are written. The implications of how these pins are connected concern both the hardware designer and programmer and must be understood by both.

The RQ/GT issue can be broken into three general categories, depending on system configuration: 86/20 or 88/20, 86/21 or 88/21, and 86/22 or 88/22. Remote operation of an IOP is not effected by the 8087 RQ/GT connection.

iAPX 86/20, 88/20

For an 86/20 or 88/20 just connect the RQ/GT0 pin of the 8087 to RQ/GT1 of the host (see Figure 5), and skip forward to the interrupt discussion on page 15.

iAPX 86/21, 88/21

For an 86/21 or 88/21, connect RQ/GT0 of the 8087 to RQ/GT1 of the host, connect RQ/GT of the 8089 to RQ/GT1 of the 8087 (see Figure 10, page 12), and skip forward to the interrupt discussion on page 15.

The RQ/GT1 pin of the 8087 exists to provide one I/O processor with a low maximum wait time for the local bus. The maximum wait times to gain control of the local bus for a device attached to RQ/GT1 of an 8087 for an 8086 or 8088 host are shown in Table 2. These numbers are all dependent on when the host will release the local bus to the 8087.

As Table 2 implies, three factors determine when the host will release the local bus:

1) What type of host is there, an 8086 or 8088?

2) What is the current instruction being executed?

3) How is the lock prefix being used?

An 8086 host will not release the local bus between the two consecutive byte operations performed for odd-aligned word operands. The 8088, in contrast, will never release the local bus between the two bytes of a word transfer, independent of its byte alignment.

Host operations such as acknowledging an interrupt will not release the local bus for several bus cycles.

Using a lock prefix in front of a host instruction prevents the host from releasing the local bus during the execution of that instruction.

8087 RQ/GT Function

The presence of the 8087 in the RQ/GT path from the IOP to the host has little effect on the maximum wait time seen by the IOP when requesting the local bus. The 8087 adds two clocks of delay to the basic time required by the host. This low delay is achieved due to a preemptive protocol implemented by the 8087 on RQ/GT1.

The 8087 always gives higher priority to a request for the local bus from a device attached to its RQ/GT1 pin than to a request generated internally by the 8087. If the 8087 currently owns the local bus and a request is made to its RQ/GT1 pin, the 8087 will finish the current memory cycle and release the local bus to the requestor. If the request from the devices arrives when the 8087 does not own the local bus, then the 8087 will pass the request on to the host via its RQ/GT0 pin.

Table 2. Worst Case Local Bus Request Wait Times in Clocks

System Configuration	No Locked Instructions	Only Locked Exchange	Other Locked Instructions
iAPX 86/21 even aligned words	15_1	35_1	max (15_1, *)
iAPX 86/21 odd aligned words	15_1	43_2	max (43_2, *)
iAPX 88/21	15_1	43_2	max (43_2, *)

Notes: 1. Add two clocks for each wait state inserted per bus cycle
2. Add four clocks for each wait state inserted per bus cycle
* Execution time of longest locked instruction

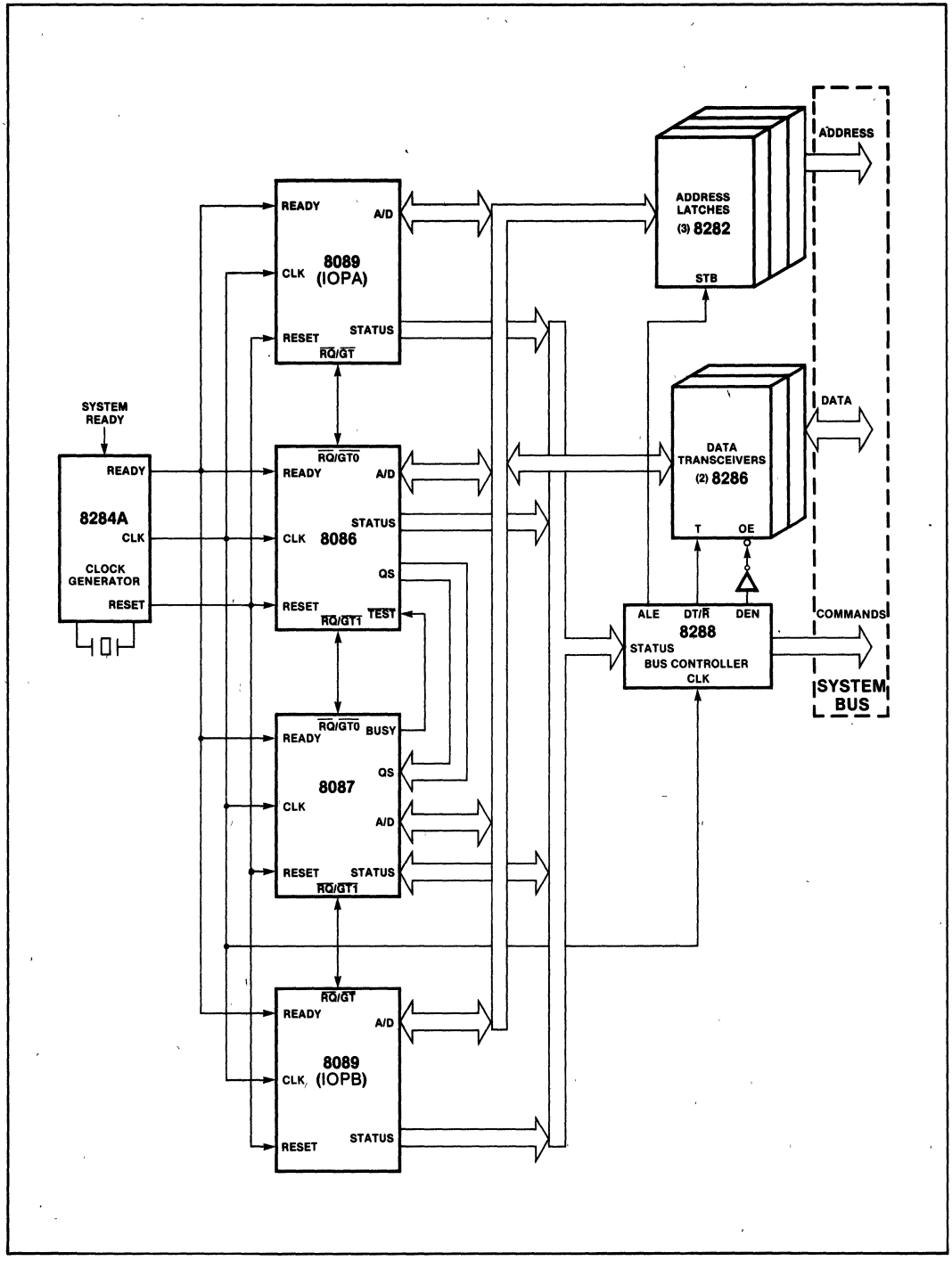


Figure 11. iAPX 86/22 System

iAPX 86/22, 88/22

An 86/22 system offers two alternates regarding to which IOP to connect an I/O device. Each IOP will offer a different maximum delay time to servide an I/O request. (See Fig. 11)

The second 8089 (IOPA) must use the RQ/GT0 pin of the host. With two IOPs the designer must decide which IOP services which I/O devices, determined by the maximum wait time allowed between when an I/O device requests IOP service and the IOP can respond. The maximum service delay times of the two IOPs can be very different. It makes little difference which of the two host RQ/GT pins are used.

The different wait times are due to the non-preemptive nature of bus grants between the two host RQ/GT pins. No communication of a need to use the local bus is possible between IOPA and the 8087/IOPB combination. Any request for the local bus by the IOPA must wait in the worst case for the host, 8087, and IOPB to finish their longest sequence of memory cycles. IOPB must wait in the worst case for the host and IOPA to finish their longest sequence of memory cycles. The 8087 has little effect on the maximum wait time of IOPB.

DELAY EFFECTS OF THE 8087

The delay effects of the 8087 on IOPA can be significant. When executing special instructions (FSAVE, FNSAVE, FRSTOR), the 8087 can perform 50 or 96 consecutive memory cycles with an 8086 or 8088 host, respectively. These instructions do not affect response time to local bus requests seen by an IOPB.

If the 8087 is performing a series of memory cycles while executing these instructions, and IOPB requests the local bus, the 8087 will stop its current memory activity, then release the local bus to IOPB.

The 8087 cannot release the bus to IOPA since it cannot know that IOPA wants to use the local bus, like it can for IOPB.

REDUCING 8087 DELAY EFFECTS

For 86/22 or 88/22 systems requiring lower maximum wait times for IOPA, it is possible to reduce the worst case bus usage of the 8087. If three 8087 instructions are never executed; namely FSAVE, FNSAVE, or FRSTOR, the maximum number of consecutive memory cycles performed by the 8087 is 10 or 16 for an 8086 or 8088 host respectively. The function of these instructions can be emulated with other 8087 instructions.

Appendix B shows an example of how these three instructions can be emulated. This improvment does have a cost, in the increased execution time of 427 or 747 additional clocks for an 8086 or 8088 respectively, for the equivalent save and restore operations. These operations appear in time-critical context-switching functions of an operating system or interrupt handler. This technique has no affect on the maximum wait time seen by IOPB or wait time seen by IOPA due to IOPB.

Which IOP to connect to which I/O device in an 86/22 or 88/22 system will depend on how quickly an I/O request by the device must be serviced by the IOP. This maximum time must be greater than the sum of the maximum delay of the IOP and the maximum wait time to gain control of the local bus by the IOP.

If neither IOP offers a fast enough response time, consider remote operation of the IOP.

8087 INT Connection

The next decision in adding the 8087 to an 8086 or 8088 system is where to attach the INT signal of the 8087. The INT pin of the 8087 provides an external indication of software-selected numeric errors. The numeric program will stop until something is done about the error. Deciding where to connect the INT signal can have important consequences on other interrupt handlers.

WHAT ARE NUMERIC ERRORS?

A numeric error occurs in the NPX whenever an operation is attempted with invalid operands or attempts to produce a result which cannot be represented. If an incorrect or questionable operation is attempted by a program, the NPX will always indicate the event. Examples of errors on the NPX are: 1/0, square root of -1, and reading from an empty register. For a detailed description of when the 8087 detects a numeric error, refer to the *Numerics Supplement*. (See Lit. Ref).

WHAT TO DO ABOUT NUMERIC ERRORS

Two possible courses of action are possible when a numeric error occurs. The NPX can itself handle the error, allowing numeric program execution to continue undisturbed, or software in the host can handle the error. To have the 8087 handle a numeric error, set its associated mask bit in the NPX control word. Each numeric error may be individually masked.

The NPX has a default fixup action defined for all possible numeric errors when they are masked. The default actions were carefully selected for their generality and safety.

For example, the default fixup for the precision error is to round the result using the rounding rules currently in effect. If the invalid error is masked, the NPX will generate a special value called indefinite as the result of any invalid operation.

NUMERIC ERRORS (CON'T)

Any arithmetic operation with an indefinite operand will always generate an indefinite result. In this manner, the result of the original invalid operation will propagate throughout the program wherever it is used.

When a questionable operation such as multiplying an unnormal value by a normal value occurs, the NPX will signal this occurrence by generating an unnormal result.

The required response by host software to a numeric error will depend on the application. The needs of each application must be understood when deciding on how to treat numeric errors. There are three attitudes towards a numeric error:

1) No response required. Let the NPX perform the default fixup.
2) Stop everything, something terrible has happened!
3) Oh, not again! But don't disrupt doing something more important.

SIMPLE ERROR HANDLING

Some very simple applications may mask all of the numeric errors. In this simple case, the 8087 INT signal may be left unconnected since the 8087 will never assert this signal. If any numeric errors are detected during the course of executing the program, the NPX will generate a safe result. It is sufficient to test the final results of the calculation to see if they are valid.

Special values like not-a-number (NAN), infinity, indefinite, denormals, and unnormals indicate the type and severity of earlier invalid or questionable operations.

SEVERE ERROR HANDLING

For dedicated applications, programs should not generate or use any invalid operands. Furthermore, all numbers should be in range. An operand or result outside this range indicates a severe fault in the system. This situation may arise due to invalid input values, program error, or hardware faults. The integrity of the program and hardware is in question, and immediate action is required.

In this case, the INT signal can be used to interrupt the program currently running. Such an interrupt would be of high priority. The interrupt handler responsible for numeric errors might perform system integrity tests and then restart the system at a known, safe state. The handler would not normally return to the point of error.

Unmasked numeric errors are very useful for testing programs. Correct use of synchronization, (Page 21), allows the programmer to find out exactly what operands, instruction, and memory values caused the error. Once testing has finished, an error then becomes much more serious.

The *8086 Family Numerics Supplement* recommends masking all errors except invalid. (See Lit. Ref.). In this case the NPX will safely handle such errors as underflow, overflow, or divide by zero. Only truly questionable operations will disturb the numerics program execution.

An example of how infinities and divide by zero can be harmless occurs when calculating the parallel resistance of several values with the standard formula (Figure 12). If R1 becomes zero, the circuit resistance becomes 0. With divide by zero and precision masked, the NPX will produce the correct result.

NUMERIC EXCEPTION HANDLING

For some applications, a numeric error may not indicate a severe problem. The numeric error can indicate that a hardware resource has been exhausted, and the software must provide more. These cases are called exceptions since they do not normally arise.

Special host software will handle numeric error exceptions when they infrequently occur. In these cases, numeric exceptions are expected to be recoverable although not requiring immediate service by the host. In effect, these exceptions extend the functionality of the NDP. Examples of extensions are: normalized only arithmetic, extending the register stack to memory, or tracing special data values.

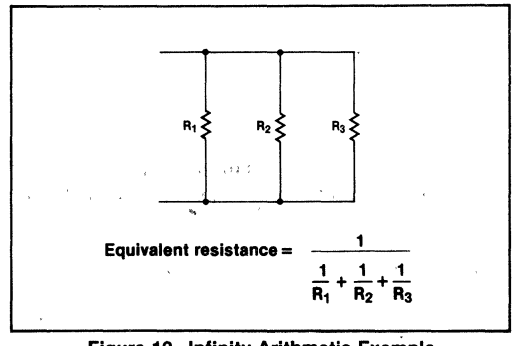


Figure 12. Infinity Arithmetic Example

HOST INTERRUPT OVERVIEW

The host has only two possible interrupt inputs, a non-maskable interrupt (NMI) and a maskable interrupt (INTR). Attaching the 8087 INT pin to the NMI input is not recommended. The following problems arise: NMI cannot be masked, it is usually reserved for more important functions like sanity timers or loss of power signal, and Intel supplied software for the NDP will not support NMI interrupts. The INTR input of the host allows interrupt masking in the CPU, using an Intel 8259A Programmable Interrupt Controller (PIC) to resolve multiple interrupts, and has Intel support.

NUMERIC INTERRUPT CHARACTERISTICS

Numeric error interrupts are different from regular instruction error interrupts like divide by zero. Numeric interrupts from the 8087 can occur long after the ESCAPE instruction that started the failing operation. For example, after starting a numeric multiply operation, the host may respond to an external interrupt and be in the process of servicing it when the 8087 detects an overflow error. In this case the interrupt is a result of some earlier, unrelated program.

From the point of view of the currently executing interrupt handler, numeric interrupts can come from only two sources: the current handler or a lower priority program.

To explicitly disable numeric interrupts, it is recommended that numeric interrupts be disabled at the 8087. The code example of Figure 13 shows how to disable any pending numeric interrupts then reenable them at the end of the handler. This code example can be safely placed in any routine which must prevent numeric interrupts from occurring. Note that the ESCAPE instructions act as NOPs if an 8087 is not present in the system. It is not recommended to use numeric mnemonics since they may be converted to emulator calls, which run comparatively slow, if the 8087 emulator used.

Interrupt systems have specific functions like fast response to external events or periodic execution of system routines. Adding an 8087 interrupt should not effect these functions. Desirable goals of any 8087 interrupt configuration are:

— Hide numeric interrupts from interrupt handlers that don't use the 8087. Since they didn't cause the numeric interrupt why should they be interrupted?

— Avoid adding code to interrupt handlers that don't use the 8087 to prevent interruption by the 8087.

— Allow other higher priority interrupts to be serviced while executing a numeric exception handler.

— Provide numeric exception handling for interrupt service routines which use the 8087.

— Avoid deadlock as described in a later section (page 24)

```
;
; Disable any possible numeric interrupt from the 8087. This code is safe to place in any
; procedure. If an 8087 is not present, the ESCAPE instructions will act as nops. These
; instructions are not affected by the TEST pin of the host. Using the 8087 emulator will not
; convert these instructions into interrupts. A word variable, called control, is required to hold
; the 8087 control word. Control must not be changed until it is reloaded into the 8087.
;
ESC  15, control          ; (FNSTCW) Save current 8087 control word
NOP                       ; Delay while 8087 saves current control
NOP                       ; register value
ESC  28,cx                ; (FNDISI) Disable any 8087 interrupts
                          ; Set IEM bit in 8087 control register
                          ; The contents of cx is irrelevant
                          ; Interrupts can now be enabled

                        (Your Code Here)

;
; Reenable any pending interrupts in the 8087. This instruction does not disturb any 8087 instruction
; currently in progress since all it does is change the IEM bit in the control register.
;
TEST control, 80H         ; Look at IEM bit
JNZ  $+4                  ; If IEM = 1 skip FNENI
ESC  28,ax                ; (FNENI) reenable 8087 interrupts
```

Figure 13. Inhibit/Enable 8087 Interrupts

Recommended Interrupt Configurations

Five categories cover most uses of the 8087 interrupt in fixed priority interrupt systems. For each category, an interrupt configuration is suggested based on the goals mentioned above.

1. All errors on the 8087 are always masked. Numeric interrupts are not possible. Leave the 8087 INT signal unconnected.
2. The 8087 is the only interrupt in the system. Connect the 8087 INT signal directly to the host's INTR input. (See Figure 14 on page19). A bus driver supplies interrupt vector 10_{16} for compatibility with Intel supplied software.
3. The 8087 interrupt is a stop everything event. Choose a high priority interrupt input that will terminate all numerics related activity. This is a special case since the interrupt handler may never return to the point of interruption (i.e. reset the system and restart rather than attempt to continue operation).
4. Numeric exceptions or numeric programming errors are expected and all interrupt handlers either don't use the 8087 or only use it with all errors masked. Use the lowest priority interrupt input. The 8087 interrupt handler should allow further interrupts by higher priority events. The PIC's priority system will automatically prevent the 8087 from disturbing other interrupts without adding extra code to them.
5. Case 4 holds except that interrupt handlers may also generate numeric interrupts. Connect the 8087 INT signal to multiple interrupt inputs. One input would still be the lowest priority input as in case 4. Interrupt handlers that may generate a numeric interrupt will require another 8087 INT connection to the next highest priority interrupt. Normally the higher priority numeric interrupt inputs would be masked and the low priority numeric interrupt enabled. The higher priority interrupt input would be unmasked only when servicing an interrupt which requires 8087 exception handling.

All of these configurations hide the 8087 from all interrupt handlers which do not use the 8087. Only those interrupt handlers that use the 8087 are required to perform any special 8087 related interrupt control activities.

A conflict can arise between the desired PIC interrupt input and the required interrupt vector of 10_{16} for compatibility with Intel software for numeric interrupts. A simple solution is to use more than one interrupt vector for numeric interrupts, all pointing at the same 8087 interrupt handler. Design the numeric interrupt handler such that it need not know what the interrupt vector was (i.e. don't use specific EOI commands).

If an interrupt system uses rotating interrupt priorities, it will not matter which interrupt input is used.

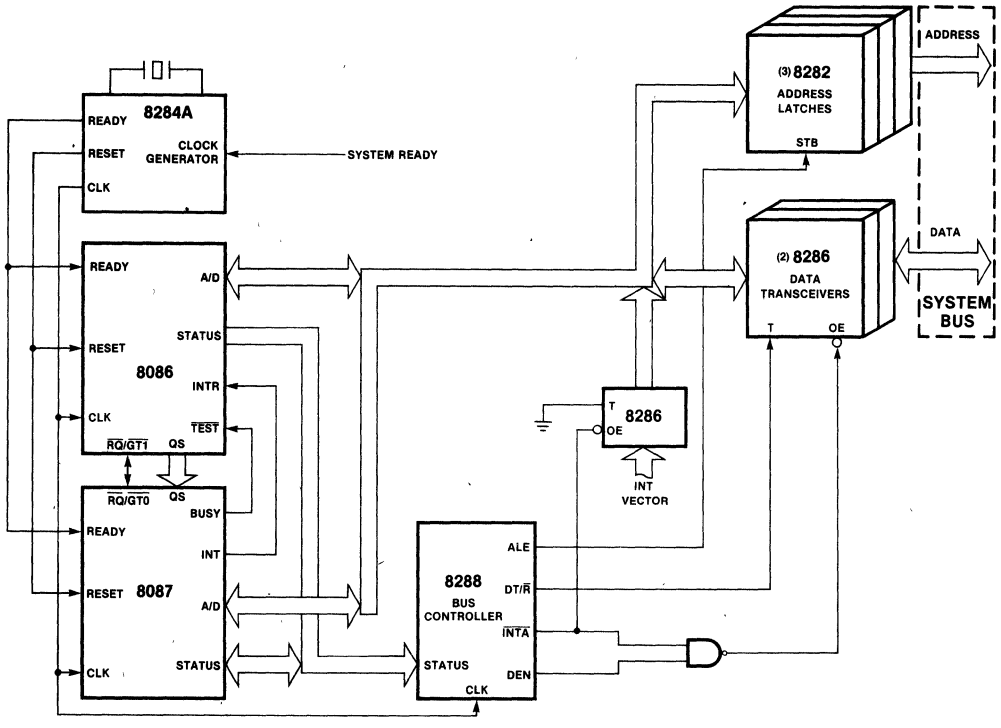


Figure 14. iAPX 86/20 With Numerics Interrupt Only

GETTING STARTED IN SOFTWARE

Now we are ready to run numeric programs. Developing numeric software will be a new experience to some programmers. This section of the application note is aimed at describing the programming environment and providing programming guidelines for the NPX. The term NPX is used to emphasize that no distinction is made between the 8087 component or an emulated 8087.

Two major areas of numeric software can be identified: systems software and applications software. Products such as iRMX™ 86 provide system software as an off-the-shelf product. Some applications use specially developed systems software optimized to their needs.

Whether the system software is specially tailored or common, they share issues such as using concurrency, maintaining synchronization between the host and 8087, and establishing programming conventions. Applications software directly performs the functions of the application. All applications will be concerned with initialization and general programming rules for the NPX. Systems software will be more concerned with context switching, use of the NPX by interrupt handlers, and numeric exception handlers.

How to Initialize the NPX

The first action required by the NPX is initialization. This places the NPX in a known state, unaffected by other activity performed earlier. This initialization is similar to that caused by the RESET signal of the 8087. All the error masks are set, all registers are tagged empty, the TOP field is set to 0, default rounding, precision, and infinity controls are set. The 8087 emulator requires more initialization than the component. Before the emulator may be used, all its interrupt vectors must be set to point to the correct entry points within the emulator.

To provide compatibility between the emulator and component in this special case, a call to an external procedure should be used before the first numeric instruction. In ASM86 the programmer must call the external function INIT87. (Fig. 15). For PLM86, the programmer must call the built-in function INIT$REAL$MATH$UNIT. PLM86 will call INIT87 when executing the INIT$REAL$MATH$UNIT built-in function.

The function supplied for INIT87 will be different, depending on whether the emulator library, called E8087.LIB, or component library, called 8087.LIB, were used at link time. INIT87 will execute either an FNINIT instruction for the 8087 or initialize the 8087 emulator interrupt vectors, as appropriate.

Concurrency Overview

With the NPX initialized, the next step in writing a numeric program is learning about concurrent execution within the NDP.

Concurrency is a special feature of the 8087, allowing it and the host to simultaneously execute different instructions. The 8087 emulator does not provide concurrency since it is implemented by the host.

The benefit of concurrency to an application is higher performance. All Intel high level languages automatically provide for and manage concurrency in the NDP. However, in exchange for the added performance, the assembly language programmer must understand and manage some areas of concurrency. This section is for the assembly language programmer or well-informed, high level language programmer.

Whether the 8087 emulator or component is used, care should be taken by the assembly language programmer to follow the rules described below regarding synchronization. Otherwise, the program may not function correctly with current or future alternatives for implementing the NDP.

Concurrency is possible in the NDP because both the host and 8087 have separate arithmetic and control units. The host and coprocessor automatically decide who will perform any single instruction. The existence of the 8087 as a separate unit is not normally apparent.

Numeric instructions, which will be executed by the 8087, are simply placed in line with the instructions for the host. Numeric instructions are executed in the same order as they are encountered by the host in its instruction stream. Since operations performed by the 8087 generally require more time than operations performed by the host, the host can execute several of its instructions while the 8087 performs one numeric operation.

```
IN PLM86:
    CALL INIT$REAL$MATH$UNIT;

IN ASM86:
    EXTRN          INIT87:FAR
      •
      •
      •
      •
    CALL           INIT87
```

Figure 15. 8087 Initialization

MANAGING CONCURRENCY

Concurrent execution of the host and 8087 is easy to establish and maintain. The activities of numeric programs can be split into two major areas: program control and arithmetic. The program control part performs activities like deciding what functions to perform, calculating addresses of numeric operands, and loop control. The arithmetic part simply performs the adds, subtracts, multiplies, and other operations on the numeric operands. The NPX and host are designed to handle these two parts separately and efficiently.

Managing concurrency is necessary because the arithmetic and control areas must converge to a well-defined state when starting another numeric operation. A well-defined state means all previous arithmetic and control operations are complete and valid.

Normally, the host waits for the 8087 to finish the current numeric operation before starting another. This waiting is called synchronization.

Managing concurrent execution of the 8087 involves three types of synchronization: instruction, data, and error. Instruction and error synchronization are automatically provided by the compiler or assembler. Data synchronization must be provided by the assembly language progammer or compiler.

Instruction Synchronization

Instruction synchronization is required because the 8087 can only perform one numeric operation at a time. Before any numeric operation is started, the 8087 must have completed all activity from previous instructions.

The WAIT instruction on the host lets it wait for the 8087 to finish all numeric activity before starting another numeric instruction. The assembler automatically provides for instruction synchronization since a WAIT instruction is part of most numeric instructions. A WAIT instruction requires 1 byte code space and 2.5 clocks average execution time overhead.

Instruction synchronization as provided by the assembler or a compiler allows concurrent operation in the NDP. An execution time comparison of NDP concurrency and non-concurrency is illustrated in Figure 16. The non-concurrent program places a WAIT instruction immediately after a multiply instruction ESCAPE instruction. The 8087 must complete the multiply operation before the host executes the MOV instruction on statement 2. In contrast, the concurrent example allows the host to calculate the effective address of the next operand while the 8087 performs the multiply. The execution time of the concurrent technique is the longest of the host's execution time from line 2 to 5 and the execution time of the 8087 for a multiply instruction. The execution time of the non-concurrent example is the sum of the execution times of statements 1 to 5.

```
;
; This code macro defines two instructions which do not allow any concurrency of execution with
; the host. A register version and memory version of the instruction is shown. It is assumed that the
; 8087 is always idle from the previous instruction. Allow space for emulator fixups.
;
R233 Record RF6:2, Mid3:3, RF7:3

CodeMacro NCMUL dst:T, src:F
RNfix 000B
R233 (11B, 001B, src)
RWfix
EndM

CodeMacro NCMUL memop:Mq
RNfixM 100B, memop
ModRM 001B, memop
RWfix
EndM
```

Statement	Concurrent		Non Concurrent	
1	FMUL	st(0), st(1)	NCMUL	st(0), st(1)
2	MOV	ax, size A	MOV	ax, size A
3	MUL	index	MUL	index
4	MOV	bx, ax	MOV	bx, ax
5	FMUL	A [bx]	NCMUL	A [bx]

Figure 16. Concurrent Versus Non-Concurrent Program

Data Synchronization

Managing concurrency requires synchronizing data references by the host and 8087.

Figure 17 shows four possible cases of the host and 8087 sharing a memory value. The second two cases require the FWAIT instruction shown for data synchronization. In the first two cases, the host will finish with the operand I before the 8087 can reference it. The coprocessor interface guarantees this. In the second two cases, the host must wait for the 8087 to finish with the memory operand before proceeding to reuse it. The FWAIT instruction in case 3 forces the host to wait for the 8087 to read I before changing it. In case 4, the FWAIT prevents the host from reading I before the 8087 sets its value.

Obviously, the programmer must recognize any form of the two cases shown above which require explicit data synchronization. Data synchronization is not a concern when the host and 8087 are using different memory operands during the course of one numeric instruction. Figure 16 shows such an example of the host performing activity unrelated to the current numeric instruction being executed by the 8087. Correct recognition of these cases by the programmer is the price to be paid for providing concurrency at the assembly language level.

Automatic Data Synchronization

Two methods exist to avoid the need for manual recognition of when data synchronization is needed: use a high level language which will automatically establish concurrency and manage it, or sacrifice some performance for automatic data synchronization by the assembler.

When a high level language is not adequate, the assembler can be changed to always place a WAIT instruction after the ESCAPE instruction. Figure 18 shows an example of how to change the ASM86 code macro for the FIST instruction to automatically place an FWAIT instruction after the ESCAPE instruction. The lack of any possible concurrent execution between the host and 8087 while the FIST instruction is executing is the price paid for automatic data synchronization.

An explicit FWAIT instruction for data synchronization, can be eliminated by using a subsequent numeric instruction. After this subsequent instruction has started execution, all memory references in earlier numeric instructions are complete. Reaching the next host instruction after the synchronizing numeric instruction indicates previous numeric operands in memory are available.

The data synchronization purpose of any FWAIT or numeric instruction must be well documented. Otherwise, a change to the program at a later time may remove the synchronizing numeric instruction, causing program failure, as:

```
FISTP   I
FMUL
MOV     AX, I     ; I is safe to use
```

```
Case 1:                 Case 3:
    MOV   I, 1              FILD    I
    FILD  I                 FWAIT
                            MOV     I,5

Case 2:                 Case 4:
    MOV   AX, I             FISTP   I
    FISTP I                 FWAIT
                            MOV     AX,I
```

Figure 17. Data Exchange Example

```
;
; This is a code macro to redefine the FIST
; instruction to prevent any concurrency
; while the instruction runs. A wait
; instruction is placed immediately after the
; escape to ensure the store is done
; before the program may continue. This
; code macro will work with the 8087
; emulator, automatically replacing the
; wait escape with a nop.
;
CodeMacro FIST memop: Mw
RfixM 111B, memop
ModRM 010B, memop
RWfix
EndM
```

Figure 18. Non-Concurrent FIST Instruction Code Macro

DATA SYNCHRONIZATION RULES EXCEPTIONS

There are five exceptions to the above rules for data synchronization. The 8087 automatically provides data synchronization for these cases. They are necessary to avoid deadlock (described on page 24). The instructions FSTSW/FNSTSW, FSTCW/FNSTCW, FLDCW, FRSTOR, and FLDENV do not require any waiting by the host before it may read or modify the referenced memory location.

The 8087 provides the data synchronization by preventing the host from gaining control of the local bus while these instructions execute. If the host cannot gain control of the local bus, it cannot change a value before the 8087 reads it, or read a value before the 8087 writes into it.

The coprocessor interface guarantees that, when the host executes one of these instructions, the 8087 will immediately request the local bus from the host. This request is timed such that, when the host finishes the read operation identifying the memory operand, it will always grant the local bus to the 8087 before the host may use the local bus for a data reference while executing a subsequent instruction. The 8087 will not release the local bus to the host until it has finished executing the numeric instruction.

Error Synchronization

Numeric errors can occur on almost any numeric instruction at any time during its execution. Page 15 describes how a numeric error may have many interpretations, depending on the application. Since the response to a numeric error will depend on the application, this section covers topics common to all uses of the NPX. We will review why error synchronization is needed and how it is provided.

Concurrent execution of the host and 8087 requires synchronization for errors just like data references and numeric instructions. In fact, the synchronization required for data and instructions automatically provides error synchronization.

However, incorrect data or instruction synchronization may not cause a problem until a numeric error occurs. A further complication is that a programmer may not expect his numeric program to cause numeric errors, but in some systems they may regularly happen. To better understand these points, let's look at what can happen when the NPX detects an error.

ERROR SYNCHRONIZATION FOR EXTENSIONS

The NPX can provide a default fixup for all numeric errors. A program can mask each individual error type to indicate that the NPX should generate a safe, reasonable result. The default error fixup activity is simply treated as part of the instruction which caused the error. No external indication of the error will be given. A flag in the numeric status register will be set to indicate that an error was detected, but no information regarding where or when will be available.

If the NPX performs its default action for all errors, then error synchronization is never exercised. But this is no reason to ignore error synchronization.

Another alternative exists to the NPX default fixup of an error. If the default NPX response to numeric errors is not desired, the host can implement any form of recovery desired for any numeric error detectable by the NPX. When a numeric error is unmasked, and the error occurs, the NPX will stop further execution of the numeric instruction. The 8087 will signal this event on the INT pin, while the 8087 emulator will cause interrupt 10_{16} to occur. The 8087 INT signal is normally connected to the host's interrupt system. Refer to page 18 for further discussion on wiring the 8087 INT pin.

Interrupting the host is a request from the NPX for help. The fact that the error was unmasked indicates that further numeric program execution under the arithmetic and programming rules of the NPX is unreasonable. Error synchronization serves to insure the NDP is in a well defined state after an unmasked numeric error occured. Without a well defined state, it is impossible to figure out why the error occured.

Allowing a correct analysis of the error is the heart of error synchronization.

NDP ERROR STATES

If concurrent execution is allowed, the state of the host when it recognizes the interrupt is undefined. The host may have changed many of its internal registers and be executing a totally different program by the time it is interrupted. To handle this situation, the NPX has special registers updated at the start of each numeric instruction to describe the state of the numeric program when the failed instruction was attempted. (See Lit. Ref. p. iii)

Besides programmer comfort, a well-defined state is important for error recovery routines. They can change the arithmetic and programming rules of the 8087. These changes may redefine the default fixup from an error, change the appearance of the NPX to the programmer, or change how arithmetic is defined on the NPX.

EXTENSION EXAMPLES

A change to an error response might be to automatically normalize all denormals loaded from memory. A change in appearance might be extending the register stack to memory to provide an "infinite" number of numeric registers. The arithmetic of the 8087 can be changed to automatically extend the precision and range of variables when exceeded. All these functions can be implemented on the NPX via numeric errors and associated recovery routines in a manner transparent to the programmer.

Without correct error synchronization, numeric subroutines will not work correctly in the above situations.

Incorrect Error Synchronization

An example of how some instructions written without error synchronization will work initially, but fail when moved into a new environment is:

```
FILD    COUNT
INC     COUNT
FSQRT
```

Three instructions are shown to load an integer, calculate its square root, then increment the integer. The coprocessor interface of the 8087 and synchronous execution of the 8087 emulator will allow this program to execute correctly when no errors occur on the FILD instruction.

But, this situation changes if the numeric register stack is extended to memory on an 8087. To extend the NPX stack to memory, the invalid error is unmasked. A push to a full register or pop from an empty register will cause an invalid error. The recovery routine for the error must recognize this situation, fixup the stack, then perform the original operation.

The recovery routine will not work correctly in the example. The problem is that there is no guarantee that COUNT will not be incremented before the 8087 can interrupt the host. If COUNT is incremented before the interrupt, the recovery routine will load a value of COUNT one too large, probably causing the program to fail.

Error Synchronization and WAITs

Error synchronization relies on the WAIT instructions required by instruction and data synchronization and the INT and BUSY signals of the 8087. When an unmasked error occurs in the 8087, it asserts the BUSY and INT signals. The INT signal is to interrupt the host, while the BUSY signal prevents the host from destroying the current numeric context.

The BUSY signal will never go inactive during a numeric instruction which asserts INT.

The WAIT instructions supplied for instruction synchronization prevent the host from starting another numeric instruction until the current error is serviced. In a like manner, the WAIT instructions required for data synchronization prevent the host from prematurely reading a value not yet stored by the 8087, or overwriting a value not yet read by the 8087.

The host has two responsibilities when handling numeric errors.1.) It must not disturb the numeric context when an error is detected, and 2.) it must clear the numeric error and attempt recovery from the error. The recovery program invoked by the numeric error may resume program execution after proper fixup, display the state of the NDP for programmer action, or simply abort the program. In any case, the host must do something with the 8087. With the INT and BUSY signals active, the 8087 cannot perform any useful work. Special instructions exist for controlling the 8087 when in this state. Later, an example is given of how to save the state of the NPX with an error pending. (See page 29)

Deadlock

An undesirable situation may result if the host cannot be interrupted by the 8087 when asserting INT. This situation, called deadlock, occurs if the interrupt path from the 8087 to the host is broken.

The 8087 BUSY signal prevents the host from executing further instructions (for instruction or data synchronization) while the 8087 waits for the host to service the exception. The host is waiting for the 8087 to finish the current numeric operation. Both the host and 8087 are waiting on each other. This situation is stable unless the host is interrupted by some other event.

Deadlock has varying affects on the NDP's performance. If no other interrupts in the system are possible, the NDP will wait forever. If other interrupts can arise, then the NDP can perform other functions, but the affected numeric program will remain "frozen".

SOLVING DEADLOCK

Finding the break in the interrupt path is simple. Look for disabled interrupts in the following places: masked interrupt enable in the host, explicitly masked interrupt request in the interrupt controller, implicitly masked interrupt request in the interrupt controller due to a higher priority interrupt in service, or other gate functions, usually in TTL, on the host interrupt signal.

DEADLOCK AVOIDANCE

Application programmers should not be concerned with deadlock. Normally, applications programs run with unmasked numeric errors able to interrupt them. Deadlock is not possible in this case. Traditionally, systems software or interrupt handlers may run with numeric interrupts disabled. Deadlock prevention lies in this domain. The golden rule to abide by is: "Never wait on the 8087 if an unmasked error is possible and the 8087 interrupt path may be broken."

Error Synchronization Summary

In summary, error synchronization involves protecting the state of the 8087 after an exception. Although not all applications may initially require error synchronization, it is just good programming practice to follow the rules. The advantage of being a "good" numerics programmer is generality of your program so it can work in other, more general environments.

Summary

Synchronization is the price for concurrency in the NDP. Intel high level language compilers will automatically provide concurrency and manage it with synchronization. The assembly language programmer can choose between using concurrency or not. Placing a WAIT instruction immediately after any numeric instruction will prevent concurrency and avoid synchronization concerns.

The rules given above are complete and allow concurrency to be used to full advantage.

Synchronization and the Emulator

The above discussion on synchronization takes on special meaning with the 8087 emulator. The 8087 emulator does not allow any concurrency. All numeric operand memory references, error tests, and wait for instruction completion occur within the emulator. As a result, programs which do not provide proper instruction, data, or error synchronization may work with the 8087 emulator while failing on the component.

Correct programs for the 8087 work correctly on the emulator.

Special Control Instructions of the NPX

The special control instructions of the NPX: FNINIT, FNSAVE, FNSTENV, FRSTOR, FLDENV, FLDCW, FNSTSW, FNSTCW, FNCLEX, FNENI, and FNDISI remove some of the synchronization requirements mentioned earlier. They are discussed here since they represent exceptions to the rules mentioned on page 21.

The instructions FNINIT, FNSAVE, FNSTENV, FNSTSW, FNCLEX, FNENI, and FNDISI do not wait for the current numeric instruction to finish before they execute. Of these instructions, FNINIT, FNSTSW, FNCLEX, FNENI and FNDISI will produce different results, depending on when they are executed relative to the current numeric instruction.

For example, FNCLEX will cause a different status value to result from a concurrent arithmetic operation, depending on whether is is executed before or after the error status bits are updated at the end of the arithmetic operation. The intended use of FNCLEX is to clear a known error status bit which has caused BUSY to be asserted, avoiding deadlock.

FNSTSW will safely, without deadlock, report the busy and error status of the NPX independent of the NDP interrupt status.

FNINIT, FNENI, and FNDISI are used to place the NPX into a known state independent of its current state. FNDISI will prevent an unmasked error from asserting BUSY without disturbing the current error status bits. Appendix A shows an example of using FNDISI.

The instructions FNSAVE and FNSTENV provide special functions. They allow saving the state of the NPX in a single instruction when host interrupts are disabled.

Several host and numeric instructions are necessary to save the NPX status if the interrupt status of the host is unknown. Appendix A and B show examples of saving the NPX state. As the *Numerics Supplement* explains, host interrupts must always be disabled when executing FNSAVE or FNSTENV.

The seven instructions FSTSW/FNSTSW, FSTCW/FNSTCW, FLDCW, FLDENV, and FRSTOR do not require explicit WAIT instructions for data synchronization. All of these instructions are used to interrogate or control the numeric context.

Data synchronization for these instructions is automatically provided by the coprocessor interface. The 8087 will take exclusive control of the memory bus, preventing the host from interfering with the data values before the 8087 can read them. Eliminating the need for a WAIT instruction avoids potential deadlock problems.

The three load instructions FLDCW, FLDENV, and FRSTOR can unmask a numeric error, activating the 8087 BUSY signal. Such an error was the result of a previous numeric instruction and is not related to any fault in the instruction.

Data synchronization is automatically provided since the host's interrupts are usually disabled in context switching or interrupt handling, deadlock might result if the host executed a WAIT instruction with its interrupts disabled after these instructions. After the host interrupts are enabled, an interrupt will occur if an unmasked error was pending.

PROGRAMMING TECHNIQUES

The NPX provides a stack-oriented register set with stack-oriented instructions for numeric operands. These registers and instructions are optimized for numeric programs. For many programmers, these are new resources with new programming options available.

Using Numeric Registers and Instructions

The register and instruction set of the NDP is optimized for the needs of numeric and general purpose programs. The host CPU provides the instructions and data types needed for general purpose data processing, while the 8087 provides the data types and instructions for numeric processing.

The instructions and data types recognized by the 8087 are different from the CPU because numeric program requirements are different from those of general purpose programs. Numeric programs have long arithmetic expressions where a few temporary values are used in a few statements. Within these statements, a single value may be referenced many times. Due to the time involved to transfer values between registers and memory, a significant speed optimization is possible by keeping numbers in the NPX register file.

In contrast, a general data processor is more concerned with addressing data in simple expressions and testing the results. Temporary values, constant across several instructions, are not as common nor is the penalty as large for placing them in memory.As a result it is simpler for compilers and programmers to manage memory based values.

NPX Register Usage

The eight numeric registers in the NDP are stack oriented. All numeric registers are addressed relative to a value called the TOP pointer, defined in the NDP status register. A register address given in an instruction is added to the TOP value to form the internal absolute address. Relative addressing of numeric registers has advantages analogous to those of relative addressing of memory operands.

Two modes are available for addressing the numeric registers. The first mode implicitly uses the top and optional next element on the stack for operands. This mode does not require any addressing bits in a numeric instruction. Special purpose instructions use this mode since full addressing flexibility is not required.

The other addressing mode allows any other stack element to be used together with the top of stack register. The top of stack or the other register may be specified as the destination. Most two-operand arithmetic instructions allow this addressing mode. Short, easy to develop numeric programs are the result.

Just as relative addressing of memory operands avoids concerns with memory allocation in other parts of a program, top relative register addressing allows registers to be used without regard for numeric register assignments in other parts of the program.

STACK RELATIVE ADDRESSING EXAMPLE

Consider an example of a main program calling a subroutine, each using register addressing independent of the other. (Fig. 19) By using different values of the TOP field, different software can use the same relative register addresses as other parts of the program, but refer to different physical registers.

```
MAIN_PROGRAM:
  FLD       A
  FADD      ST, ST(1)
  CALL      SUBROUTINE      ; Argument is in ST(0)
  FSTP      B

SUBROUTINE:
  FLD       ST              ; ST(0) = ST(1) = Argument
  FSQRT                     ; Main program ST(1) is
  FADD      C               ; safe in ST(2) here
  FMULP     ST(1), ST
  RET
```

Figure 19. Stack Relative Addressing Example

Of course, there is a limit to any physical resource. The NDP has eight numeric registers. Normally, programmers must ensure a maximum of eight values are pushed on the numeric register stack at any time. For time-critical inner loops of real-time applications, eight registers should contain all the values needed.

REGISTER STACK EXTENSION

This hardware limitation can be hidden by software. Software can provide "virtual" numeric registers, expanding the register stack size to 6000 or more.

The numeric register stack can be extended into memory via unmasked numeric invalid errors which cause an interrupt on stack overflow or underflow. The interrupt handler for the invalid error would manage a memory image of the numeric stack copying values into and out of memory as needed.

The NPX will contain all the necessary information to identify the error, failing instruction, required registers, and destination register. After correcting for the missing hardware resource, the original numeric operation could be repeated. Either the original numeric instruction could be single stepped or the affect of the instruction emulated by a composite of table-based numeric instructions executed by the error handler.

With proper data, error, and instruction synchronization, the activity of the error handler will be transparent to programs. This type of extension to the NDP allows programs to push and pop numeric registers without regard for their usage by other subroutines.

Programming Conventions

With a better understanding of the stack registers, let's consider some useful programming conventions. Following these conventions ensures compatibility with Intel support software and high level language calling conventions.

1) If the numeric registers are not extended to memory, the programmer must ensure that the number of temporary values left in the NPX stack and those registers used by the caller does not exceed 8. Values can be stored to memory to provide enough free NPX registers.
2) Pass the first seven numeric parameters to a subroutine in the numeric stack registers. Any extra parameters can be passed on the host's stack. Push the values on the register or memory stack in left to right order. If the subroutine does not need to allocate any more numeric registers, it can execute solely out of the numeric register stack. The eighth register can be used for arithmetic operations. All parameters should be popped off when the subroutine completes.
3) Return all numeric values on the numeric stack. The caller may now take advantage of the extended precision and flexible store modes of the NDP.
4) Finish all memory reads or writes by the NPX before exiting any subroutine. This guarantees correct data and error synchronization. A numeric operation based solely on register contents is safe to leave running on subroutine exit.
5) The operating mode of the NDP should be transparent across any subroutine. The operating mode is defined by the control word of the NDP. If the subroutine needs to use a different numeric operating mode than that of the caller, the subroutine should first save the current control word, set the new operating mode, then restore the original control word when completed.

PROGRAMMING EXAMPLES

The last section of this application note will discuss five programming examples. These examples were picked to illustrate NDP programming techniques and commonly used functions. All have been coded, assembled, and tested. However, no guarantees are made regarding their correctness.

The programming examples are: saving numeric context switching, save numeric context without FSAVE/FNSAVE, converting ASCII to floating point, converting floating point to ASCII, and trigonometric functions. Each example is listed in a different appendix with a detailed written description in the following text. The source code is available in machine readable form from the Intel Insite User's Library, "Interactive 8087 Instruction Interpreter," catalog item AA20.

The examples provide some basic functions needed to get started with the numeric data processor. They work with either the 8087 or the 8087 emulator with no source changes.

The context switching examples are needed for operating systems or interrupt handlers which may use numeric instructions and operands. Converting between floating point and decimal ASCII will be needed to input or output numbers in easy to read form. The trigonometric examples help you get started with sine or cosine functions and can serve as a basis for optimizations if the angle arguments always fall into a restricted range.

APPENDIX A

OVERVIEW

Appendix A shows deadlock-free examples of numeric context switching. Numeric context switching is required by interrupt handlers which use the NPX and operating system context switchers. Context switching consists of two basic functions, save the numeric context and restore it. These functions must work independent of the current state of the NPX.

Two versions of the context save function are shown. They use different versions of the save context instruction. The FNSAVE/FSAVE instructions do all the work of saving the numeric context. The state of host interrupts will decide which instruction to use.

Using FNSAVE

The FNSAVE instruction is intended to save the NPX context when host interrupts are disabled. The host does not have to wait for the 8087 to finish its current operation before starting this operation. Eliminating the instruction synchronization wait avoids any potential deadlock.

The 8087 Bus Interface Unit (BIU) will save this instruction when encountered by the host and hold it until the 8087 Floating point Execution Unit (FEU) finishes its current operation. When the FEU becomes idle, the BIU will start the FEU executing the save context operation.

The host can execute other non-numeric instructions after the FNSAVE while the BIU waits for the FEU to finish its current operation. The code starting at NO_INT_NPX_SAVE shows how to use the FNSAVE instruction.

When executing the FNSAVE instruction, host interrupts must be disabled to avoid recursions of the instruction. The 8087 BIU can hold only one FNSAVE instruction at a time. If host interrupts were not disabled, another host interrupt might cause a second FNSAVE instruction to be executed, destroying the previous one saved in the 8087 BIU.

It is not recommended to explicitly disable host interrupts just to execute an FNSAVE instruction. In general, such an operation may not be the best course of action or even be allowed.

If host interrupts are enabled during the NPX context save function, it is recommended to use the FSAVE instruction as shown by the code starting at NPX_SAVE. This example will always work, free of deadlock, independent of the NDP interrupt state.

Using FSAVE

The FSAVE instruction performs the same operation as FNSAVE but it uses standard instruction synchronization. The host will wait for the FEU to be idle before initiating the save operation. Since the host ignores all interrupts between completing a WAIT instruction and starting the following ESCAPE instruction, the FEU is ready to immediately accept the operation (since it is not signalling BUSY). No recursion of the save context operation in the BIU is possible. However, deadlock must be considered since the host executes a WAIT instruction.

To avoid deadlock when using the FSAVE instruction, the 8087 must be prevented from signalling BUSY when an unmasked error exists.

The Interrupt Enable Mask (IEM) bit in the NPX control word provides this function. When IEM = 1, the 8087 will not signal BUSY or INT if an unmasked error exists. The NPX instruction FNDISI will set the IEM independent of any pending errors without causing deadlock or any other errors. Using the FNDISI and FSAVE instructions together with a few other glue instructions allows a general NPX context save function.

Standard data and instruction synchronization is required after executing the FNSAVE/FSAVE instruction. The wait instruction following an FNSAVE/FSAVE instruction is always safe since all NPX errors will be masked as part of the instruction execution. Deadlock is not possible since the 8087 will eventually signal not busy, allowing the host to continue on.

PLACING THE SAVE CONTEXT FUNCTION

Deciding on where to save the NPX context in an interrupt handler or context switcher is dependent on whether interrupts can be enabled inside the function. Since interrupt latency is measured in terms of the maximum time interrupts are disabled, the maximum wait time of the host at the data synchronizing wait instruction after the FNSAVE or the FSAVE instruction is important if host interrupts are disabled while waiting.

The wait time will be the maximum single instruction execution time of the 8087 plus the execution time of the save operation. This maximum time will be approximately 1300 or 1500 clocks, depending on whether the host is an 8086 or 8088, respectively. The actual time will depend on how much concurrency of execution between the host and 8087 is provided. The greater the concurrency, the lesser the maximum wait time will be.

If host interrupts can be enabled during the context save function, it is recommended to use the FSAVE instruction for saving the numeric context in the interruptable section. The FSAVE instruction allows instruction and data synchronizing waits to be interruptable. This technique removes the maximum execution time of 8087 instructions from system interrupt latency time considerations.

It is recommended to delay starting the numeric save function as long as possible to maintain the maximum amount of concurrent execution between the host and the 8087.

Using FRSTOR

Restoring the numeric context with FRSTOR does not require a data synchronizing wait afterwards since the 8087 automatically prevents the host from interfering with the memory load operation.

The code starting with NPX_RESTORE illustrates the restore operation. Error synchronization is not necessary since the FRSTOR instruction itself does not cause errors, but the previous state of the NPX may indicate an error.

If further numeric instructions are executed after the FRSTOR, and the error state of the new NPX context is unknown, deadlock may occur if numeric exceptions cannot interrupt the host.

NPX_save

```
;
; General purpose save of NPX context. This function will work independent of the interrupt state of
; the NDP. Deadlock can not occur. 47 words of memory are required by the variable save_area.
; Register ax is not transparent across this code.
;
NPX_save:
    FNSTCW    save_area        ; Save IEM bit status
    NOP                        ; Delay while 8087 saves control register
    FNDISI                     ; Disable 8087 BUSY signal
    MOV       ax, save_area    ; Get original control word
    FSAVE     save_area        ; Save NPX context, the host can be safely interrupted while
                               ; waiting for the 8087 to finish. Deadlock is not possible since
    FWAIT                      ; IEM = 1.Wait for save to finish. Put original control word into
    MOV       save_area, ax    ; NPX context area. All done
```

no_int_NPX_save

```
;
; Save the NPX context with host interrupts disabled. No deadlock is possible. 47 words of memory
; are required by the variable save_area.
;
no_int_NPX_save:
    FNSAVE    save_area        ; Save NPX context. Wait for save to finish, no deadlock
    FWAIT                      ; is possible. Interrupts may be enabled now, all done
```

NPX_restore

```
;
; Restore the NPX context saved earlier. No deadlock is possible if no further numeric instructions
; are executed until the 8087 numeric error interrupt is enabled. The variable save_area is assumed
; to hold an NPX context saved earlier. It must be 47 words long.
;
NPX_restore:
    FRSTOR    save_area        ; Load new NPX context
```

APPENDIX B

OVERVIEW

Appendix B shows alternative techniques for switching the numeric context without using the FSAVE/FNSAVE or FRSTOR instructions. These alternative techniques are slower than those of Appendix A but they reduce the worst case continuous local bus usage of the 8087.

Only an iAPX 86/22 or iAPX 88/22 could derive any benefit from this alternative. By replacing all FSAVE/FNSAVE instructions in the system, the worst case local bus usage of the 8087 will be 10 or 16 consecutive memory cycles for an 8086 or 8088 host, respectively.

Instead of saving and loading the entire numeric context in one long series of memory transfers, these routines use the FSTENV/FNSTENV/FLDENV instructions and separate numeric register load/store instructions. Using separate load/store instructions for the numeric registers forces the 8087 to release the local bus after each numeric load/store instruction. The longest series of back-to-back memory transfers required by these instructions are 8/12 memory cycles for an 8086 or 8088 host, respectively. In contrast, the FSAVE/FNSAVE/FRSTOR instructions perform 50/94 back-to-back memory cycles for an 8086 or 8088 host.

Compatibility With FSAVE/FNSAVE

This function produces a context area of the same format produced by FSAVE/FNSAVE instructions. Other software modules expecting such a format will not be affected. All the same interrupt and deadlock considerations of FSAVE and FNSAVE also apply to FSTENV and FNSTENV. Except for the fact that the numeric environment is 7 words rather than the 47 words of the numeric context, all the discussion of Appendix A also applies here.

The state of the NPX registers must be saved in memory in the same format as the FSAVE/FNSAVE instructions. The program example starting at the label SMALL_BLOCK_NPX_SAVE illustrates a software loop that will store their contents into memory in the same top relative order as that of FSAVE/FNSAVE.

To save the registers with FSTP instructions, they must be tagged valid, zero, or special. This function will force all the registers to be tagged valid, independent of their contents or old tag, and then save them. No problems will arise if the tag value conflicts with the register's content for the FSTP instruction. Saving empty registers insures compatibility with the FSAVE/FNSAVE instructions. After saving all the numeric registers, they will all be tagged empty, the same as if an FSAVE/FNSAVE instruction had been executed.

Compatibility With FRSTOR

Restoring the numeric context reverses the procedure described above, as shown by the code starting at SMALL_BLOCK_NPX_RESTORE. All eight registers are reloaded in the reverse order. With each register load, a tag value will be assigned to each register. The tags assigned by the register load does not matter since the tag word will be overwritten when the environment is reloaded later with FLDENV.

Two assumptions are required for correct operation of the restore function: all numeric registers must be empty and the TOP field must be the same as that in the context being restored. These assumptions will be satisfied if a matched set of pushes and pops were performed between saving the numeric context and reloading it.

If these assumptions cannot be met, then the code example starting at NPX_CLEAN shows how to force all the NPX registers empty and set the TOP field of the status word.

small_block_NPX_save

```
;
; Save the NPX context independent of NDP interrupt state. Avoid using the FSAVE instruction to
; limit the worst case memory bus usage of the 8087. The NPX context area formed will appear the
; same as if an FSAVE instruction had written into it. The variable save_area will hold the NPX
; context and must be 47 words long. The registers ax, bx, and cx will not be transparent.
;
small_block_NPX_save:
    FNSTCW  save_area          ; Save current IEM bit
    NOP                        ; Delay while 8087 saves control register
    FNDISI                     ; Disable 8087 BUSY signal
    MOV     ax, save_area      ; Get original control word
    MOV     cx, 8              ; Set numeric register count
    XOR     bx, bx             ; Tag field value for stamping all registers as valid
    FSTENV  save_area          ; Save NPX environment
    FWAIT                      ; Wait for the store to complete
    XCHG    save_area+4, bx    ; Get original tag value and set new tag value
    FLDENV  save_area          ; Force all register tags as valid. BUSY is still masked. No data
    MOV     save_area, ax      ; synchronization needed. Put original control word into NPX
    MOV     save_area+4, bx    ; environment. Put original tag word into NPX environment
    XOR     bx, bx             ; Set initial register index
reg_store_loop:
    FSTP    saved_reg [bx]     ; Save register
    ADD     bx, type saved_reg ; Bump pointer to next register
    LOOP    reg_store_loop
                               ; All done
```

NPX_clean

```
;
; Force the NPX into a clean state with TOP matching the TOP field stored in the NPX context and all
; numeric registers tagged empty. Save_area must be the NPX environment saved earlier.
; Temp_env is a 7 word temporary area used to build a prototype NPX environment. Register ax will
; not be transparent.
;
NPX_clean:
    FINIT                      ; Put NPX into known state
    MOV     ax, save_area+2    ; Get original status word
    AND     ax, 3800H          ; Mask out the top field
    FSTENV  temp_env           ; Format a temporary environment area with all registers
                               ; stamped empty and TOP field = 0.
    FWAIT                      ; Wait for the store to finish.
    OR      temp_env+2, ax     ; Put in the desired TOP value.
    FLDENV  temp_env           ; Setup new NPX environment.
                               ; Now enter small_block_NPX_restore
```

small_block_NPX_restore

```
;
; Restore the NPX context without using the FRSTOR instruction. Assume the NPX context is in the
; same form as that created by an FSAVE/FNSAVE instruction, all the registers are empty, and that
; the TOP field of the NPX matches the TOP field of the NPX context. The variable save_area must
; be an NPX context save area, 47 words long. The registers bx and cx will not be transparent.
;
small_block_NPX_restore:
    MOV     cx, 8                   ; Set register count
    MOV     bx, type saved_reg*7    ; Starting offset of ST(7)
reg_load_loop:
    FLD     saved_reg [bx]          ; Get the register
    SUB     bx, type saved_reg      ; Bump pointer to next register
    LOOP    reg_load_loop
    FLDENV  save_area               ; Restore NPX context
                                    ; All done
```

APPENDIX C

OVERVIEW

Appendix C shows how floating point values can be converted to decimal ASCII character strings. The function can be called from PLM/86, PASCAL/86, FORTRAN/86, or ASM/86 functions.

Shortness, speed, and accuracy were chosen rather than providing the maximum number of significant digits possible. An attempt is made to keep integers in their own domain to avoid unnecessary conversion errors.

Using the extended precision real number format, this routine achieves a worst case accuracy of three units in the 16th decimal position for a non-integer value or integers greater than 10^{18}. This is double precision accuracy. With values having decimal exponents less than 100 in magnitude, the accuracy is one unit in the 17th decimal position.

Higher precision can be achieved with greater care in programming, larger program size, and lower performance.

Function Partitioning

Three separate modules implement the conversion. Most of the work of the conversion is done in the module FLOATING_TO_ASCII. The other modules are provided separately since they have a more general use. One of them, GET_POWER_10, is also used by the ASCII to floating point conversion routine. The other small module, TOS_STATUS, will identify what, if anything, is in the top of the numeric register stack.

Exception Considerations

Care is taken inside the function to avoid generating exceptions. Any possible numeric value will be accepted. The only exceptions possible would occur if insufficient space exists on the numeric register stack.

The value passed in the numeric stack is checked for existence, type (NAN or infinity), and status (unnormal, denormal, zero, sign). The string size is tested for a minimum and maximum value. If the top of the register stack is empty, or the string size is too small, the function will return with an error code.

Overflow and underflow is avoided inside the function for very large or very small numbers.

Special Instructions

The functions demonstrate the operation of several numeric instructions, different data types, and precision control. Shown are instructions for automatic conversion to BCD, calculating the value of 10 raised to an integer value, establishing and maintaining concurrency, data synchronization, and use of directed rounding on the NPX.

Without the extended precision data type and built-in exponential function, the double precision accuracy of this function could not be attained with the size and speed of the shown example.

The function relies on the numeric BCD data type for conversion from binary floating point to decimal. It is

not difficult to unpack the BCD digits into separate ASCII decimal digits. The major work involves scaling the floating point value to the comparatively limited range of BCD values. To print a 9-digit result requires accurately scaling the given value to an integer between 10^8 and 10^9. For example, the number +0.123456789 requires a scaling factor of 10^9 to produce the value +123456789.0 which can be stored in 9 BCD digits. The scale factor must be an exact power of 10 to avoid to changing any of the printed digit values.

These routines should exactly convert all values exactly representable in decimal in the field size given. Integer values which fit in the given string size, will not be scaled, but directly stored into the BCD form. Non-integer values exactly representable in decimal within the string size limits will also be exactly converted. For example, 0.125 is exactly representable in binary or decimal. To convert this floating point value to decimal, the scaling factor will be 1000, resulting in 125. When scaling a value, the function must keep track of where the decimal point lies in the final decimal value.

DESCRIPTION OF OPERATION

Converting a floating point number to decimal ASCII takes three major steps: identifying the magnitude of the number, scaling it for the BCD data type, and converting the BCD data type to a decimal ASCII string.

Identifying the magnitude of the result requires finding the value X such that the number is represented by $I*10^X$, where $1.0 <= I < 10.0$. Scaling the number requires multiplying it by a scaling factor 10^S, such that the result is an integer requiring no more decimal digits than provided for in the ASCII string.

Once scaled, the numeric rounding modes and BCD conversion put the number in a form easy to convert to decimal ASCII by host software.

Implementing each of these three steps requires attention to detail. To begin with, not all floating point values have a numeric meaning. Values such as infinity, indefinite, or Not A Number (NAN) may be encountered by the conversion routine. The conversion routine should recognize these values and identify them uniquely.

Special cases of numeric values also exist. Denormals, unnormals, and pseudo zero all have a numeric value but should be recognized since all of them indicate that precision was lost during some earlier calculations.

Once it has been determined that the number has a numeric value, and it is normalized setting appropriate unnormal flags, the value must be scaled to the BCD range.

Scaling the Value

To scale the number, its magnitude must be determined. It is sufficient to calculate the magnitude to an accuracy of 1 unit, or within a factor of 10 of the given value. After scaling the number, a check will be made to see if the result falls in the range expected. If not, the result can be adjusted one decimal order of magnitude up or down. The adjustment test after the scaling is necessary due to inevitable inaccuracies in the scaling value.

Since the magnitude estimate need only be close, a fast technique is used. The magnitude is estimated by multiplying the power of 2, the unbiased floating point exponent, associated with the number by $\log_{10}2$. Rounding the result to an integer will produce an estimate of sufficient accuracy. Ignoring the fraction value can introduce a maximum error of 0.32 in the result.

Using the magnitude of the value and size of the number string, the scaling factor can be calculated. Calculating the scaling factor is the most inaccurate operation of the conversion process. The relation $10^X = 2^{**}(X^*\log_2 10)$ is used for this function. The exponentiate instruction (F2XM1) will be used.

Due to restrictions on the range of values allowed by the F2XM1 instruction, the power of 2 value will be split into integer and fraction components. The relation $2^{**}(I+F) = 2^{**}I * 2^{**}F$ allows using the FSCALE instruction to recombine the 2**F value, calculated through F2XM1, and the 2**I part.

Inaccuracy in Scaling

The inaccuracy of these operations arises because of the trailing zeroes placed into the fraction value when stripping off the integer valued bits. For each integer valued bit in the power of 2 value separated from the fraction bits, one bit of precision is lost in the fraction field due to the zero fill occurring in the least significant bits.

Up to 14 bits may be lost in the fraction since the largest allowed floating point exponent value is $2^{14}-1$.

AVOIDING UNDERFLOW AND OVERFLOW

The fraction and exponent fields of the number are separated to avoid underflow and overflow in calculating the scaling values. For example, to scale 10^{-4932} to 10^8 requires a scaling factor of 10^{4950} which cannot be represented by the NPX.

By separating the exponent and fraction, the scaling operation involves adding the exponents separate from multiplying the fractions. The exponent arithmetic will involve small integers, all easily represented by the NPX.

FINAL ADJUSTMENTS

It is possible that the power function (Get_Power_10) could produce a scaling value such that it forms a scaled result larger than the ASCII field could allow. For example, scaling 9.9999999999999999e4900 by 1.0000000000000010e-4883 would produce 1.0000000000000009e18. The scale factor is within the accuracy of the NDP and the result is within the conversion accuracy, but it cannot be represented in BCD format. This is why there is a post-scaling test on the magnitude of the result. The result can be multiplied or divided by 10, depending on whether the result was too small or too large, respectively.

Output Format

For maximum flexibility in output formats, the position of the decimal point is indicated by a binary integer called the power value. If the power value is zero, then the decimal point is assumed to be at the right of the right-most digit. Power values greater than zero indicate how many trailing zeroes are not shown. For each unit below zero, move the decimal point to the left in the string.

The last step of the conversion is storing the result in BCD and indicating where the decimal point lies. The BCD string is then unpacked into ASCII decimal characters. The ASCII sign is set corresponding to the sign of the original value.

```
LINE     SOURCE

   1     $title(Convert a floating point number to ASCII)
   2                     name    floating_to_ascii
   3                     public  floating_to_ascii
   4                     extrn   get_power_10:near,tos_status:near
   5     ;
   6     ;         This subroutine will convert the floating point number in the
   7     ;       top of the 8087 stack to an ASCII string and separate power of 10
   8     ;       scaling value (in binary).  The maximum width of the ASCII string
   9     ;       formed is controlled by a parameter which must be > 1.  Unnormal values,
  10     ;       denormal values, and psuedo zeroes will be correctly converted.
  11     ;       A returned value will indicate how many binary bits of
  12     ;       precision were lost in an unnormal or denormal value.  The magnitude
  13     ;       (in terms of binary power) of a psuedo zero will also be indicated.
  14     ;       Integers less than 10**18 in magnitude are accurately converted if the
  15     ;       destination ASCII string field is wide enough to hold all the
  16     ;       digits.  Otherwise the value is converted to scientific notation.
  17     ;
  18     ;         The status of the conversion is identified by the return value,
  19     ;       it can be:
  20     ;
  21     ;                 0       conversion complete, string_size is defined
  22     ;                 1       invalid arguments
  23     ;                 2       exact integer conversion, string_size is defined
  24     ;                 3       indefinite
  25     ;                 4       + NAN (Not A Number)
  26     ;                 5       - NAN
  27     ;                 6       + Infinity
  28     ;                 7       - Infinity
  29     ;                 8       psuedo zero found, string_size is defined
  30     ;
  31     ;         The PLM/86 calling convention is:
  32     ;
  33     ; floating_to_ascii:
  34     ;       procedure (number,denormal_ptr,string_ptr,size_ptr,field_size,
  35     ;               power_ptr) word external;
  36     ;       declare (denormal_ptr,string_ptr,power_ptr,size_ptr) pointer;
  37     ;       declare field_size word, string_size based size_ptr word;
  38     ;       declare number real;
  39     ;       declare denormal integer based denormal_ptr;
  40     ;       declare power integer based power_ptr;
  41     ;       end floating_to_ascii;
  42     ;
  43     ;         The floating point value is expected to be on the top of the NPX
  44     ;       stack.  This subroutine expects 3 free entries on the NPX stack and
  45     ;       will pop the passed value off when done.  The generated ASCII string
  46     ;       will have a leading character either '-' or '+' indicating the sign
  47     ;       of the value.  The ASCII decimal digits will immediately follow.
  48     ;       The numeric value of the ASCII string is (ASCII STRING.)*10**POWER.
```

```
49      ;       It the given number was zero, the ASCII string will contain a sign
50      ;       and a single zero chacter.  The value string_size indicates the total
51      ;       length of the ASCII string including the sign character.  String(0) will
52      ;       always hold the sign.  It is possible for string_size to be less than
53      ;       field_size.  This occurs for zeroes or integer values.  A psuedo zero
54      ;       will return a special return code.  The denormal count will indicate
55      ;       the power of two originally associated with the value.  The power of
56      ;       ten and ASCII string will be as if the value was an ordinary zero.
57      ;
58      ;         This subroutine is accurate up to a maximum of 18 decimal digits for
59      ;       integers.  Integer values will have a decimal power of zero associated
60      ;       with them.  For non integers, the result will be accurate to within 2
61      ;       decimal digits of the 16th decimal place (double precision).  The
62      ;       exponentiate instruction is also used for scaling the value into the
63      ;       range acceptable for the BCD data type.  The rounding mode in effect
64      ;       on entry to the subroutine is used for the conversion.
65      ;
66      ;         The following registers are not transparent:
67      ;
68      ;               ax bx cx dx si di flags
69      ;
70
71      ;
72      ;         Define the stack layout.
73      ;
74      bp_save         equ     word ptr [bp]
75      es_save         equ     bp_save + size bp_save
76      return_ptr      equ     es_save + size es_save
77      power_ptr       equ     return_ptr + size return_ptr
78      field_size      equ     power_ptr + size power_ptr
79      size_ptr        equ     field_size + size field_size
80      string_ptr      equ     size_ptr + size size_ptr
81      denormal_ptr    equ     string_ptr + size string_ptr
82
83      parms_size      equ     size power_ptr + size field_size + size size_ptr +
84      &                       size string_ptr + size denormal_ptr
85      ;
86      ;         Define constants used
87      ;
88      BCD_DIGITS      equ     18              ; Number of digits in bcd_value
89      WORD_SIZE       equ     2
90      BCD_SIZE        equ     10
91      MINUS           equ     1               ; Define return values
92      NAN             equ     4               ; The exact values chosen here are
93      INFINITY        equ     6               ; important.  They must correspond to
94      INDEFINITE      equ     3               ; the possible return values and be in
95      PSUEDO_ZERO     equ     8               ; the same numeric order as tested by
96      INVALID         equ     -2              ; the program.
97      ZERO            equ     -4
98      DENORMAL        equ     -6
99      UNNORMAL        equ     -8
100     NORMAL          equ     0
101     EXACT           equ     2
102     ;
103     ;         Define layout of temporary storage area.
104     ;
105     status          equ     word ptr [bp-WORD_SIZE]
106     power_two       equ     status - WORD_SIZE
107     power_ten       equ     power_two - WORD_SIZE
108     bcd_value       equ     tbyte ptr power_ten - BCD_SIZE
109     bcd_byte        equ     byte ptr bcd_value
110     fraction        equ     bcd_value
111
112     local_size      equ     size status + size power_two + size power_ten
113     &                       + size bcd_value
114     ;
115     ;         Allocate stack space for the temporaries so the stack will be big enough
116     ;
117     stack           segment stack 'stack'
118                     db      (local_size+6) dup (?)
```

```
120
121     cgroup          group   code
122     code            segment public 'code'
123                     assume  cs:cgroup
124                     extrn   power_table:qword
125     ;
126     ;         Constants used by this function.
127     ;
128                     even                    ; Optimize for 16 bits
129     const10         dw      10              ; Adjustment value for too big BCD
130     ;
131     ;         Convert the C3,C2,C1,C0 encoding from tos_status into meaningful bit
132     ;       flags and values.
133     ;
134     status_table    db      UNNORMAL, NAN, UNNORMAL + MINUS, NAN + MINUS,

135     &                       NORMAL, INFINITY, NORMAL + MINUS, INFINITY + MINUS,

136     &                       ZERO, INVALID, ZERO + MINUS, INVALID,

137     &                       DENORMAL, INVALID, DENORMAL + MINUS, INVALID

138
139     floating_to_ascii proc
140
141             call    tos_status              ; Look at status of ST(0)
142             mov     bx,ax                   ; Get descriptor from table
143             mov     al,status_table[bx]
144             cmp     al,INVALID              ; Look for empty ST(0)
145             jne     not_empty
146     ;
147     ;         ST(0) is empty!  Return the status value.
148     ;
149             ret     parms_size
150     ;
151     ;         Remove infinity from stack and exit.
152     ;
153     found_infinity:
154
155             fstp    st(0)                   ; OK to leave fstp running
156             jmp     short exit_proc
157     ;
158     ;         String space is too small!  Return invalid code.
159     ;
160     small_string:
161
162             mov     al,INVALID
163
164     exit_proc:
165
166             mov     sp,bp                   ; Free stack space
167             pop     bp                      ; Restore registers
168             pop     es
169             ret     parms_size
170     ;
171     ;         ST(0) is NAN or indefinite.  Store the value in memory and look
172     ;       at the fraction field to separate indefinite from an ordinary NAN.
173     ;
174     NAN_or_indefinite:
175
176             fstp    fraction                ; Remove value from stack for examination
177             test    al,MINUS                ; Look at sign bit
178             fwait                           ; Insure store is done
179             jz      exit_proc               ; Can't be indefinite if positive
180
```

```
181             mov     bx,0C000H               ; Match against upper 16 bits of fraction
182             sub     bx,word ptr fraction+6  ; Compare bits 63-48
183             or      bx,word ptr fraction+4  ; Bits 32-47 must be zero
184             or      bx,word ptr fraction+2  ; Bits 31-16 must be zero
185             or      bx,word ptr fraction    ; Bits 15-0 must be zero
186             jnz     exit_proc
187
188             mov     al,INDEFINITE           ; Set return value for indefinite value
189             jmp     exit_proc
190     ;
191     ;         Allocate stack space for local variables and establish parameter
192     ;       addressibility.
193     ;
194     not_empty:
195
196             push    es                      ; Save working register
197             push    bp
198             mov     bp,sp                   ; Establish stack addressibility
199             sub     sp,local_size
200
201             mov     cx,field_size           ; Check for enough string space
202             cmp     cx,2
203             jl      small_string
204
205             dec     cx                      ; Adjust for sign character
206             cmp     cx,BCD_DIGITS           ; See if string is too large for BCD
207             jbe     size_ok
208
209             mov     cx,BCD_DIGITS           ; Else set maximum string size
210
211     size_ok:
212
213             cmp     al,INFINITY             ; Look for infinity
214             jge     found_infinity          ; Return status value for + or - inf.
215
216             cmp     al,NAN                  ; Look for NAN or INDEFINITE
217             jge     NAN_or_indefinite
218     ;
219     ;         Set default return values and check that the number is normalized.
220     ;
221             fabs                            ; Use positive value only
222                                             ; sign bit in al has true sign of value
223             mov     dx,ax                   ; Save return value for later
224             xor     ax,ax                   ; Form 0 constant
225             mov     di,denormal_ptr         ; Zero denormal count
226             mov     word ptr [di],ax
227             mov     bx,power_ptr            ; Zero power of ten value
228             mov     word ptr [bx],ax
229             cmp     dl,ZERO                 ; Test for zero
230             jae     real_zero               ; Skip power code if value is zero
231
232             cmp     dl,DENORMAL             ; Look for a denormal value
233             jae     found_denormal          ; Handle it specially
234
235             fxtract                         ; Separate exponent from significand
236             cmp     dl,UNNORMAL             ; Test for unnormal value
237             jb      normal_value
238
239             sub     dl,UNNORMAL-NORMAL      ; Return normal status with correct sign
240     ;
241     ;         Normalize the fraction, adjust the power of two in ST(1) and set
242     ;       the denormal count value.
243     ;
244     ;       Assert: 0 <= ST(0) < 1.0
245     ;
246             fld1                            ; Load constant to normalize fraction
247
248     normalize_fraction:
249
250             fadd    st(1),st                ; Set integer bit in fraction
251             fsub                            ; Form normalized fraction in ST(0)
252             fxtract                         ; Power of two field will be negative
253                                             ; of denormal count
254             fxch                            ; Put denormal count in ST(0)
```

```
255             fist    word ptr [di]           ; Put negative of denormal count in memory
256             faddp   st(2),st                ; Form correct power of two in st(1)
257                                             ; OK to use word ptr [di] now
258             neg     word ptr [di]           ; Form positive denormal count
259             jnz     not_psuedo_zero
260     ;
261     ;         A psuedo zero will appear as an unnormal number.  When attempting
262     ;       to normalize it, the resultant fraction field will be zero.  Performing
263     ;       an fxtract on zero will yield a zero exponent value.
264     ;
265             fxch                            ; Put power of two value in st(0)
266             fistp   word ptr [di]           ; Set denormal count to power of two value
267                                             ; Word ptr [di] is not used by convert
268                                             ; integer, OK to leave running
269             sub     dl,NORMAL-PSUEDO_ZERO   ; Set return value saving the sign bit
270             jmp     convert_integer         ; Put zero value into memory
271     ;
272     ;         The number is a real zero, set the return value and setup for
273     ;       conversion to BCD.
274     ;
275     real_zero:
276
277             sub     dl,ZERO-NORMAL          ; Convert status to normal value
278             jmp     convert_integer         ; Treat the zero as an integer
279     ;
280     ;         The number is a denormal.  FXTRACT will not work correctly in this
281     ;       case.  To correctly separate the exponent and fraction, add a fixed
282     ;       constant to the exponent to guarantee the result is not a denormal.
283     ;
284     found_denormal:
285
286             fldl                            ; Prepare to bump exponent
287             fxch
288             fprem                           ; Force denormal to smallest representable
289                                             ; extended real format exponent
290             fxtract                         ; This will work correctly now
291     ;
292     ;         The power of the original denormal value has been safely isolated.
293     ;       Check if the fraction value is an unnormal.
294     ;
295             fxam                            ; See if the fraction is an unnormal
296             fstsw   status                  ; Save status for later
297             fxch                            ; Put exponent in ST(0)
298             fxch    st(2)                   ; Put 1.0 into ST(0), exponent in ST(2)
299             sub     dl,DENORMAL-NORMAL      ; Return normal status with correct sign
300             test    status,4400H            ; See if C3=C2=0 impling unnormal or NAN
301             jz      normalize_fraction      ; Jump if fraction is an unnormal
302
303             fstp    st(0)                   ; Remove unnecessary 1.0 from st(0)
304     ;
305     ;         Calculate the decimal magnitude associated with this number to
306     ;       within one order.  This error will always be inevitable due to
307     ;       rounding and lost precision.  As a result, we will deliberately fail
308     ;       to consider the LOG10 of the fraction value in calculating the order.
309     ;       Since the fraction will always be 1 <= F < 2, its LOG10 will not change
310     ;       the basic accuracy of the function.  To get the decimal order of magnitude,
311     ;       simply multiply the power of two by LOG10(2) and truncate the result to
312     ;       an integer.
313     ;
314     normal_value:
315     not_psuedo_zero:
316
317             fstp    fraction                ; Save the fraction field for later use
318             fist    power_two               ; Save power of two
319             fldlg2                          ; Get LOG10(2)
320                                             ; Power_two is now safe to use
321             fmul                            ; Form LOG10(of exponent of number)
322             fistp   power_ten               ; Any rounding mode will work here
323     ;
324     ;         Check if the magnitude of the number rules out treating it as
325     ;       an integer.
326     ;
327     ;       CX has the maximum number of decimal digits allowed.
```

```
328     ;
329             fwait                           ; Wait for power_ten to be valid
330             mov     ax,power_ten            ; Get power of ten of value
331             sub     ax,cx                   ; Form scaling factor necessary in ax
332             ja      adjust_result           ; Jump if number will not fit
333     ;
334     ;         The number is between 1 and 10**(field_size).
335     ;       Test if it is an integer.
336     ;
337             fild    power_two               ; Restore original number
338             mov     si,dx                   ; Save return value
339             sub     dl,NORMAL-EXACT         ; Convert to exact return value
340             fld     fraction
341             fscale                          ; Form full value, this is safe here
342             fst     st(1)                   ; Copy value for compare
343             frndint                         ; Test if its an integer
344             fcomp                           ; Compare values
345             fstsw   status                  ; Save status
346             test    status,4000H            ; C3=1 implies it was an integer
347             jnz     convert_integer
348
349             fstp    st(0)                   ; Remove non integer value
350             mov     dx,si                   ; Restore original return value
351     ;
352     ;         Scale the number to within the range allowed by the BCD format.
353     ;       The scaling operation should produce a number within one decimal order
354     ;       of magnitude of the largest decimal number representable within the
355     ;       given string width.
356     ;
357     ;         The scaling power of ten value is in ax.
358     ;
359     adjust_result:
360
361             mov     word ptr [bx],ax        ; Set initial power of ten return value
362             neg     ax                      ; Subtract one for each order of
363                                             ; magnitude the value is scaled by
364             call    get_power_10            ; Scaling factor is returned as exponent
365                                             ; and fraction
366             fld     fraction                ; Get fraction
367             fmul                            ; Combine fractions
368             mov     si,cx                   ; Form power of ten of the maximum
369             shl     si,1                    ; BCD value to fit in the string
370             shl     si,1                    ; Index in si
371             shl     si,1
372             fild    power_two               ; Combine powers of two
373             faddp   st(2),st
374             fscale                          ; Form full value, exponent was safe
375             fstp    st(1)                   ; Remove exponent
376     ;
377     ;         Test the adjusted value against a table of exact powers of ten.
378     ;       The combined errors of the magnitude estimate and power function can
379     ;       result in a value one order of magnitude too small or too large to fit
380     ;       correctly in the BCD field.  To handle this problem, pretest the
381     ;       adjusted value, if it is too small or large, then adjust it by ten and
382     ;       adjust the power of ten value.
383     ;
384     test_power:
385
386             fcom    power_table[si]+type power_table; Compare against exact power
387                                             ; entry.  Use the next entry since cx
388                                             ; has been decremented by one
389             fstsw   status                  ; No wait is necessary
390             test    status,4100H            ; If C3 = C0 = 0 then too big
391             jnz     test_for_small
392
393             fidiv   const10                 ; Else adjust value
394             and     dl,not EXACT            ; Remove exact flag
395             inc     word ptr [bx]           ; Adjust power of ten value
396             jmp     short in_range          ; Convert the value to a BCD integer
397
398     test_for_small:
399
400             fcom    power_table[si]         ; Test relative size
401             fstsw   status                  ; No wait is necessary
```

```
402             test    status,100H             ; If C0 = 0 then st(0) >= lower bound
403             jz      in_range                ; Convert the value to a BCD integer
404
405             fimul   const10                 ; Adjust value into range
406             dec     word ptr [bx]           ; Adjust power of ten value
407
408     in_range:
409
410             frndint                         ; Form integer value
411     ;
412     ;       Assert: 0 <= TOS <= 999,999,999,999,999,999
413     ;       The TOS number will be exactly representable in 18 digit BCD format.
414     ;
415     convert_integer:
416
417             fbstp   bcd_value               ; Store as BCD format number
418     ;
419     ;         While the store BCD runs, setup registers for the conversion to
420     ;       ASCII.
421     ;
422             mov     si,BCD_SIZE-2           ; Initial BCD index value
423             mov     cx,0f04h                ; Set shift count and mask
424             mov     bx,1                    ; Set initial size of ASCII field for sign
425             mov     di,string_ptr           ; Get address of start of ASCII string
426             mov     ax,ds                   ; Copy ds to es
427             mov     es,ax
428             cld                             ; Set autoincrement mode
429             mov     al,'+'                  ; Clear sign field
430             test    dl,MINUS                ; Look for negative value
431             jz      positive_result
432
433             mov     al,'-'
434
435     positive_result:
436
437             stosb                           ; Bump string pointer past sign
438             and     dl,not MINUS            ; Turn off sign bit
439             fwait                           ; Wait for fbstp to finish
440     ;
441     ;         Register usage:
442     ;                                 ah:     BCD byte value in use
443     ;                                 al:     ASCII character value
444     ;                                 dx:     Return value
445     ;                                 ch:     BCD mask = 0fh
446     ;                                 cl:     BCD shift count = 4
447     ;                                 bx:     ASCII string field width
448     ;                                 si:     BCD field index
449     ;                                 di:     ASCII string field pointer
450     ;                                 ds,es:  ASCII string segment base
451     ;
452     ;         Remove leading zeroes from the number.
453     ;
454     skip_leading_zeroes:
455
456             mov     ah,bcd_byte[si]         ; Get BCD byte
457             mov     al,ah                   ; Copy value
458             shr     al,cl                   ; Get high order digit
459             and     al,ch                   ; Set zero flag
460             jnz     enter_odd               ; Exit loop if leading non zero found
461
462             mov     al,ah                   ; Get BCD byte again
463             and     al,ch                   ; Get low order digit
464             jnz     enter_even              ; Exit loop if non zero digit found
465
466             dec     si                      ; Decrement BCD index
467             jns     skip_leading_zeroes
468     ;
469     ;         The significand was all zeroes.
470     ;
471             mov     al,'0'                  ; Set initial zero
472             stosb
473             inc     bx                      ; Bump string length
474             jmp     short exit_with_value
```

```
475     ;
476     ;         Now expand the BCD string into digit per byte values 0-9.
477     ;
478     digit_loop:
479
480             mov     ah,bcd_byte[si]         ; Get BCD byte
481             mov     al,ah
482             shr     al,cl                   ; Get high order digit
483
484     enter_odd:
485
486             add     al,'0'                  ; Convert to ASCII
487             stosb                           ; Put digit into ASCII string area
488             mov     al,ah                   ; Get low order digit
489             and     al,ch
490             inc     bx                      ; Bump field size counter
491
492     enter_even:
493
494             add     al,'0'                  ; Convert to ASCII
495             stosb                           ; Put digit into ASCII area
496             inc     bx                      ; Bump field size counter
497             dec     si                      ; Go to next BCD byte
498             jns     digit_loop
499     ;
500     ;         Conversion complete.  Set the string size and remainder.
501     ;
502     exit_with_value:
503
504             mov     di,size_ptr
505             mov     word ptr [di],bx
506             mov     ax,dx                   ; Set return value
507             jmp     exit_proc
508
509     floating_to_ascii       endp
510     code                    ends
511                             end

ASSEMBLY COMPLETE, NO ERRORS FOUND


LINE    SOURCE

   1    $title(Calculate the value of 10**ax)
   2    ;
   3    ;         This subroutine will calculate the value of 10**ax.
   4    ;       All 8086 registers are transparent and the value is returned on
   5    ;       the TOS as two numbers, exponent in ST(1) and fraction in ST(0).
   6    ;       The exponent value can be larger than the maximum representable
   7    ;       exponent.  Three stack entries are used.
   8    ;
   9                    name    get_power_10
  10                    public  get_power_10,power_table
  11
  12    stack           segment stack  'stack'
  13                    dw      4 dup (?)               ; Allocate space on the stack


  14    stack           ends
  15
  16    cgroup          group   code
  17    code            segment public 'code'
  18                    assume  cs:cgroup
  19    ;
  20    ;         Use exact values from 1.0 to 1e18.
  21    ;
  22                    even                            ; Optimize 16 bit access
  23    power_table     dq      1.0,1e1,1e2,1e3
```

```
24                      dq      le4,le5,le6,le7

25                      dq      le8,le9,le10,le11

26                      dq      le12,le13,le14,le15

27                      dq      le16,le17,le18


28
29      get_power_10    proc
30
31              cmp     ax,18                   ; Test for 0 <= ax < 19
32              ja      out_of_range
33
34              push    bx                      ; Get working index register
35              mov     bx,ax                   ; Form table index
36              shl     bx,1
37              shl     bx,1
38              shl     bx,1
39              fld     power_table[bx]         ; Get exact value
40              pop     bx                      ; Restore register value
41              fxtract                         ; Separate power and fraction
42              ret                             ; OK to leave fxtract running
43      ;
44      ;         Calculate the value using the exponentiate instruction.
45      ;       The following relations are used:
46      ;               10**x = 2**(log2(10)*x)
47      ;               2**(I+F) = 2**I * 2**F
48      ;               if st(1) = I and st(0) = 2**F then fscale produces 2**(I+F)
49      ;
50      out_of_range:
51
52              fldl2t                          ; TOS = LOG2(10)
53              push    bp                      ; Establish stack addressibility
54              mov     bp,sp
55              push    ax                      ; Put power (P) in memory
56              push    ax                      ; Allocate space for status
57              fimul   word ptr [bp-2]         ; TOS,X = LOG2(10)*P = LOG2(10**P)
58              fnstcw  word ptr [bp-4]         ; Get current control word
59                                              ; Control word is a static value
60              mov     ax,word ptr [bp-4]      ; Get control word, no wait necessary
61              and     ax,not 0C00H            ; Mask off current rounding field
62              or      ax,0400H                ; Set round to negative infinity
63              xchg    ax,word ptr [bp-4]      ; Put new control word in memory
64                                              ; old control word is in ax
65              fldl                            ; Set TOS = -1.0
66              fchs
67              fld     st(1)                   ; Copy power value in base two
68              fldcw   word ptr [bp-4]         ; Set new control word value
69              frndint                         ; TOS = I: -inf < I <= X, I is an integer
70              mov     word ptr [bp-4],ax      ; Restore original rounding control
71              fldcw   word ptr [bp-4]
```

```
 72              fxch    st(2)                   ; TOS = X, ST(1) = -1.0, ST(2) = I
 73              pop     ax                      ; Remove original control word
 74              fsub    st,st(2)                ; TOS,F = X-I: 0 <= TOS < 1.0
 75              pop     ax                      ; Restore power of ten
 76              fscale                          ; TOS = F/2: 0 <= TOS < 0.5
 77              f2xml                           ; TOS = 2**(F/2) - 1.0
 78              pop     bp                      ; Restore stack
 79              fsubr                           ; Form 2**(F/2)
 80              fmul    st,st(0)                ; Form 2**F
 81              ret                             ; OK to leave fmul running
 82
 83      get_power_10    endp
 84      code            ends
 85                      end

ASSEMBLY COMPLETE, NO ERRORS FOUND


LINE    SOURCE

  1     $title(Determine TOS register contents)
  2     ;
  3     ;         This subroutine will return a value from 0-15 in ax corresponding
  4     ;       to the contents of 8087 TOS.  All registers are transparent and no
  5     ;       errors are possible.  The return value corresponds to c3,c2,c1,c0
  6     ;       of FXAM instruction.
  7     ;
  8             name    tos_status
  9             public  tos_status
 10
 11     stack            segment stack 'stack'
 12                      dw      3 dup (?)       ; Allocate space on the stack


 13     stack            ends
 14
 15     cgroup           group   code
 16     code             segment public 'code'
 17                      assume  cs:cgroup
 18     tos_status       proc
 19
 20             fxam                     ; Get register contents status
 21             push    ax               ; Allocate space for status value
 22             push    bp               ; Establish stack addressibility
 23             mov     bp,sp
 24             fstsw   word ptr [bp+2]  ; Put tos status in memory
 25             pop     bp               ; Restore registers
 26             pop     ax               ; Get status value, no wait necessary
 27             mov     al,ah            ; Put bit 10-8 into bits 2-0
 28             and     ax,4007h         ; Mask out bits c3,c2,c1,c0
 29             shr     ah,1             ; Put bit c3 into bit 11
 30             shr     ah,1
 31             shr     ah,1
 32             or      al,ah            ; Put c3 into bit 3
 33             mov     ah,0             ; Clear return value
 34             ret
 35
 36     tos_status       endp
 37     code             ends
 38                      end

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX D

OVERVIEW

Appendix D shows a function for converting ASCII input strings into floating point values. The returned value can be used by PLM/86, PASCAL/86, FORTRAN/86, or ASM/86. The routine will accept a number in ASCII of standard FORTRAN formats. Up to 18 decimal digits are accepted and the conversion accuracy is the same as for converting in the other direction. Greater accuracy can also be achieved with similar tradeoffs, as mentioned earlier.

Description of Operation

Converting from ASCII to floating point is less complex numerically than going from floating point to ASCII. It consists of four basic steps: determine the size in decimal digits of the number, build a BCD value corresponding to the number string if the decimal point were at the far right, calculate the exponent value, and scale the BCD value. The first three steps are performed by the host software. The fourth step is mainly performed by numeric operations.

The complexity in this function arises due to the flexible nature of the input values it will recognize. Most of the code simply determines the meaning of each character encountered. Two separate number inputs must be recognized, mantissa and exponent values. Performing the numerics operations is very straightforward.

The length of the number string is determined first to allow building a BCD number from low digits to high digits. This technique guarantees that an integer will be converted to its exact BCD integer equivalent.

If the number is a floating point value, then the digit string can be scaled appropriately. If a decimal point occurs within the string, the scale factor must be decreased by one for each digit the decimal point is moved to the right. This factor must be added to any exponent value specified in the number.

ACCURACY CONSIDERATIONS

All the same considerations for converting floating point to ASCII apply to calculating the scaling factor. The accuracy of the scale factor determines the accuracy of the result.

The exponents and fractions are again kept separate to prevent overflows or underflows during the scaling operations.

```
LINE    SOURCE

   1    $title(ASCII to floating point conversion)
   2    ;
   3    ;         Define the publicly known names.
   4    ;
   5                  name    ascii_to_floating
   6                  public  ascii_to_floating
   7                  extrn   get_power_10:near
   8    ;
   9    ;         This function will convert an ASCII character string to a floating
  10    ;       point representation.  Character strings in integer or scientific form
  11    ;       will be accepted.  The allowed format is:
  12    ;
  13    ;       [+,-][digit(s)][.][digit(s)][E,e][+,-][digit(s)]
  14    ;
  15    ;          Where a digit must have been encountered before the exponent
  16    ;       indicator 'E' or'e'.  If a '+', '-', or '.' was encountered, then at
  17    ;       least one digit must exist before the optional exponent field.  A value
  18    ;       will always be returned in the 8087 stack.  In case of invalid numbers,
  19    ;       values like indefinite or infinity will be returned.
  20    ;
  21    ;         The first character not fitting within the format will terminate the
  22    ;       conversion.  The address of the terminating character will be returned
  23    ;       by this subroutine.
  24    ;
  25    ;         The result will be left on the top of the NPX stack.  This
  26    ;       subroutine expects 3 free NPX stack registers.  The sign of the result
  27    ;       will correspond to any sign characters in the ASCII string.  The rounding
  28    ;       mode in effect at the time the subroutine was called will be used for
  29    ;       the conversion from base 10 to base 2.  Up to 18 significant decimal
  30    ;       digits may appear in the number.  Leading zeroes, trailing zeroes, or
  31    ;       exponent digits do not count towards the 18 digit maximum.  Integers
  32    ;       or exactly representable decimal numbers of 18 digits or less will be
  33    ;       exactly converted.  The technique used constructs a BCD number
```

```
34      ;       representing the significant ASCII digits of the string with the decimal
35      ;       point removed.
36      ;
37      ;         An attempt is made to exactly convert relatively small integers or
38      ;       small fractions.  For example the values: .06125, 123456789012345678,
39      ;       1e17, 1.23456e5, and 125e-3 will be exactly converted to floating point.
40      ;         The exponentiate instruction is used to scale the generated BCD vaslue
41      ;       to very large or very small numbers.  The basic accuracy of this function
42      ;       determines the accuracy of this subroutine.  For very large or very small
43      ;       numbers, the accuracy of this function is 2 units in the 16th decimal
44      ;       place or double precision.  The range of decimal powers accepted is
45      ;       10**-4930 to 10**4930.
46      ;
47      ;         The PLM/86 calling format is:
48      ;
49      ;       ascii_to_floating:
50      ;               procedure (string_ptr,end_ptr,status_ptr) real external;
51      ;               declare (string_ptr,end_ptr,status_ptr) pointer;
52      ;               declare end based end_ptr pointer;
53      ;               declare status based status_ptr word;
54      ;               end;
55      ;
56      ;         The status value has 6 possible states:
57      ;
58      ;               0       A number was found.
59      ;               1       No number was found, return indefinite.
60      ;               2       Exponent was expected but none found, return indefinite.
61      ;               3       Too many digits were found, return indefinite.
62      ;               4       Exponent was too big, return a signed infinity.
63      ;
64      ;         The following registers are used by this subroutine:
65      ;
66      ;       ax      bx      cx      dx      si      di
67      ;
68
69      ;
70      ;         Define constants.
71      ;
72      LOW_EXPONENT    equ     -4930                   ; Smallest allowed power of 10
73      HIGH_EXPONENT   equ     4930                    ; Largest allowed power of 10
74      WORD_SIZE       equ     2
75      BCD_SIZE        equ     10
76      ;
77      ;         Define the parameter layouts involved:
78      ;
79      bp_save         equ     word ptr [bp]
80      return_ptr      equ     bp_save + size bp_save
81      status_ptr      equ     return_ptr + size return_ptr
82      end_ptr         equ     status_ptr + size status_ptr
83      string_ptr      equ     end_ptr + size end_ptr
84
85      parms_size      equ     size status_ptr + size end_ptr + size string_ptr
86      ;
87      ;         Define the local variable data layouts
88      ;
89      power_ten       equ     word ptr [bp- WORD_SIZE] ; power of ten value
90      bcd_form        equ     tbyte ptr power_ten - BCD_SIZE; BCD representation
91
92      local_size      equ     size power_ten + size bcd_form
93      ;
94      ;         Define common expressions used
95      ;
96      bcd_byte        equ     byte ptr bcd_form       ; Current byte in the BCD form
97      bcd_count       equ     (type(bcd_form)-1)*2    ; Number of digits in BCD form
98      bcd_sign        equ     byte ptr bcd_form + 9   ; Address of BCD sign byte
99      bcd_sign_bit    equ     80H
100     ;
101     ;         Define return values.
102     ;
103     NUMBER_FOUND    equ     0               ; Number was found
104     NO_NUMBER       equ     1               ; No number was found
105     NO_EXPONENT     equ     2               ; No exponent was found when expected
106     TOO_MANY_DIGITS equ     3               ; Too many digits were found
107     EXPONENT_TOO_BIG equ    4               ; Exponent was too big
```

```
108     ;
109     ;         Allocate stack space to insure enough exists at run time.
110     ;
111     stack           segment stack 'stack'
112                     db      (local_size+4) dup (?)

113     stack           ends
114
115     cgroup          group   code
116     code            segment public 'code'
117                     assume  cs:cgroup
118     ;
119     ;         Define some of the possible return values.
120     ;
121                     even                    ; Optimize 16 bit access
122     indefinite      dd      0FFC00000R      ; Single precision real for indefinite
123     infinity        dd      07FF80000R      ; Single precision real for +infinity
124
125     ascii_to_floating proc
126
127             fldz                            ; Prepare to zero BCD value
128             push    bp                      ; Save callers stack environment
129             mov     bp,sp                   ; Establish stack addressibility
130             sub     sp,local_size           ; Allocate space for local variables
131     ;
132     ;         Get any leading sign character to form initial BCD template.
133     ;
134             mov     si,string_ptr           ; Get starting address of the number
135             xor     dx,dx                   ; Set initial decimal digit count
136             cld                             ; Set autoincrement mode
137     ;
138     ;         Register usage:
139     ;
140     ;               al:     Current character value being examined
141     ;               cx:     Digit count before the decimal point
142     ;               dx:     Total digit count
143     ;               si:     Pointer to character string
144     ;
145     ;         Look for an initial sign and skip it if found.
146     ;
147             lodsb                           ; Get first character
148             cmp     al,'+'                  ; Look for a sign
149             jz      scan_leading_digits
150
151             cmp     al,'-'
152             jnz     enter_leading_digits    ; If not "-" test current character
153
154             fchs                            ; Set TOS = -0
155     ;
156     ;         Count the number of digits appearing before an optional decimal point.
157     ;
158     scan_leading_digits:
159
160             lodsb                           ; Get next character
161
162     enter_leading_digits:
163
164             call    test_digit              ; Test for digit and bump counter
165             jnc     scan_leading_digits
166     ;
167     ;         Look for a possible decimal point and start fbstp operation.
168     ;       The fbstp zeroes out the BCD value and sets the correct sign.
169     ;
170             fbstp   bcd_form                ; Set initial sign and value of BCD number
171             mov     cx,dx                   ; Save count of digits before decimal point
172             cmp     al,'.'
173             jnz     test_for_digits
174     ;
175     ;         Count the number of digits appearing after the decimal point.
176     ;
177     scan_trailing_digits:
178
179             lodsb                           ; Look at next character
```

```
180             call    test_digit              ; Test for digit and bump counter
181             jnc     scan_trailing_digits
182     ;
183     ;         There must be at least one digit counted at this point.
184     ;
185     test_for_digits:
186
187             dec     si                      ; Put si back on terminating character
188             or      dx,dx                   ; Test digit count
189             jz      no_number_found         ; Jump if no digits were found
190
191             push    si                      ; Save pointer to terminator
192             dec     si                      ; Backup pointer to last digit
193     ;
194     ;         Check that the number will fit in the 18 digit BCD format.
195     ;       CX becomes the initial scaling factor to account for the implied
196     ;       decimal point.
197     ;
198             sub     cx,dx                   ; For each digit to the right of the
199                                             ; decimal point, subtract one from the
200                                             ; initial scaling power
201             neg     dx                      ; Use negative digit count so the
202                                             ; test_digit routine can count dx up
203                                             ; to zero
204             cmp     dx,-bcd_count           ; See if too many digits found
205             jb      test_for_unneeded_digits
206     ;
207     ;         Setup initial register values for scanning the number right to left
208     ;       while building the BCD value in memory.
209     ;
210     form_bcd_value:
211
212             std                             ; Set autodecrement mode
213             mov     power_ten,cx            ; Set initial power of ten
214             xor     di,di                   ; Clear BCD number index
215             mov     cl,4                    ; Set digit shift count
216             fwait                           ; Ensure BCD store is done
217             jmp     enter_digit_loop
218     ;
219     ;         No digits were encountered before testing for the exponent.
220     ;       Restore the string pointer and return an indefinite value.
221     ;
222     no_number_found:
223
224             mov     ax,NO_NUMBER            ; Set return status
225             fld     indefinite              ; Return an indefinite numeric value
226             jmp     exit
227     ;
228     ;         Test for a number of the form ???00000.
229     ;
230     test_terminating_point:
231
232             lodsb                           ; Get last character
233             cmp     al,'.'                  ; Look for decimal point
234             jz      enter_power_zeroes      ; Skip forward if found
235
236             inc     si                      ; Else bump pointer back
237             jmp     short enter_power_zeroes
238     ;
239     ;         Too many decimal digits encountered.  Attempt to remove leading and
240     ;       trailing digits to bring the total into the bounds of the BCD format.
241     ;
242     test_for_unneeded_digits:
243
244             std                             ; Set autodecrement mode
245             or      cx,cx                   ; See if any digits appeared to the
246                                             ; right of the decimal point
247             jz      test_terminating_point  ; Jump if none exist
248
249             dec     dx                      ; Adjust digit counter for loop
250     ;
251     ;         Scan backwards from the right skipping trailing zeroes.
252     ;       If the end of the number is encountered, dx=0, the string consists of
253     ;       all zeroes!
```

```
254     ;
255     skip_trailing_zeroes:
256
257             inc     dx                      ; Bump digit count
258             jz      look_for_exponent       ; Jump if string of zeroes found!
259
260             lodsb                           ; Get next character
261             inc     cx                      ; Bump power value for each trailing
262             cmp     al,'0'                  ; zero dropped
263             jz      skip_trailing_zeroes
264
265             dec     cx                      ; Adjust power counter from loop
266             cmp     al,'.'                  ; Look for decimal point
267             jnz     scan_leading_zeroes     ; Skip forward if none found
268
269             dec     dx                      ; Adjust counter for the decimal point
270     ;
271     ;         The string is of the form: ????.0000000
272     ;       See if any zeroes exist to the left of the decimal point.
273     ;
274     enter_power_zeroes:
275
276             dec     dx                      ; Adjust digit counter for loop
277
278     skip_power_zeroes:
279
280             inc     dx                      ; Bump digit count
281             jz      look_for_exponent
282
283             lodsb                           ; Get next character
284             inc     cx                      ; Bump power value for each trailing
285             cmp     al,'0'                  ; zero dropped
286             jz      skip_power_zeroes
287
288             dec     cx                      ; Adjust power counter from loop
289     ;
290     ;         Scan the leading digits from the left to see if they are zeroes.
291     ;
292     scan_leading_zeroes:
293
294             lea     di,byte ptr [si+1]      ; Save new end of number pointer
295             cld                             ; Set autoincrement mode
296             mov     si,string_ptr           ; Set pointer to the start
297             lodsb                           ; Look for sign character
298             cmp     al,'+'
299             je      skip_leading_zeroes
300
301             cmp     al,'-'
302             jne     enter_leading_zeroes
303     ;
304     ;         Drop leading zeroes.  None of them affect the power value in cx.
305     ;       We are guarenteed at least one non zero digit to terminate the loop.
306     ;
307     skip_leading_zeroes:
308
309             lodsb                           ; Get next character
310
311     enter_leading_zeroes:
312
313             inc     dx                      ; Bump digit count
314             cmp     al,'0'                  ; Look for a zero
315             jz      skip_leading_zeroes
316
317             dec     dx                      ; Adjust digit count from loop
318             cmp     al,'.'                  ; Look for 000.??? form
319             jnz     test_digit_count
320     ;
321     ;         Number is of the form 000.????
322     ;       Drop all leading zeroes with no effect on the power value.
323     ;
324     skip_middle_zeroes:
325
326             inc     dx                      ; Remove the digit
327             lodsb                           ; Get next character
```

```
328             cmp     al,'0'
329             jz      skip_middle_zeroes
330
331             dec     dx                      ; Adjust digit count from loop
332     ;
333     ;         All superflous zeroes are removed.  Check if all is well now.
334     ;
335     test_digit_count:
336
337             cmp     dx,-bcd_count
338             jb      too_many_digits_found
339
340             mov     si,di                   ; Restore string pointer
341             jmp     form_bcd_value
342
343     too_many_digits_found:
344
345             fld     indefinite              ; Set return numeric value
346             mov     ax,TOO_MANY_DIGITS      ; Set return flag
347             pop     si                      ; Get last address
348             jmp     exit
349     ;
350     ;         Build BCD form of the decimal ASCII string from right to left with
351     ;       trailing zeroes and decimal point removed.  Note that the only non
352     ;       digit possible is a decimal point which can be safely ignored.
353     ;       Test_digit will correctly count dx back towards zero to terminate
354     ;       the BCD build function.
355     ;
356     get_digit_loop:
357
358             lodsb                           ; Get next character
359             call    test_digit              ; Check if digit and bump digit count
360             jc      get_digit_loop          ; Skip the decimal point if found
361
362             shl     al,cl                   ; Put digit into high nibble
363             or      ah,al                   ; Form BCD byte in ah
364             mov     bcd_byte[di],ah         ; Put into BCD string
365             inc     di                      ; Bump BCD pointer
366             or      dx,dx                   ; Check if digit is available
367             jz      look_for_exponent
368
369     enter_digit_loop:
370
371             lodsb                           ; Get next character
372             call    test_digit              ; Check if digit
373             jc      enter_digit_loop        ; Skip the decimal point
374
375             mov     ah,al                   ; Save digit
376             or      dx,dx                   ; Check if digit is available
377             jnz     get_digit_loop
378
379             mov     bcd_byte[di],ah         ; Save last odd digit
380     ;
381     ;         Look for an exponent indicator.
382     ;
383     look_for_exponent:
384
385             pop     si                      ; Restore string pointer
386             cld                             ; Set autoincrement direction
387             mov     di,power_ten            ; Get current power of ten
388             lodsb                           ; Get next character
389             cmp     al,'e'                  ; Look for exponent indication
390             je      exponent_found
391
392             cmp     al,'E'
393             jne     convert
394     ;
395     ;         An exponent is expected, get its numeric value.
396     ;
397     exponent_found:
398
399             lodsb                           ; Get next character
400             xor     di,di                   ; Clear power variable
401             mov     cx,di                   ; Clear exponent sign flag and digit flag
```

```
402             cmp     al,'+'                  ; Test for positive sign
403             je      skip_power_sign
404
405             cmp     al,'-'                  ; Test for negative sign
406             jne     enter_power_loop
407     ;
408     ;         The exponent is negative.
409     ;
410             inc     ch                      ; Set exponent sign flag
411
412     skip_power_sign:
413     ;
414     ;         Register usage:
415     ;
416     ;               al:     exponent character being examined
417     ;               bx:     return value
418     ;               ch:     exponent sign flag      0 positive, 1 negative
419     ;               cl:     digit flag      0 no digits found, 1 digits found
420     ;               dx:     not usable since test_digit increments it
421     ;               si:     string pointer
422     ;               di:     binary value of exponent
423     ;
424     ;         Scan off exponent digits until a non-digit is encountered.
425     ;
426     power_loop:
427
428             lodsb                           ; Get next character
429
430     enter_power_loop:
431
432             mov     ah,0                    ; Clear ah since ax is added to later
433             call    test_digit              ; Test for a digit
434             jc      form_power_value        ; Exit loop if not
435
436             mov     cl,1                    ; Set power digit flag
437             sal     di,1                    ; old*2
438             add     ax,di                   ; old*2+digit
439             sal     di,1                    ; old*4
440             sal     di,1                    ; old*8
441             add     di,ax                   ; old*10+digit
442             cmp     di,HIGH_EXPONENT+bcd_count; Check if exponent is too big
443             jna     power_loop
444     ;
445     ;         The exponent is too large.
446     ;
447     exponent_ verflow:
448
449             mov     ax,EXPONENT_TOO_BIG     ; Set return value
450             fld     infinity                ; Return infinity
451             test    bcd_sign,bcd_sign_bit   ; Return correctly signed infinity
452             jz      exit                    ; Jump if not
453
454             fchs                            ; Return -infinity
455             jmp     short exit
456     ;
457     ;         No exponent was found.
458     ;
459     no_exponent_found:
460
461             dec     si                      ; Put si back on terminating character
462             mov     ax,NO_EXPONENT          ; Set return value
463             fld     indefinite              ; Set number to return
464             jmp     short exit
465     ;
466     ;         The string examination is complete.  Form the correct power of ten.
467     ;
468     form_power_value:
469
470             dec     si                      ; Backup string pointer to terminating
471                                             ; character
472             rcr     ch,1                    ; Test exponent sign flag
473             jnc     positive_exponent
474
475             neg     di                      ; Force exponent negative
```

```
476
477     positive_exponent:
478
479             rcr     cl,1                    ; Test exponent digit flag
480             jnc     no_exponent_found       ; If zero then no exponent digits were
481                                             ; found
482             add     di,power_ten            ; Form the final power of ten value
483             cmp     di,LOW_EXPONENT         ; Check if the value is in range
484             js      exponent_overflow       ; Jump if exponent is too small
485
486             cmp     di,HIGH_EXPONENT
487             jg      exponent_overflow
488
489             inc     si                      ; Adjust string pointer
490     ;
491     ;         Convert the base 10 number to base 2.
492     ;       Note:   10**exp = 2**(exp*log2(10))
493     ;
494     ;       di has binary power of ten value to scale the BCD value with.
495     ;
496     convert:
497
498             dec     si                      ; Bump string pointer back to last character
499             mov     ax,di                   ; Set power of ten to calculate
500             or      ax,ax                   ; Test for positive or negative value
501             js      get_negative_power
502     ;
503     ;         Scale the BCD value by a value >= 1.
504     ;
505             call    get_power_10            ; Get the adjustment power of ten
506             fbld    bcd_form                ; Get the digits to use
507             fmul                            ; Form converged result
508             jmp     short done
509     ;
510     ;         Calculate a power of ten value > 1 then divide the BCD value with
511     ;       it.  This technique is more exact than multiplying the BCD value by
512     ;       a fraction since no negative power of ten can be exactly represented
513     ;       in binary floating point.  Using this technique will guarentee exact
514     ;       conversion of values like .5 and .0625.
515     ;
516     get_negative_power:
517
518             neg     ax                      ; Force positive power
519             call    get_power_10            ; Get the adjustment power of ten
520             fbld    bcd_form                ; Get the digits to use
521             fdivr                           ; Divide fractions
522             fxch                            ; Negate scale factor
523             fchs
524             fxch
525     ;
526     ;         All done, set return values.
527     ;
528     done:
529
530             fscale                          ; Update exponent of the result
531             mov     ax,NUMBER_FOUND         ; Set return value
532             fstp    st(1)                   ; Remove the scale factor
533
534     exit:
535
536             mov     di,status_ptr           ; Set status of the conversion
537             mov     word ptr [di],ax
538             mov     di,end_ptr              ; Set ending string address
539             mov     word ptr [di],si
540             mov     sp,bp                   ; Deallocate local storage area
541             pop     bp                      ; Restore caller's environment
542             fwait                           ; Insure all loads from memory are done
543             ret     parms_size
544     ;
545     ;         Test if the character in al is an ASCII digit.
546     ;       If so then convert to binary, bump cx, and clear the carry flag.
547     ;       Else leave as is and set the carry flag.
```

```
548     ;
549     test_digit:
550             cmp     al,'9'                  ; See if a digit
551             ja      not_digit
552
553             cmp     al,'0'
554             jb      not_digit
555     ;
556     ;         Character is a digit.
557     ;
558             inc     dx                      ; Bump digit count
559             sub     al,'0'                  ; Convert to binary and clear carry flag
560             ret
561     ;
562     ;         Character is not a digit.
563     ;
564     not_digit:
565             stc                             ; Leave as is and set the carry flag
566             ret
567
568     ascii_to_floating endp
569     code            ends
570                     end

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

APPENDIX E

OVERVIEW

Appendix E contains three trigonometric functions for sine, cosine, and tangent. All accept a valid angle argument between -2^{62} and $+2^{62}$. They may be called from PLM/86, PASCAL/86, FORTRAN/86 or ASM/86 functions.

They use the partial tangent instruction together with trigonometric identities to calculate the result. They are accurate to within 16 units of the low 4 bits of an extended precision value. The functions are coded for speed and small size, with tradeoffs available for greater accuracy.

FPTAN and FPREM

These trigonometric functions use the FPTAN instruction of the NPX. FPTAN requires that the angle argument be between 0 and PI/4 radians, 0 to 45 degrees. The FPREM instruction is used to reduce the argument down to this range. The low three quotient bits set by FPREM identify which octant the original angle was in.

One FPREM instruction iteration can reduce angles of 10^{18} radians or less in magnitude to PI/4! Larger values can be reduced, but the meaning of the result is questionable since any errors in the least significant bits of that value represent changes of 45 degrees or more in the reduced angle.

Cosine Uses Sine Code

To save code space, the cosine function uses most of the sine function code. The relation sin (|A| + PI/2) = cos(A) is used to convert the cosine argument into a sine argument. Adding PI/2 to the angle is performed by adding 010_2 to the FPREM quotient bits identifying the argument's octant.

It would be very inaccurate to add PI/2 to the cosine argument if it was very much different from PI/2.

Depending on which octant the argument falls in, a different relation will be used in the sine and tangent functions. The program listings show which relations are used.

For the tangent function, the ratio produced by FPTAN will be directly evaluated. The sine function will use either a sine or cosine relation depending on which octant the angle fell into. On exit these functions will normally leave a divide instruction in progress to maintain concurrency.

If the input angles are of a restricted range, such as from 0 to 45 degrees, then considerable optimization is possible since full angle reduction and octant identification is not necessary.

All three functions begin by looking at the value given to them. Not a number (NAN), infinity, or empty registers must be specially treated. Unnormals need to be converted to normal values before the FPTAN instruction will work correctly. Denormals will be converted to very small unnormals which do work correctly for the FPTAN instruction. The sign of the angle is saved to control the sign of the result.

Within the functions, close attention was paid to maintain concurrent execution of the 8087 and host. The concurrent execution will effectively hide the execution time of the decision logic used in the program.

```
LINE     SOURCE

   1     $title(8087 Trignometric Functions)
   2
   3                     public          sine,cosine,tangent
   4                     name            trig_functions
   5
   6 +1  $include (:f1:8087.anc)
   7     ;
   8     ;          Define 8087 word packing in the environment area.
   9     ;
  10     cw_87           record  res871:3,infinity_control:1,rounding_control:2,
  11     &                       precision_control:2,error_enable:1,res872:1,
  12     &                       precision_mask:1,underflow_mask:1,overflow_mask:1,
  13     &                       zero_divide_mask:1,denormal_mask:1,invalid_mask:1
  14
  15     sw_87           record  busy:1,cond3:1,top:3,cond2:1,cond1:1,cond0:1,
  16     &                       error_pending:1,res873:1,precision_error:1,
  17     &                       underflow_error:1,overflow_error:1,zero_divide_error:1,
  18     &                       denormal_error:1,invalid_error:1
  19
  20     tw_87           record  reg7_tag:2,reg6_tag:2,reg5_tag:2,reg4_tag:2,
  21     &                       reg3_tag:2,reg2_tag:2,reg1_tag:2,reg0_tag:2
  22
  23     low_ip_87       record  low_ip:16
  24
  25     high_ip_op_87   record  hi_ip:4,res874:1,opcode_87:11
  26
  27     low_op_87       record  low_op:16
  28
  29     high_op_87      record  hi_op:4,res875:12
  30
  31     environment_87  struc                   ; 8087 environemnt layout
  32     env87_cw        dw              ?
  33     env87_sw        dw              ?
  34     env87_tw        dw              ?
  35     env87_low_ip    dw              ?
  36     env87_hip_op    dw              ?
  37     env87_low_op    dw              ?
  38     env87_hop       dw              ?
  39     environment_87  ends
  40     ;
  41     ;          Define 8087 related constants.
  42     ;
  43     TOP_VALUE_INC   equ             sw_87 <0,0,1,0,0,0,0,0,0,0,0,0,0,0,0>
  44
  45     VALID_TAG       equ             0       ; Tag register values
  46     ZERO_TAG        equ             1
  47     SPECIAL_TAG     equ             2
  48     EMPTY_TAG       equ             3
  49     REGISTER_MASK   equ             7
  50
  51     ;
  52     ;          Define local variable areas.
  53     ;
  54     stack           segment stack 'stack'
  55
  56     local_area      struc
  57     sw1             dw      ?               ; 8087 status value
  58     local_area      ends
  59
  60                     db      size local_area+4       ; Allocate stack space
  61     stack           ends
  62
  63     code            segment public 'code'
  64                     assume  cs:code,ss:stack
  65     ;
  66     ;          Define local constants.
  67     ;
  68     status          equ     [bp].sw1                ; 8087 status value location
  69
  70                     even
  71
  72     pi_quarter      dt      3FFEC90FDAA22168C235R   ; PI/4
```

```
 73     indefinite      dd      0FFC00000R              ; Indefinite special value
 74
 75     ;
 76     ;       This subroutine calculates the sine or cosine of the angle, given in
 77     ;       radians.  The angle is in ST(0), the returned value will be in ST(0).
 78     ;       The result is accurate to within 7 units of the least significant three
 79     ;       bits of the NPX extended real format.  The PLM/86 definition is:
 80     ;
 81     ;       sine:   procedure (angle) real external;
 82     ;               declare angle real;
 83     ;               end sine;
 84     ;
 85     ;       cosine: procedure (angle) real external;
 86     ;               declare angle real;
 87     ;               end cosine;
 88     ;
 89     ;       Three stack registers are required.  The result of the function is
 90     ;       defined as follows for the following arguments:
 91     ;
 92     ;                       angle                                   result
 93     ;
 94     ;               valid or unnormal less than 2**62 in magnitude  correct value
 95     ;               zero                                            0 or 1
 96     ;               denormal                                        correct denormal
 97     ;               valid or unnormal greater than 2**62            indefinite
 98     ;               infinity                                        indefinite
 99     ;               NAN                                             NAN
100     ;               empty                                           empty
101
102     ;
103     ;       This function is based on the NPX fptan instruction.  The fptan
104     ;       instruction will only work with an angle of from 0 to PI/4.  With this
105     ;       instruction, the sine or cosine of angles from 0 to PI/4 can be accurately
106     ;       calculated.  The technique used by this routine can calculate a general
107     ;       sine or cosine by using one of four possible operations:
108     ;
109     ;                       Let R = |angle mod PI/4|
110     ;                           S = -1 or 1, according to the sign of the angle
111     ;
112     ;       1) sin(R)       2) cos(R)       3) sin(PI/4-R)  4) cos(PI/4-R)
113     ;
114     ;       The choice of the relation and the sign of the result follows the
115     ;       decision table shown below based on the octant the angle falls in:
116     ;
117     ;               octant          sine            cosine
118     ;
119     ;                 0              S*1              2
120     ;                 1              S*4              3
121     ;                 2              S*2             -1*1
122     ;                 3              S*3             -1*4
123     ;                 4             -S*1             -1*2
124     ;                 5             -S*4             -1*3
125     ;                 6             -S*2               1
126     ;                 7             -S*3               4
127     ;
128
129     ;
130     ;       Angle to sine function is a zero or unnormal.
131     ;
132     sine_zero_unnormal:
133
134             fstp    st(1)                           ; Remove PI/4
135             jnz     enter_sine_normalize            ; Jump if angle is unnormal
136     ;
137     ;       Angle is a zero.
138     ;
139             pop     bp                              ; Return the zero as the result
140             ret
141     ;
142     ;       Angle is an unnormal.
143     ;
144     enter_sine_normalize:
145
```

```
146             call    normalize_value
147             jmp     short enter_sine
148
149     cosine  proc                                    ; Entry point to cosine
150
151             fxam                                    ; Look at the value
152             push    bp                              ; Establish stack addressibility
153             sub     sp,size local_area              ; Allocate stack space for status
154             mov     bp,sp
155             fstsw   status                          ; Store status value
156             fld     pi_quarter                      ; Setup for angle reduce
157             mov     cl,1                            ; Signal cosine function
158             pop     ax                              ; Get status value
159             lahf                                    ; ZF = C3, PF = C2, CF = C0
160             jc      funny_parameter                 ; Jump if parameter is
161                                                     ; empty, NAN, or infinity
162     ;
163     ;         Angle is unnormal, normal, zero, denormal.
164     ;
165             fxch                                    ; st(0) = angle, st(1) = PI/4
166             jpe     enter_sine                      ; Jump if normal or denormal
167     ;
168     ;         Angle is an unnormal or zero.
169     ;
170             fstp    st(1)                           ; Remove PI/4
171             jnz     enter_sine_normalize
172     ;
173     ;         Angle is a zero.  cos(0) = 1.0
174     ;
175             fstp    st(0)                           ; Remove 0
176             pop     bp                              ; Restore stack
177             fld1                                    ; Return 1
178             ret
179     ;
180     ;         All work is done as a sine function.  By adding PI/2 to the angle
181     ;       a cosine is converted to a sine.  Of course the angle addition is not
182     ;       done to the argument but rather to the program logic control values.
183     ;
184     sine:                                           ; Entry point for sine function
185
186             fxam                                    ; Look at the parameter
187             push    bp                              ; Establish stack addressibility
188             sub     sp,size local_area              ; Allocate local space
189             mov     bp,sp
190             fstsw   status                          ; Look at fxam status
191             fld     pi_quarter                      ; Get PI/4 value
192             pop     ax                              ; Get fxam status
193             lahf                                    ; CF = C0, PF = C2, ZF = C3
194             jc      funny_parameter                 ; Jump if empty, NAN, or infinity
195     ;
196     ;          Angle is unnormal, normal, zero, or denormal.
197     ;
198             fxch                                    ; ST(1) = PI/4, st(0) angle
199             mov     cl,0                            ; Signal sine
200             jpo     sine_zero_unnormal              ; Jump if zero or unnormal
201     ;
202     ;         ST(0) is either a normal or denormal value.  Both will work.
203     ;       Use the fprem instruction to accurately reduce the range of the given
204     ;       angle to within 0 and PI/4 in magnitude.  If fprem cannot reduce the
205     ;       angle in one shot, the angle is too big to be meaningful, > 2**62
206     ;       radians.  Any roundoff error in the calculation of the angle given
207     ;       could completely change the result of this function.  It is safest to
208     ;       call this very rare case an error.
209     ;
210     enter_sine:
211
212             fprem                                   ; Reduce angle
213                                                     ; Note that fprem will force a
214                                                     ; denormal to a very small unnormal
215                                                     ; Fptan of a very small unnormal
216                                                     ; will be the same very small
217                                                     ; unnormal, which is correct.
218             mov     sp,bp                           ; Allocate stack space for status
219             fstsw   status                          ; Check if reduction was complete
```

```
220                                             ; Quotient in C0,C3,C1
221             pop     bx                      ; Get fprem status
222             test    bh,high(mask cond2)     ; sin(2*N*PI+x) = sin(x)
223             jnz     angle_too_big
224     ;
225     ;         Set sign flags and test for which eighth of the revolution the
226     ;       angle fell into.
227     ;
228     ;       Assert: -PI/4 < st(0) < PI/4
229     ;
230             fabs                            ; Force the argument positive
231                                             ; cond1 bit in bx holds the sign
232             or      cl,cl                   ; Test for sine or cosine function
233             jz      sine_select             ; Jump if sine function
234     ;
235     ;         This is a cosine function.  Ignore the original sign of the angle
236     ;       and add a quarter revolution to the octant id from the fprem instruction.
237     ;       cos(A) = sin(A+PI/2) and cos(|A|) = cos(A)
238     ;
239             and     ah,not high(mask cond1) ; Turn off sign of argument
240             or      bh,high(mask busy)      ; Prepare to add 010 to C0,C3,C1
241                                             ; status value in ax
242                                             ; Set busy bit so carry out from
243             add     bh,high(mask cond3)     ; C3 will go into the carry flag
244             mov     al,0                    ; Extract carry flag
245             rcl     al,1                    ; Put carry flag in low bit
246             xor     bh,al                   ; Add carry to C0 not changing
247                                             ; C1 flag
248     ;
249     ;         See if the argument should be reversed, depending on the octant in
250     ;       which the argument fell during fprem.
251     ;
252     sine_select:
253
254             test    bh,high(mask cond1)     ; Reverse angle if C1 = 1
255             jz      no_sine_reverse
256     ;
257     ;         Angle was in octants 1,3,5,7.
258     ;
259             fsub                            ; Invert sense of rotation
260             jmp     short do_sine_fptan     ; 0 < arg <= PI/4
261     ;
262     ;         Angle was in octants 0,2,4,6.
263     ;       Test for a zero argument since fptan will not work if st(0) = 0
264     ;
265     no_sine_reverse:
266
267             ftst                            ; Test for zero angle
268             mov     sp,bp                   ; Allocate stack space
269             fstsw   status                  ; cond3 = 1 if st(0) = 0
270             fstp    st(1)                   ; Remove PI/4
271             pop     cx                      ; Get ftst status
272             test    ch,high(mask cond3)     ; If C3=1, argument is zero
273             jnz     sine_argument_zero
274     ;
275     ;       Assert: 0 < st(0) <= PI/4
276     ;
277     do_sine_fptan:
278
279             fptan                           ; TAN ST(0) = ST(1)/ST(0) = Y/X
280
281     after_sine_fptan:
282
283             pop     bp                      ; Restore stack
284             test    bh,high(mask cond3 + mask cond1); Look at octant angle fell into
285             jpo     X_numerator             ; Calculate cosine for octants
286                                             ; 1,2,5,6
287     ;
288     ;         Calculate the sine of the argument.
289     ;       sin(A) = tan(A)/sqrt(1+tan(A)**2)       if tan(A) = Y/X then
290     ;       sin(A) = Y/sqrt(X*X + Y*Y)
291     ;
292             fld     st(1)                   ; Copy Y value
293             jmp     short finish_sine       ; Put Y value in numerator
```

```
294     ;
295     ;         The top of the stack is either NAN, infinity, or empty.
296     ;
297     funny_parameter:
298
299             fstp    st(0)                           ; Remove PI/4
300             jz      return_empty                    ; Return empty if no parm
301
302             jpo     return_NAN                      ; Jump if st(0) is NAN
303     ;
304     ;         st(0) is infinity.  Return an indefinite value.
305     ;
306             fprem                                   ; ST(1) can be anything
307
308     return_NAN:
309     return_empty:
310
311             pop     bp                              ; Restore stack
312             ret                                     ; Ok to leave fprem running
313     ;
314     ;         Simulate fptan with st(0) = 0
315     ;
316     sine_argument_zero:
317
318             fldl                                    ; Simulate tan(0)
319             jmp     after_sine_fptan                ; Return the zero value
320     ;
321     ;         The angle was too large.  Remove the modulus and dividend from the
322     ;       stack and return an indefinite result.
323     ;
324     angle_too_big:
325
326             fcompp                                  ; Pop two values from the stack
327             fld     indefinite                      ; Return indefinite
328             pop     bp                              ; Restore stack
329             fwait                                   ; Wait for load to finish
330             ret
331     ;
332     ;         Calculate the cosine of the argument.
333     ;       cos(A) = 1/sqrt(1+tan(A)**2)    if tan(A) = Y/X then
334     ;       cos(A) = X/sqrt(X*X + Y*Y)
335     ;
336     X_numerator:
337
338             fld     st(0)                           ; Copy X value
339             fxch    st(2)                           ; Put X in numerator
340
341     finish_sine:
342
343             fmul    st,st(0)                        ; Form X*X + Y*Y
344             fxch
345             fmul    st,st(0)
346             fadd                                    ; st(0) = X*X + Y*Y
347             fsqrt                                   ; st(0) = sqrt(X*X + Y*Y)
348
349     ;
350     ;         Form the sign of the result.  The two conditions are the C1 flag from
351     ;       FXAM in bh and the C0 flag from fprem in ah.
352     ;
353             and     bh,high(mask cond0)             ; Look at the fprem C0 flag
354             and     ah,high(mask cond1)             ; Look at the fxam C1 flag
355             or      bh,ah                           ; Even number of flags cancel
356             jpe     positive_sine                   ; Two negatives make a positive
357
358             fchs                                    ; Force result negative
359
360     positive_sine:
361
362             fdiv                                    ; Form final result
363             ret                                     ; Ok to leave fdiv running
364
365     cosine  endp
366
```

```
367     ;
368     ;         This function will calculate the tangent of an angle.
369     ;       The angle, in radians is passed in ST(0), the tangent is returned
370     ;       in ST(0).  The tangent is calculated to an accuracy of 4 units in the
371     ;       least three significant bits of an extended real format number.  The
372     ;       PLM/86 calling format is:
373     ;
374     ;    tangent:  procedure (angle) real external;
375     ;              declare angle real;
376     ;              end tangent;
377     ;
378     ;         Two stack registers are used.  The result of the tangent function is
379     ;       defined for the following cases:
380     ;
381     ;                                  angle                              result
382     ;
383     ;              valid or unnormal < 2**62 in magnitude          correct value
384     ;              0                                               0
385     ;              denormal                                        correct denormal
386     ;              valid or unnormal > 2**62 in magnitude          indefinite
387     ;              NAN                                             NAN
388     ;              infinity                                        indefinite
389     ;              empty                                           empty
390     ;
391     ;         The tangent instruction uses the fptan instruction.  Four possible
392     ;       relations are used:
393     ;
394     ;         Let R = |angle MOD PI/4|
395     ;             S = -1 or 1 depending on the sign of the angle
396     ;
397     ;       1) tan(R)       2) tan(PI/4-R)  3) 1/tan(R)     4) 1/tan(PI/4-R)
398     ;
399     ;         The following table is used to decide which relation to use depending
400     ;       on in which octant the angle fell.
401     ;
402     ;       octant          relation
403     ;
404     ;         0              S*1
405     ;         1              S*4
406     ;         2             -S*3
407     ;         3             -S*2
408     ;         4              S*1
409     ;         5              S*4
410     ;         6             -S*3
411     ;         7             -S*2
412     ;
413     tangent proc
414
415             fxam                                    ; Look at the parameter
416             push    bp                              ; Establish stack addressibility
417             sub     sp,size local_area              ; Allocate local variable space
418             mov     bp,sp
419             fstsw   status                          ; Get fxam status
420             fld     pi_quarter                      ; Get PI/4
421             pop     ax
422             lahf                                    ; CF = C0, PF = C2, ZF = C3
423             jc      funny_parameter
424     ;
425     ;         Angle is unnormal, normal, zero, or denormal.
426     ;
427             fxch                                    ; st(0) = angle, st(1) = PI/4
428             jpe     tan_zero_unnormal
429     ;
430     ;         Angle is either an normal or denormal.
431     ;       Reduce the angle to the range -PI/4 < result < PI/4.
432     ;       If fprem cannot perform this operation in one try, the magnitude of the
433     ;       angle must be > 2**62.  Such an angle is so large that any rounding
434     ;       errors could make a very large difference in the reduced angle.
435     ;       It is safest to call this very rare case an error.
436     ;
437     tan_normal:
438
439             fprem                                   ; Quotient in C0,C3,C1
440                                                     ; Convert denormals into unnormals
```

```
441             mov     sp,bp                           ; Allocate stack spce
442             fstsw   status                          ; Quotient identifies octant
443                                                     ; original angle fell into
444             pop     bx                              ; tan(PI*N+x) = tan(x)
445             test    bh,high(mask cond2)             ; Test for complete reduction
446             jnz     angle_too_big                   ; Exit if angle was too big
447     ;
448     ;         See if the angle must be reversed.
449     ;
450     ;       Assert: -PI/4 < st(0) < PI/4
451     ;
452             fabs                                    ; 0 <= st(0) < PI/4
453                                                     ; C1 in bx has the sign flag
454             test    bh,high(mask cond1)             ; must be reversed
455             jz      no_tan_reverse
456     ;
457     ;         Angle fell in octants 1,3,5,7.  Reverse it, subtract it from PI/4.
458     ;
459             fsub                                    ; Reverse angle
460             jmp     short do_tangent
461     ;
462     ;         Angle is either zero or an unnormal.
463     ;
464     tan_zero_unnormal:
465
466             fstp    st(1)                           ; Remove PI/4
467             jz      tan_angle_zero
468     ;
469     ;         Angle is an unnormal.
470     ;
471             call    normalize_value
472             jmp     tan_normal
473
474     tan_angle_zero:
475
476             pop     bp                              ; Restore stack
477             ret
478     ;
479     ;         Angle fell in octants 0,2,4,6.  Test for st(0) = 0, fptan won't work.
480     ;
481     no_tan_reverse:
482
483             ftst                                    ; Test for zero angle
484             mov     sp,bp                           ; Allocate stack space
485             fstsw   status                          ; C3 = 1 if st(0) = 0
486             fstp    st(1)                           ; Remove PI/4
487             pop     cx                              ; Get ftst status
488             test    ch,high(mask cond3)
489             jnz     tan_zero
490
491     do_tangent:
492
493             fptan                                   ; tan ST(0) = ST(1)/ST(0)
494
495     after_tangent:
496     ;
497     ;         Decide on the order of the operands and their sign for the divide
498     ;       operation while the fptan instruction is working.
499     ;
500             pop     bp                              ; Restore stack
501             mov     al,bh                           ; Get a copy of fprem C3 flag
502             and     ax,mask cond1 + high(mask cond3); Examine fprem C3 flag and
503                                                     ; fxtract C1 flag
504             test    bh,high(mask cond1 + mask cond3); Use reverse divide if in
505                                                     ; octants 1,2,5,6
506             jpo     reverse_divide                  ; Note! parity works on low
507                                                     ;  8 bits only!
508     ;
509     ;         Angle was in octants 0,3,4,7.
510     ;       Test for the sign of the result.  Two negatives cancel.
511     ;
512             or      al,ah
513             jpe     positive_divide
```

```
514
515             fchs                                    ; Force result negative
516
517     positive_divide:
518
519             fdiv                                    ; Form result
520             ret                                     ; Ok to leave fdiv running
521
522     tan_zero:
523
524             fldl                                    ; Force 1/0 = tan(PI/2)
525             jmp     after_tangent
526     ;
527     ;         Angle was in octants 1,2,5,6.
528     ;       Set the correct sign of the result.
529     ;
530     reverse_divide:
531
532             or      al,ah
533             jpe     positive_r_divide
534
535             fchs                                    ; Force result negative
536
537     positive_r_divide:
538
539             fdivr                                   ; Form reciprocal of result
540             ret                                     ; Ok to leave fdiv running
541
542     tangent endp
543     ;
544     ;         This function will normalize the value in st(0).
545     ;       Then PI/4 is placed into st(1).
546     ;
547     normalize_value:
548
549             fabs                                    ; Force value positive
550             fxtract                                 ; 0 <= st(0) < 1
551             fldl                                    ; Get normalize bit
552             fadd    st(1),st                        ; Normalize fraction
553             fsub                                    ; Restore original value
554             fscale                                  ; Form original normalized value
555             fstp    st(1)                           ; Remove scale factor
556             fld     pi_quarter                      ; Get PI/4
557             fxch
558             ret
559
560     code    ends
561             end
```

```
ASSEMBLY COMPLETE, NO ERRORS FOUND
```

intel

APPLICATION NOTE

AP-122

September 1981

Hard Disk Controller Design Using the Intel® 8089

Hal Kop
Microcomputer Applications

210202-001

I. INTRODUCTION

This application note describes the design of a disk controller for a Shugart SA4008 Winchester disk drive. An 8089 I/O processor is used to offload many of the disk control overhead tasks from the host processor. The intelligent controller maximizes system throughput by performing the disk control tasks concurrently with data processing by the host processor. The features of the 8089 I/O processor which make it ideal for disk control applications are also described.

As newer microprocessors provide more throughput and address more memory, larger and more complex microprocessor based applications are designed. Many of these applications require high performance and high capacity mass storage devices such as hard disk drives. Winchester-technology (filtered air system and non-removable platters) disk drives are cost and performance compatible with high performance microprocessors. These drives provide more performance and reliability than floppy disk drives yet are less expensive than removable platter disk drives of comparable performance.

For applications requiring high performance disk drives, a major task of the system designer is the design of the disk controller—the interface between the high performance processor and disk drive. The conventional approach (Fig. 1) is to develop specialized control circuitry which interfaces the disk drive to the host processor's system bus. The host has complete control over the disk drive and executes a separate command sequence for each function—such as seek, format or read data. The host is assisted by a DMA (direct memory access) controller which performs the high speed transfers of read or write data between the drive interface and the system memory. Any error processing, such as CRC (cyclic redundancy check) error checking and initiating retries, is also performed by the host processor. A major disadvantage of this approach is that a large portion of the host's time and bus bandwidth is consumed by disk control overhead (command execution, interrupt servicing, and error processing) leaving little time for data processing.

A better approach is to partition the system functions and implement an intelligent disk controller which would perform the overhead tasks and free more host processor time for data processing. This intelligent controller would be able to accept a single high level command and perform multiple functions such as seek, read data, and process errors. Here the host has more time for data processing since it generates one high level command rather than several simple commands. It also services only one interrupt at the completion of the high level command rather than several.

The system configuration of an intelligent disk controller based on the Intel 8089 I/O processor is shown in

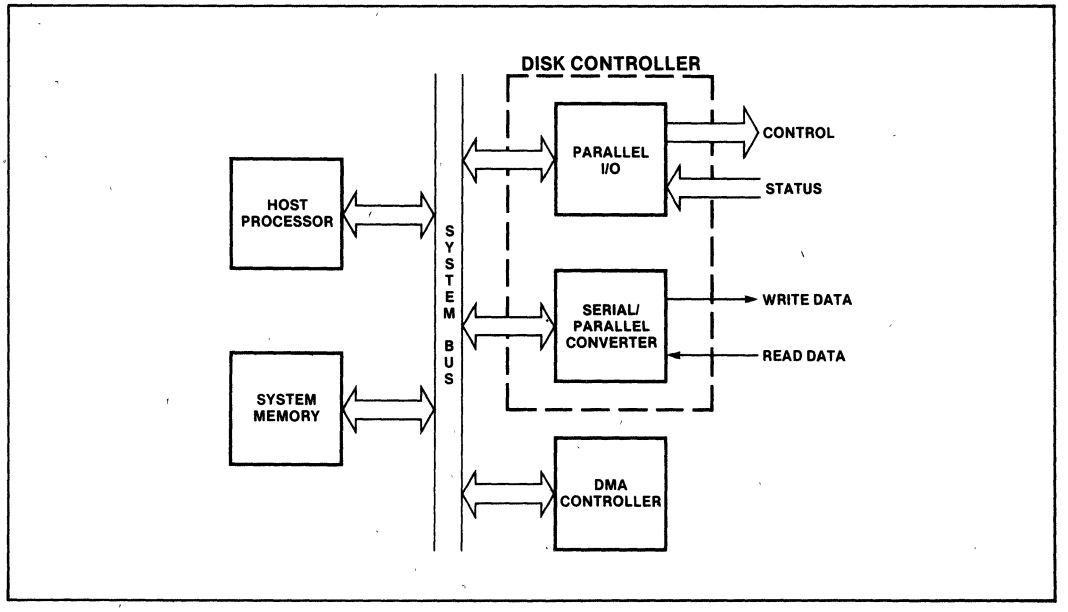


Figure 1. Conventional Disk Controller System Configuration

AFN02057A

Figure 2 where it is used in conjunction with an 8086 CPU as the host processor. This type of system configuration is called the iAPX 86/11 since it contains an 8086 and an 8089. The 8089 I/O processor is ideal for implementing an intelligent controller since it provides processing capabilities well suited for controlling a disk drive and high speed DMA transfers for moving data to and from the disk drive. The 8089 also supports a private local bus which provides access to the drive control circuitry, program memory, and local data buffers. This minimizes access to the shared system bus and hence increases overall system throughput. It will be seen later that local data buffering allows:

— high speed burst transfers without overrun and underrun errors

— disk controller operation at lower system bus priority than the host to maximize host processing

— error detection and retries directly by the disk controller without host intervention

The 8089-based disk controller maximizes system throughput. The disk control overhead tasks are offloaded from the host and performed by the 8089. This frees host processor time for data processing and other control processing. Host processor performance is reduced when both the host and 8089 try to access the system bus at the same time. These system bus conflicts can only occur when the 8089 accesses the system bus—during the accessing of memory-based communication blocks (used for transferring command and status information) and during sector data transfers between the system memory buffer and the 8089's local data buffer. For a single drive, this can mean host processor performance degradation of no more than 3%. With the conventional approach of Figure 1, the degradation can approach 10% due to CPU overhead time to control the disk operation and system bus time used by the DMA controller. Thus the 8089-based controller allows significantly more processing by the host, especially when multiple drives are supported.

This application note describes how basic disk control functions are implemented with an 8089. Therefore, the design described here does not exhibit all features possible in an intelligent controller. However, the hardware design allows the software to be easily enhanced to provide extra features. A later section addresses software enhancements.

The application note begins with an overview of the 8089 I/O processor followed by a brief description of the SA4008 drive. Next a discussion of the implemented functions is provided. A detailed description of the hardware and software design is then presented. Finally, a discussion of possible enhancements concludes the note.

Additional information related to topics discussed in this application note can be found in the following Intel documents:

The 8086 Family User's Manual

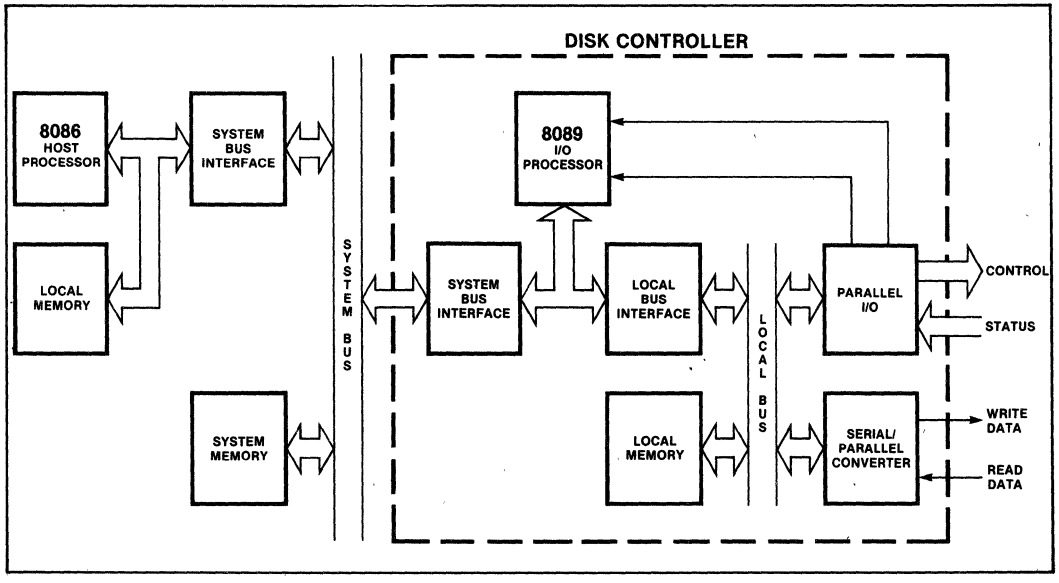


Figure 2. Intelligent Disk Controller System Configuration (iAPX 86/11)

AFN02057A

Intel Multibus Specification

iSBC 86/12A Hardware Reference Manual

iSBC 604/614 Cardcage Hardware Reference Manual

ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users

RBF-89 Real-Time Breakpoint Facility Operating Instructions for ICE-86 In-Circuit Emulator Users

8089 Macro Assembler User's Guide

In addition, the following documents from Shugart Associates provides detailed information on the disk drive:

SA4000 Fixed Disk Drive OEM Manual

SA4000 Fixed Disk Drive Service Manual

II. INTEL® 8089 I/O PROCESSOR

This section briefly describes the 8089 I/O processor's features and modes of operation. A more detailed discussion can be found in *The 8086 Family User's Manual* (October 1979).

A block diagram of the 8089 I/O processor is shown in Figure 3. The 8089 provides two independent channels. Both channels can execute task program instructions and perform high speed DMA transfers. Each channel has its own register set to support these operations.

A channel starts operation by executing task program instructions. These instructions are conceptually similar to instructions of other microprocessors but are typically executed to prepare the channel and I/O device for DMA transfers. Execution of the XFER (transfer) instruction switches the channel from instruction execution mode to DMA mode and high speed data transfer cycles are performed. When the DMA transfer terminates, task program instruction execution resumes for any post-DMA processing (e.g., status analysis, error processing, etc.). One channel or two channels may be operating at any given time. When two channels are active, they operate in a time-multiplexed manner sharing a common multiplexed address/data bus. A flexible priority structure allows both channels to operate with equal priorities or either channel to operate at a higher priority.

The 8089's bus structure and timing are identical with other members of the iAPX 86 and iAPX 88 families, such as the 8086 CPU and 8087 numeric processor extension. This allows the bipolar support circuits of the iAPX 86 and 88 families (8284A clock generator, 8288 bus controller, 8289 bus arbiter, etc.) to be used with the 8089. The 8089 generates 20 address signals and, depending on how it is initialized, supports an 8- or 16-bit data bus. This provides compatibility with the 16-bit 8086 CPU or the 8-bit 8088 CPU.

Both channels can access a 1 megabyte system address space and a 64 kilobyte local address space. Each address space accommodates both memory and I/O devices. This allows task program execution, memory data access, and I/O device access in both system and local address spaces. Task program and DMA access of the two address spaces is discussed later.

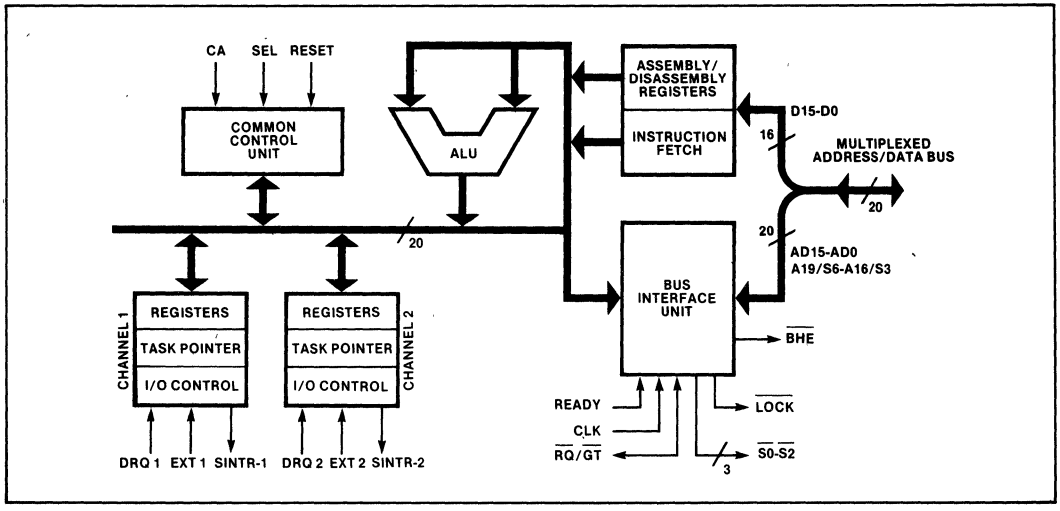


Figure 3. 8089 I/O Processor Block Diagram

AFN02057A

The 8288 bus controller used with the 8089 provides separate command signals for each address space. The bus controller's memory read and write commands provide access to the system address space and the I/O read and write commands are used to access the local address space. Separate commands for each address space allow two external buses to be implemented which promotes concurrent processing between the 8089 and the host processor and increases system throughput.

In addition, the 8089 allows these physical buses to be either 8 or 16 bits wide. During the 8089's initialization sequence, the widths of the system and local buses are defined. Although the 8089 supports two buses, a single bus may be used which is shared with a CPU. This will be described later when local and remote mode configurations are discussed.

The interface signals used to communicate with a host processor are also shown in Figure 3. The channel attention (CA) and select (SEL) input signals are used to start channel operation. Both signals are activated simultaneously by the host. SEL selects channel 1 or channel 2 (0 or 1, respectively). The SINTR1 and SINTR2 output signals are used to interrupt the host processor. One of these signals is activated whenever the set interrupt instruction, SINTR, is executed. SINTR1 is activated by channel 1 and SINTR2 by channel 2. The memory-based communication structure used to transfer command and status information between the 8089 and the host processor is discussed in a later section.

System Configurations

Systems using the 8089 may be configured in one of two different ways—local mode or remote mode. In the local configuration, the 8089 provides capabilities of an intelligent DMA controller for a single CPU. In the remote configuration, the 8089 provides capabilities of a control processor and a DMA controller and can operate concurrently with one or more host processors.

Local Mode Configuration

In the local mode configuration, the 8089 resides on the same local bus as an 8086 or 8088 CPU and shares the clock generator, address latches, data transceivers, and bus controller with the CPU. An example of a local mode iAPX 88/11 (8088 CPU and 8089 I/O processor) configuration is shown in Figure 4.

The 8089 is a slave to the CPU in local mode configurations and access to the shared bus is controlled by the bidirectional request/grant ($\overline{RQ}/\overline{GT}$) line. The CPU has possession of the bus when system operation begins. Whenever the 8089 needs access to the bus, it signals the CPU of this need by pulsing the $\overline{RQ}/\overline{GT}$ line. The CPU may be presently accessing the bus. As soon as the CPU is finished with the bus, it pulses the $\overline{RQ}/\overline{GT}$ line. The 8089 receives this grant pulse and accesses the bus. When the 8089 is finished using the bus, the 8089 pulses the $\overline{RQ}/\overline{GT}$ line to notify the CPU that it has released the bus.

Once the 8089 acquires the bus, it retains bus possession until finished. The request/grant protocol provides no

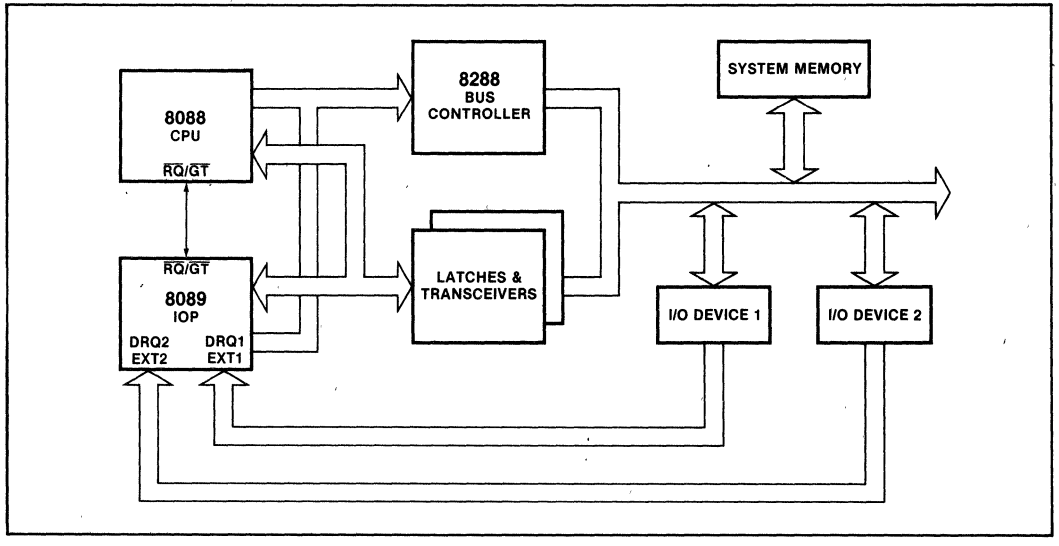


Figure 4. Typical Local Mode Configuration (iAPX 88/11)

mechanism for the CPU to regain the bus from the 8089. Care should be used when selecting this configuration since frequent or lengthy periods of 8089 activity can limit the CPU's use of the bus. However, the local mode configuration is an economical technique for adding intelligent, high speed DMA transfer capabilities to the system.

In local mode configurations, the 8089's 1 megabyte system address space coincides with the CPU's memory address space and the 64 kilobyte local address space coincides with the CPU's I/O address space. This means that when the 8089 accesses its system space or when the CPU accesses its memory space, the 8288 bus controller's memory read or write command is activated. When the 8089 accesses its local space or when the CPU accesses its I/O space, the bus controller's I/O read or write command is activated.

The 8089's physical data bus widths must be defined the same as the CPU's during the initialization sequence (to be discussed later) in local mode configurations. With an 8088 CPU the 8089's system and local physical bus widths must be initialized as 8 bits. When used with an 8086 CPU, both buses must be initialized as 16 bits.

Although the 8089 can execute programs and access memory and I/O devices from its two address spaces, several rules should be followed to ensure compatibility with the CPU. Data memory that is shared with the CPU must be accessed in the 8089's system address space. I/O devices which are accessed by the CPU in its I/O address space must be accessed in the 8089's local address space. Other memory and I/O devices accessed by the 8089 only may reside in either the 8089's system or local address space.

Remote Mode Configuration

In the remote mode configuration, a shared system bus with memory provides communications between the host processor and the 8089 I/O processor (Fig. 5). The 8089 supports two independent externally-implemented physical buses (Fig. 6). One bus is the shared system bus and the other is a private local bus. The system bus interface contains address latches, data transceivers, a bus controller, and a bus arbiter. The host processor uses an identical interface to access the system bus. The 8289 bus arbiter controls access to the system bus and is responsible for acquiring and surrendering the bus based on system priorities. The local bus interface contains address latches and data transceivers (if required by loading conditions).

The 8089's 1 megabyte system address space is used to access the shared system bus and the 64 kilobyte local address space is used to access the private local bus. A single 8288 bus controller provides command signals for both the system and local buses. The memory read and write commands are used to access both memory and I/O devices on the system bus. The I/O read and write commands are used when accessing memory or I/O devices on the local bus.

The physical widths of the system and local buses may be 8 or 16 bits. The widths are defined during the initialization sequence (to be discussed later). All four bus width combinations are available:

8-bit system bus and 8-bit local bus

8-bit system bus and 16-bit local bus

16-bit system bus and 8-bit local bus

16-bit system bus and 16-bit local bus

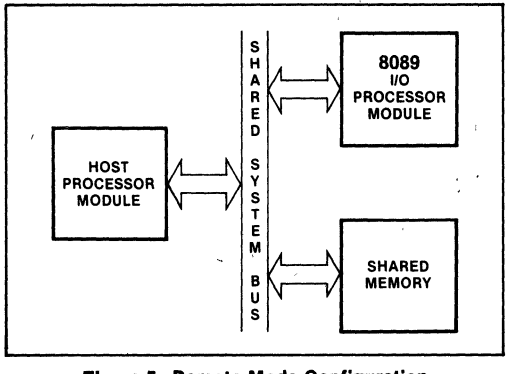


Figure 5. Remote Mode Configuration

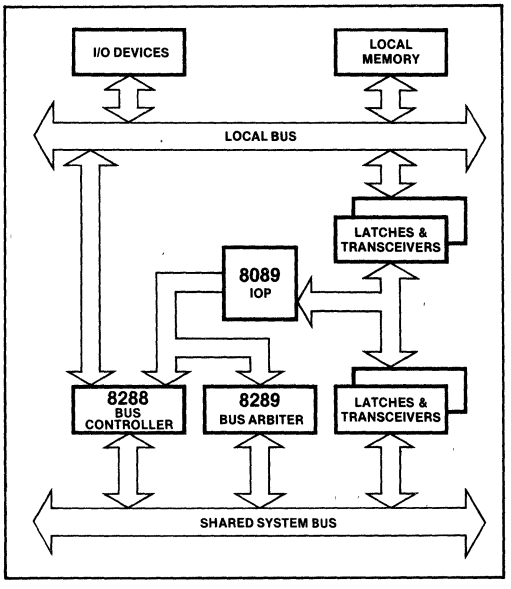


Figure 6. Typical 8089 I/O Processor Module (Remote Mote)

AFN02057A

The system bus width is typically established by the host processor. A 16-bit system bus is usually used with a 16-bit host while an 8-bit bus with an 8-bit host. The local bus width is typically selected based on the peripheral devices supported—8-bit bus with 8-bit peripherals and 16-bit bus with 16-bit peripherals. A 16-bit local bus is selected when both 8- and 16-bit peripherals are supported since it allows task program and DMA accessing of both 8- and 16-bit I/O devices. DMA capabilities are discussed later. Memory devices are configured so that the width of the memory's data path is the same as the physical bus.

Communications With Host Processor

Communications between the host processor and the 8089 I/O processor are primarily through shared memory. The hardwired signals (CA and SEL to the 8089 and SINTR1 and SINTR2 from the 8089) are used as startup and interrupt signals. Memory-based communication is implemented through a series of five linked control blocks (Fig. 7). This feature provides a very flexible communication structure and allows the 8089 to handle a wide variety of I/O functions.

The first three linked blocks in the communication structure are used during the 8089's initialization sequence (Fig. 8). The system configuration pointer (SCP) and system configuration block (SCB) are used only during initialization. Initialization is required after a RESET signal is received by the 8089. When the first channel attention after reset is received, the initialization sequence begins and the 8089 reads the data in the system configuration pointer. The parameter SYSBUS defines the physical width of the system bus (8 or 16 bits). The SCB offset and segment base point to the second block, the system configuration block (SCB). The 8089 next reads the data in the SCB. The SOC parameter defines the local bus's physical width and request/grant mode (refer to *The 8086 Family User's Manual*). The CB offset and segment base point to the channel control block (CB). The 8089 clears (zeros) channel 1's BUSY byte in the CB which completes the initialization sequence. With subsequent channel attentions, the 8089 directly accesses the CB as described below.

The SCP, SCB and CB must reside in shared memory since both the host and the 8089 access them. The SCP must begin at 0FFFF6H while SCB and CB locations are user-defined. The SCP is typically located in ROM while the SCB and CB are in RAM. With the SCP in ROM, the SCB's location remains fixed once defined. Since each 8089 must have a unique CB, the SCB (which points to the 8089's CB) must be placed in RAM if multiple 8089's exist in the system. This allows each 8089 to be initialized and directed to its own CB. The CB must be in RAM since certain parameters are updated during 8089 operation (e.g., BUSY byte).

The channel control, parameter and task blocks are used whenever the host starts channel operation. The host initializes certain parameters in the channel control and parameter blocks before generating the channel attention signal. When the 8089 receives a channel attention signal, the proper half of the CB is accessed depending on the value of SEL (0 for channel 1 and 1 for channel 2). The CCW (channel control word) instructs the selected channel what action to perform,

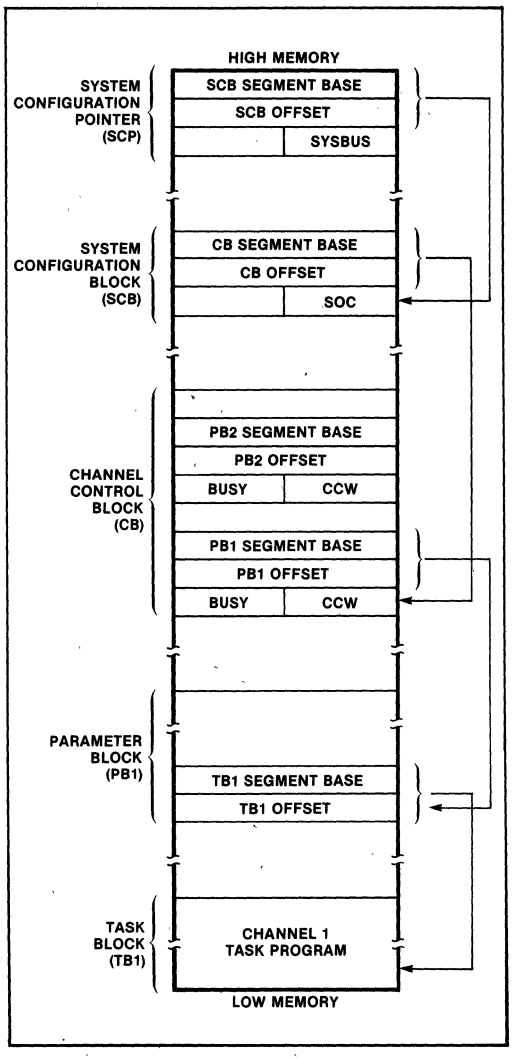


Figure 7. Memory Based Communication Blocks

AFN02057A

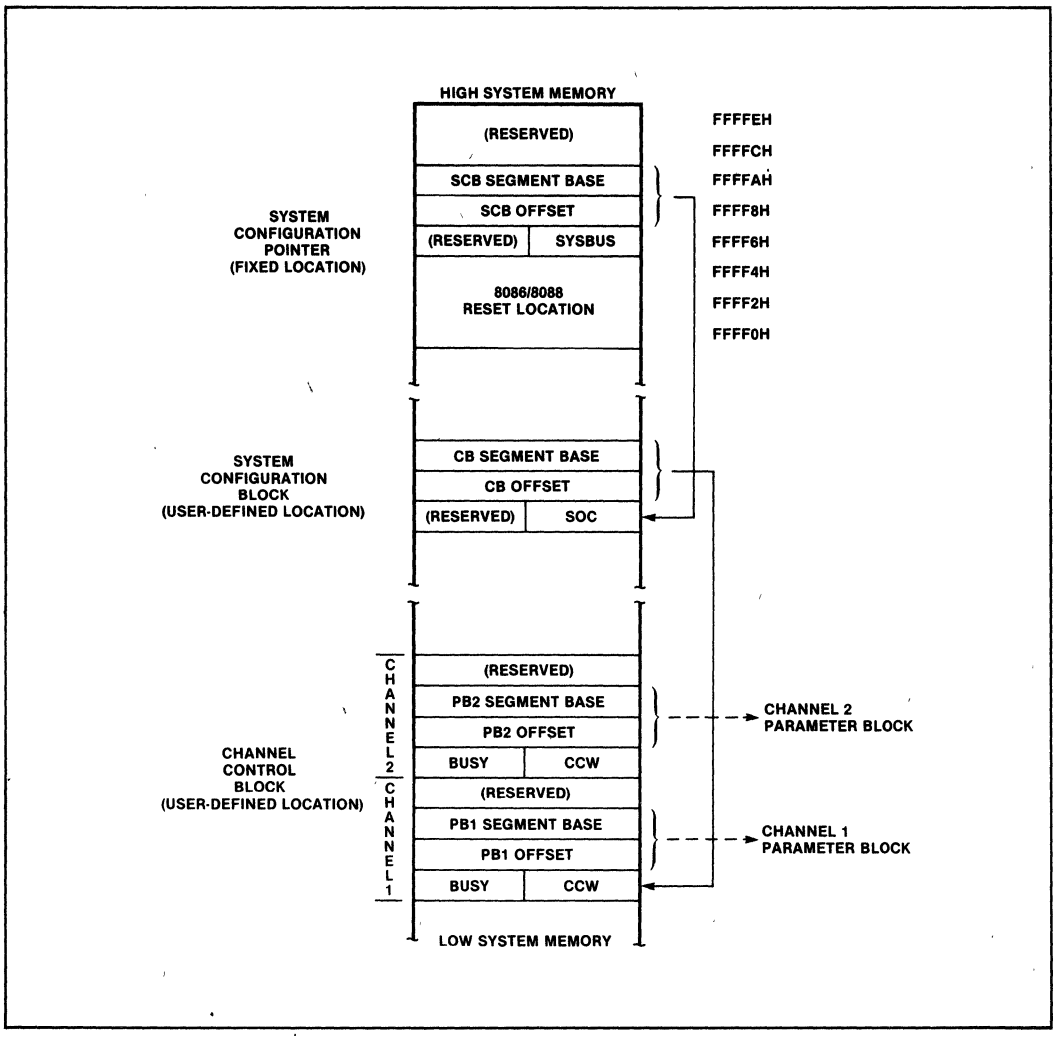


Figure 8. Initialization Control Blocks

such as start, suspend, resume, or halt task program execution. The BUSY byte is set to 0FFH by the 8089 if task program execution is started or resumed. The 8089 clears it to 0H if task program execution is suspended or halted. Within the channel control block, PB1 offset and segment base point to the parameter block for channel 1 and PB2 offset and segment base point to the parameter block for channel 2. From the proper parameter block, the 8089 reads the task block (TB1 or TB2) offset and segment base which point to the task program. The task block address must be the first two words of the parameter block. All other parameters in the PB are user-defined allowing parameters to be tailored to a specific I/O task.

Of all the five linked blocks, only the task block may reside either in system or local memory. For remote mode configurations, the task block typically resides in local memoty to obtain maximum system performance. However, executing task programs from system memory is advantageous for initial debugging or for executing a task program that downloads another task program from system memory to local RAM.

DMA Capabilities

The 8089's high speed DMA capability is ideal for disk controller applications. The maximum DMA transfer rate with a 5 MHz clock is 1.25 megabytes/sec. Conventional DMA controllers use a single bus cycle and gate data from the source device (memory or I/O) to the destination device. However, the 8089's DMA transfer uses two bus cycles. The first bus cycle reads the data from the source device and the second bus cycle writes the data to the destination device. The advantages of two cycle DMA are discussed in a later section.

All possible combinations of source and destination device specifications are available. Both source and destination may be memory or an I/O device. This means that memory to memory, I/O to I/O, and memory to or from I/O DMA transfers are available. In addition, DMA transfers between system and local address spaces or within the same address space can be specified.

Both memory and I/O devices (source and destination) are specified as addresses either in the system or local address space. These address values are loaded into source and destination pointer registers. After each word or byte is transferred, a register used as a memory pointer is incremented by one for byte transfers or by two for word transfers. A register used as an I/O device pointer is not modified. Registers used as DMA memory pointers are incremented only. No provisions exist for decrementing memory pointer registers during DMA.

DMA Synchronization

To accommodate a wide range of I/O device transfer rates, the 8089 allows DMA transfers to be synchronized. Each byte or word is transferred between the I/O device and the 8089 upon receiving a DMA request synchronizing signal from the I/O device. Each channel has a DMA request input: DRQ1 for channel 1 and DRQ2 for channel 2. Three options exist when specifying DMA transfer synchronization. DMA transfers may be source synchronized, destination synchronized, or unsynchronized.

During source synchronized DMA transfers, the channel waits until the DMA request input is activated by the source device before reading the data. External circuitry decodes the source device's address and provides a DMA acknowledge signal to the source device allowing it to deactivate the DMA request signal. Immediately after reading the data, the 8089 writes it to the destination device. The next read and write cycles begin when the source device activates DMA request again. Source synchronized DMA transfers are typically used when transferring data from an I/O device to memory.

During destination synchronized DMA transfers, the channel reads the data from the source device and waits for the DMA request signal before writing the data to the destination device. Similar to source synchronization, the DMA acknowledge signal is generated by decoding the destination device's address. This type of synchronization is commonly used when transferring data from memory to an I/O device.

The final synchronization option is to specify no synchronization. Here the DMA request input is not examined and the channel transfers data without waiting for a request. This specification is usually reserved for memory to memory transfers. The channel runs at full memory speed. Wait states may be used when accessing slow memory devices or when waiting to access the shared system bus.

DMA latency is the time required for the 8089 to respond to a DMA request; i.e., the time from DMA request signal activation until the synchronized bus cycle begins. DMA latency is due to DMA request propagation through internal pipelined control circuitry. The maximum DMA latency time when one channel is active and waiting for DMA request is 6 clocks. When both channels are active, the latency time may be up to 12 clocks.

Due to DMA latency, the DMA request signal cannot be used to synchronize transfers when the transfer rate of the I/O device is close (greater than 0.7 megabytes/sec when one channel is active) to the maximum transfer rate of the 8089, 1.25 megabytes/sec. For this case, wait states may be used to synchronize transfers. Since hard disk drives are in this category, the disk controller described in this application note uses wait states to synchronize disk transfers.

Advantages of Two Cycle DMA

The two bus cycle implementation of DMA transfers allows enhanced DMA capabilities. Data transfers between source and destination devices with different data widths may be specified. For example (Fig. 9), a DMA transfer cycle from an 8-bit I/O device to 16-bit memory is accomplished by reading two bytes from the I/O device (two bus cycles), assembling the bytes into a word, and then writing a single word into memory (one bus cycle). Here buses are accessed efficiently since three bus cycles are required as compared to four bus cycles if a single byte at a time were read and written. In the same example, since the 16-bit memory resides on the shared system bus, 50% fewer system bus accesses are required and overall system throughput may be increased.

Use of this bus matching DMA feature involves specifying logical DMA source and destination bus widths with

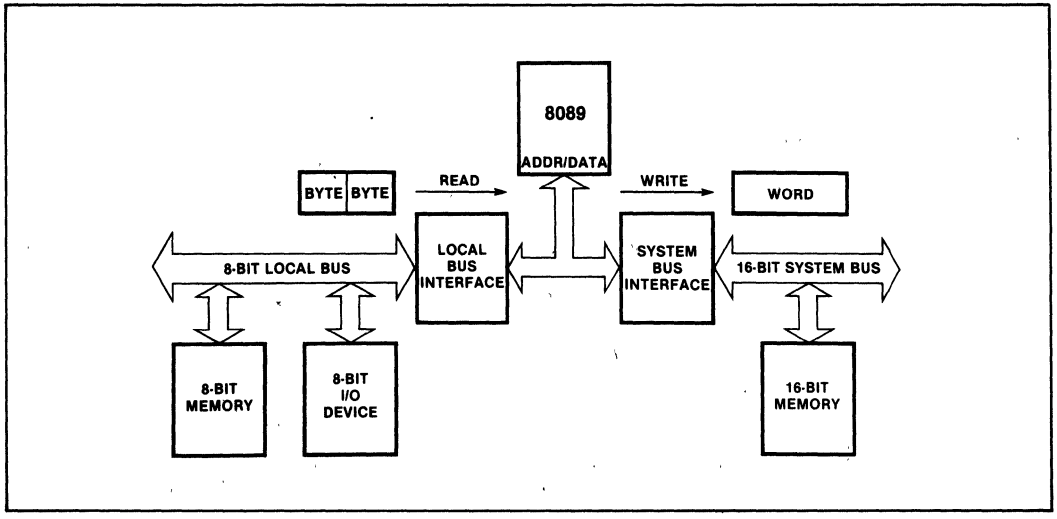


Figure 9. 8-Bit I/O to 16-Bit Memory DMA

the WID instruction. This allows DMA transfers to or from 8-bit I/O devices which reside on a 16-bit bus. The only restriction is that the logical bus width may not exceed the physical width. Thus 8- or 16-bit transfers may be performed with a 16-bit bus while only 8-bit transfers are permitted with an 8-bit bus (Fig. 10). Synchronized DMA transfers between dissimilar width logical buses may have more than one synchronized bus cycle. For example, destination synchronized transfers from 16-bit memory to an 8-bit I/O device perform two synchronized 8-bit write bus cycles for each 16-bit fetch from memory.

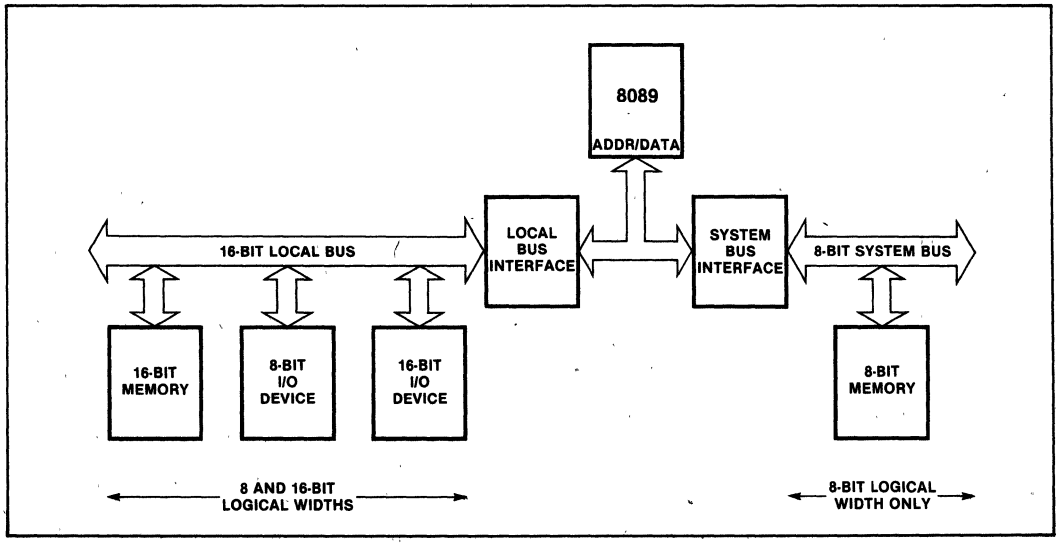


Figure 10. Logical Bus Widths for DMA Transfers

AFN02057A

Another feature derived from the two bus cycle DMA approach is character translation during DMA mode. Byte data may be translated via a 256-byte translation or lookup table. During each DMA transfer cycle, a byte of data is read from the source device, the data byte is translated, and then the translated byte is written to the destination device. Three bus cycles are required here since the translation requires a fetch cycle from memory.

Two bus cycle DMA also allows DMA transfers to be terminated based on masked comparison of the transferred data. This is discussed in the next section.

DMA Termination

The 8089 allows several conditions to terminate DMA transfers. One condition or several conditions may be specified. When several conditions are specified, DMA transfers are terminated when any one condition is detected. In addition, different task program re-entry points may be specified for each condition. This permits special post-DMA processing based on the terminate condition. Task program re-entry points are specified as offsets which are added to the task pointer. Three offsets are available: 0, 4, or 8. These offsets permit long or short jumps to termination routines. When more than one terminate condition occurs simultaneously, task program execution is resumed at the largest offset of the simultaneously occurring terminate conditions. An exception to this rule exists. The byte count terminate condition has highest priority and its offset is used if this terminate condition occurred.

DMA transfers can be terminated when the byte count (BC) register, which is decremented after each byte or word is transferred, reaches zero. The 16-bit BC register is initialized by task program instructions before the DMA transfer is started and permits data transfers of up to 64 kilobytes to be terminated. Each channel has an external terminate (EXT) input which can be activated by external circuitry to terminate the DMA transfer. Another condition allows termination based on masked comparison of transferred data. As byte data is transferred, an 8-bit mask value selects which bits of the data are compared with corresponding bits of an 8-bit compare value. Termination can be specified to occur either when a match occurs or does not occur. Examples using this terminate condition are transferring data until an EOF character is detected (match) and transferring data while bit 7 = 1 (mismatch). A final terminate condition called single transfer allows a single byte or word to be transferred.

Register Set

The register set of the 8089 is presented in Figure 11. Each channel has its own set of registers, except for the

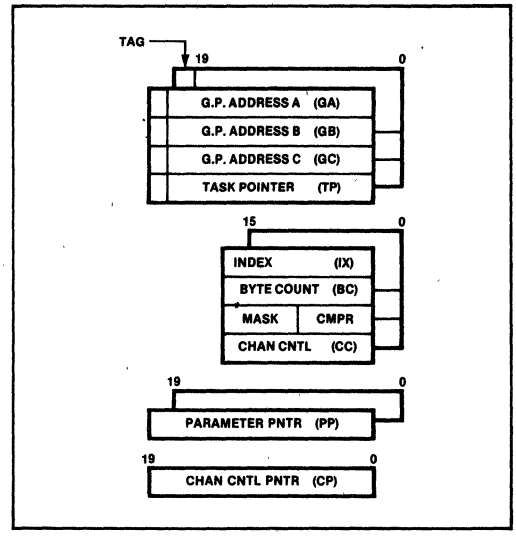


Figure 11. 8089 Register Set

channel control pointer register (CP) which is shared by both channels. This register is 20 bits in size and is used to access the channel control block whenever a channel receives a channel attention signal. Each channel has a 20-bit parameter pointer register (PP) which provides access to the parameter block. The common CP register is initialized during the 8089's initialization sequence while the PP registers are initialized whenever a channel attention signal is received. Therefore, the CP and PP may be read during task program execution, but cannot be changed.

Each channel has four 20-bit registers, each with an associated tag bit. The tag bit is used whenever the register is used as a pointer and indicates which address space (system or local) is accessed. If the tag bit is equal to 0, the 1 megabyte system address space is accessed using all 20 bits of the register. However, if the tag bit is equal to 1, the 64 kilobyte local address space is accessed using the lower 16 bits of the register. Instructions that initialize these registers either set or clear the tag bit. The load pointer instruction clears the tag bit, the move instruction sets the tag bit, and the move pointer instruction which moves data from memory into the register's 20 bits and tag bit either sets or clears the tag bit based on the contents of the referenced memory location.

The task pointer register (TP) is used as a task program counter. The remaining three 20-bit registers (GA, GB, and GC) are general-purpose registers. During task program execution, they may be used for data manipulation or as pointers. During DMA mode, the GA and GB registers point to source and destination devices and if

the translation option is specified, the GC register points to a 256-byte translation table. Two source/destination register specifications are possible: (1) GA points to the source and GB to the destination and (2) GB points to the source and GA to the destination.

Four 16-bit registers are also included in each channel's register set. The index register (IX) may be used by task program instructions to access memory and I/O devices. The address of the memory or I/O device is computed by adding the contents of IX with the contents of the specified pointer register. The byte count register (BC) can terminate DMA transfers. The mask/compare register (MC) may be used to perform masked compare operations during task program execution or masked compare DMA terminations. The channel control register (CC) specifies the details of DMA transfers (refer to *The 8086 Family User's Manual*). Although these four 16-bit registers have special functions at times, they may also be used as general-purpose registers for data manipulation. Use of the CC register for general-purpose functions is not recommended when both channels are simultaneously used since the chain bit specifies channel priority.

Instruction Set

In addition to intelligent, high speed DMA transfers which make the 8089 well-suited for I/O processing, the set of 53 instructions is tailored for I/O operations rather than data processing. Task programs are primarily used to prepare for and initiate DMA transfers and to perform post-DMA status checking. Included in the instruction set are data transfer, arithmetic, logical and bit manipulation, program transfer, and processor control instructions.

Data transfer instructions move information between registers and memory or I/O devices. Movement of data between any two devices in either address space is easily accomplished with the MOV instruction. This includes memory to memory and I/O to I/O transfers. Arithmetic instructions such as add, increment, and decrement are provided for simple computations (e.g., pointer manipulation) required in I/O processing. The logical and bit manipulation instructions are especially useful in the I/O environment to mask data and set or clear bits.

Procedure calls and conditional and unconditional jumps are provided with the program transfer instructions. Jump if masked compare equal or not equal and jump if bit true or false instructions are also included in this group. Finally, the processor control instructions perform test and set while locked operations (semaphore access), define logical DMA bus widths, initiate DMA transfers, activate the SINTR interrupt output lines, and halt task program execution.

Special Design Considerations

Most interrupt signals received by the 8089 are used to synchronize DMA transfers and the 8089's DMA request (DRQ) inputs support these interrupts. The 8089 also supports non-DMA related interrupt signals.

Most non-DMA interrupts are used to synchronize channel program execution with some external event. Here channel program execution is suspended and the channel waits until the synchronizing signal is received before resuming task program execution. A disk control example is waiting for the INDEX signal before formatting the track.

A dummy DMA transfer can be used to implement this function. This is a synchronized, externally terminated DMA transfer where no data is actually transferred. The DMA request (DRQ) signal is held inactive and the channel executes idle cycles while waiting for either DRQ or EXT (external terminate) signals.

No bus cycles are executed by the channel during idle cycles. The channel's EXT input is used to receive the synchronizing signal. When received, the dummy DMA transfer is terminated and channel program execution resumes. The dummy DMA transfer can also be viewed as the iAPX 86/10's WAIT instruction.

This concept can also be applied when two channels are operating. For example, one channel may be waiting for a synchronizing signal while the other channel is operating. Here the second channel can execute at full speed since the first channel is executing idle cycles.

One application of this two channel approach is to perform two independent DMA transfers in rapid succession. After the first DMA transfer, conditions are tested to determine if the second DMA transfer is performed. One channel (e.g., channel 1) initializes its registers for the second DMA transfer and executes a dummy DMA transfer. Next, the other channel (e.g., channel 2) initializes its registers for the first DMA transfer. Channel 2 performs the first DMA transfer, activates channel 1's EXT input, and halts. Channel 1 resumes task program execution and determines whether conditions permit the second DMA transfer. If the proper conditions are present, the DMA transfer is performed. The two DMA transfers are performed in rapid succession because both channels initialized their registers before either DMA transfer was performed. A single channel implementation must re-initialize its registers after the first DMA transfer before performing the second DMA transfer. Therefore, the time between successive DMA transfers is increased.

In the example above, channel 1 performs two DMA transfers—a dummy DMA transfer and then the second DMA transfer. Registers are initialized for the second DMA transfer before the dummy DMA transfer is per-

formed. Therefore, all DMA register changes resulting from the dummy DMA transfer must be accounted for when initializing the registers. Synchronized DMA transfers between I/O and memory update the byte count register (BC) and the memory pointer register (GA or GB). During each two cycle transfer, BC is decremented during the data fetch bus cycle and GA or GB is incremented during the data store bus cycle. Since the dummy DMA transfer never stores the data (DRQ remains inactive), the memory pointer is never incremented. However, BC may or may not be decremented depending on whether source or destination synchronization is selected. If source synchronization is selected, BC is not decremented because the data is not fetched. However, since the data is prefetched during destination synchronized DMA transfers, BC is decremented. This means that BC must be adjusted only when a destination synchronized DMA transfer follows the dummy DMA transfer. Here BC must be loaded with the actual number of data bytes to be transferred plus one for byte transfers or plus two for word transfers. A byte transfer is defined as the fetching and storing of a single byte. All other cases are considered word transfers since the net result is that 16 bits of data are transferred during the two or more bus cycles.

III. SHUGART SA4008 DRIVE

The Shugart Associates SA4008 disk drive is typical of Winchester drives now being used in microcomputer systems. The unformatted drive capacity is 29 megabytes. Typical of high performance drives, the transfer rate is 889 kilobytes/second and the average seek time is 65 milliseconds. A summary of the drive's performance and functional specifications is included in Appendix A.

Drive Organization

The Shugart SA4008 drive has two 14-inch disk platters. The top and bottom surfaces of these two platters provide four recording surfaces. Each recording surface contains 404 concentric circular data paths called tracks. The tracks on each surface are accessed by two read/write heads which move along the radial distance of the circular platter (Fig. 12). The two heads are rigidly connected and move in unison. One read/write head travels from the outermost track of the surface to the midway point between the outermost and innermost tracks. The other head travels from the midway point to the innermost track. Each of the four surfaces has two read/write heads (eight total heads). The drive's head positioning mechanism moves all eight heads in unison onto 202 discrete positions called cylinders (numbered 0 through 201). The head mechanism is positioned at

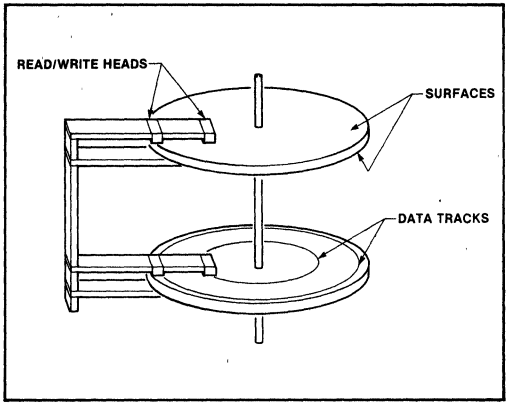


Figure 12. SA4008 Drive with Two Heads Per Surface

cylinder 0 when the outermost track is accessed and at cylinder 201 when the innermost track is accessed. At each cylinder position, eight unique data tracks are accessible, one by each head. By activating the electronics of one read/write head, a single data track is accessed. With 8 heads and 202 cylinders, the SA4008 has a total of 1,616 tracks.

Sector Format

Data is recorded on sections of the track called sectors. The number of sectors per track is a function of the controller design. The SA4008 allows any number of sectors per track. This design organizes each track into 30 sectors (Fig. 13). The 600 bytes of each sector is divided into an ID field, data field and gaps. The ID field is a unique identifier or address used to locate a particular data record. The data field contains the 512 byte data record that is read or written by the host processor. Gaps containing no usable information are inserted before and after the ID and data fields to allow the drive and controller electronics time for synchronization and switching between read and write modes.

Assignment of sequential records to sectors is interleaved using an interleave code of 3 such that logical sectors are three physical sectors apart (Fig. 14). Since a data record is buffered in local memory, this interleave scheme allows two sector times to transfer the data record to or from system memory. This allows the disk controller to operate at lower system bus priority and provides enough time to transfer the data record between the local buffer and system buffer. When the 8089 has complete use of the system bus, a 512 byte data record can be transferred in 564 μsec which is 84% of

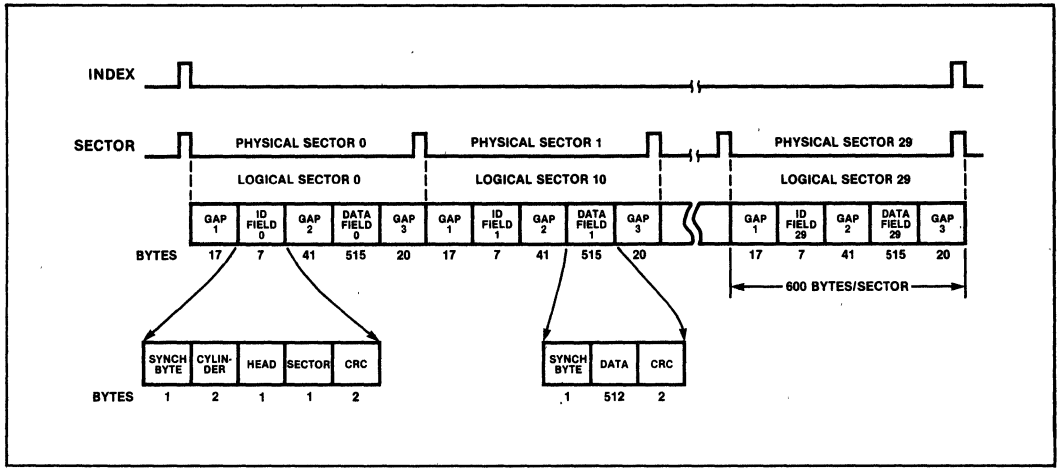


Figure 13. Track Format

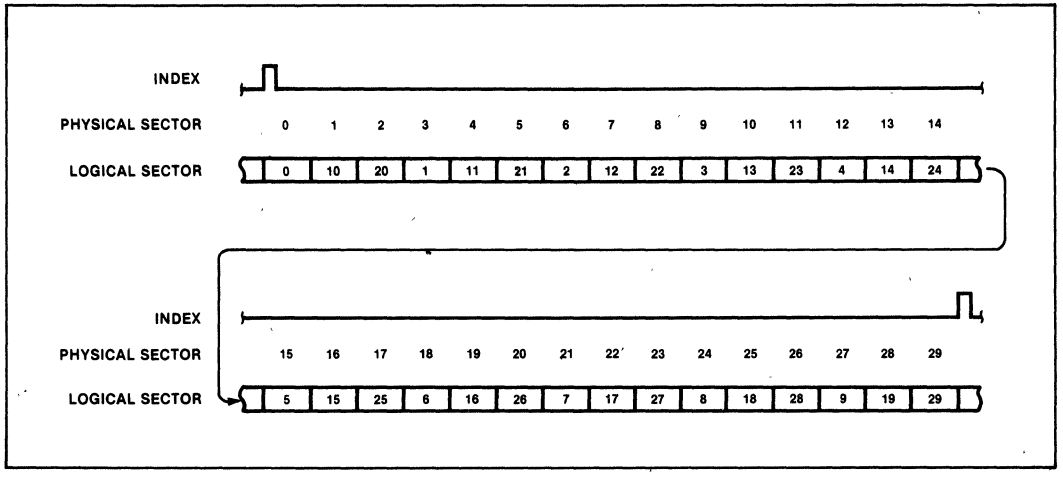


Figure 14. Interleaved Sector Ordering

the 672 μsec sector time. The selected interleave scheme permits up to 10 sequential logical sectors to be accessed per 20.2 millisecond disk revolution.

To access up to 15 sequential logical sectors per revolution, an interleave code of 2 could be used. For this case, logical sectors are two physical sectors apart and the buffered data record must be transferred to or from system memory in one sector time (672 μsec). This requires that the 8089 retain possession of the system bus for the entire data record transfer after acquiring the system bus. The 8089 can accomplish this with a $\overline{\text{LOCK}}$ output signal which is discussed later. The 564 μsec data record transfer time allows 108 μsec to set up the DMA transfer to or from the system bus, obtain possession of the system bus, and prepare for a subsequent disk sector access.

Disk Drive Interface Signals

The interface signals (Fig. 15) between the SA4008 drive and the controller are now described. The input control signals are first described, followed by the output control signals, and finally the data transfer signals.

The input control signals to the drive are DRIVE SELECT, DIRECTION SELECT, STEP, HEAD SELECT, FAULT CLEAR, WRITE GATE, and READ GATE. Four drive select signals, DRIVE SELECT 1 to 4, allow selection of one drive in a multiple drive configuration of up to four drives. A jumper is used to select one of the DRIVE SELECT signals and allows the drive to respond to only one DRIVE SELECT signal. The DRIVE SELECT 4/SEEK COMPLETE line can be jumper selected as the DRIVE SELECT 4 signal or SEEK COMPLETE signal (see description below). The DIRECTION SELECT and STEP signals are used to position the read/write heads. DIRECTION SELECT defines an inward or outward movement while the STEP line is pulsed. Each pulse moves the heads one cylinder position. Four head select signals, HEAD SELECT 1, 2, 4 and 8, are used to select one of the SA4008's eight read/write heads. Four signals are provided to allow eight optional fixed heads to be selected. The FAULT CLEAR signal is used to reset a write fault condition. The WRITE GATE signal enables data to be written on the selected data track, while the READ GATE enables reading from the track.

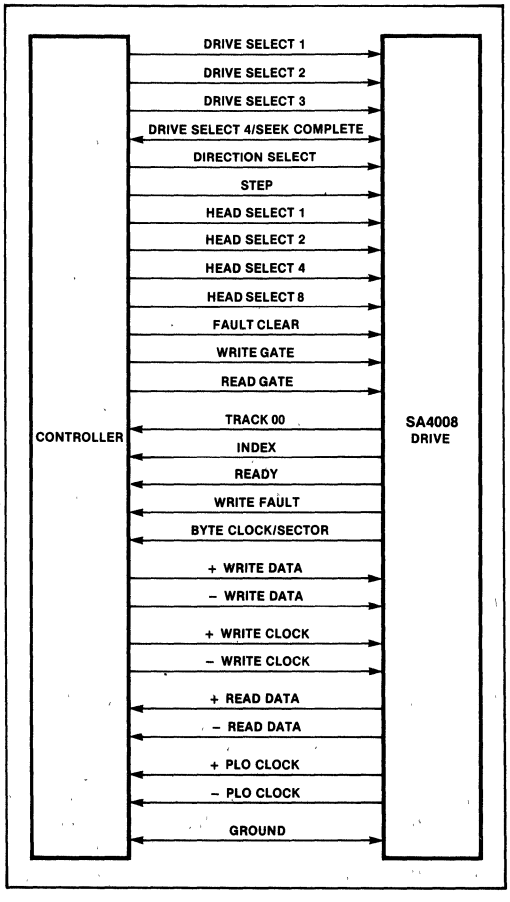


Figure 15. SA4008 Interface Signals

The output control signals from the drive are TRACK 00, INDEX, READY, WRITE FAULT, SEEK COMPLETE, and BYTE CLOCK/SECTOR. The TRACK 00 signal is activated when the read/write heads are positioned at track 0 (cylinder 0). The INDEX signal is pulsed once each revolution (20.2 msec) indicating the beginning of the data track. The READY signal indicates that the drive is ready to position the read/write heads, read data, or write data. The WRITE FAULT signal indicates that a condition which caused improper writing on the disk occurred. The SEEK COMPLETE signal is available in a single drive configuration and indicates when the read/write heads have arrived at the desired cylinder during a seek operation. The DRIVE SELECT 4/SEEK COMPLETE line can be jumper selected as the DRIVE SELECT 4 signal (multiple drive configuration) or SEEK COMPLETE (signal drive configuration). The SEEK COMPLETE signal is selected with the controller described in this application note. The BYTE CLOCK/SECTOR line is another jumper selectable signal. It can be configured as the BYTE CLOCK signal (1.12 μsec period) or as the SECTOR signal. The number of SECTOR pulses per revolution is jumper programmable. The controller described here requires selection of the SECTOR signal and 30 sector pulses per revolution.

The SA4008 provides four data transfer signals: WRITE DATA, WRITE CLOCK, READ DATA and PLO (Phase Locked Oscillator) CLOCK. All of these are differential signals. The WRITE DATA and WRITE CLOCK signals are received by the drive and used to write data on the track. The WRITE DATA signal provides the data while the WRITE CLOCK signal is used to sample the data. The READ DATA and PLO CLOCK signals are transmitted by the drive and used to read data from the track. The READ DATA signal provides the data while the PLO CLOCK signal is used to sample the data. Both the WRITE DATA and READ DATA signals are in the non-return to zero (NRZ) format.

A detailed description and timing of the interface signals can be obtained from the Shugart Associates manuals referenced in the introduction.

Functional Operations

The SA4008 provides three functional operations: track accessing, write data, and read data. These operations are initiated and controlled by certain interface signals.

Track accessing (seeking from one track to another) is accomplished by activating the DRIVE SELECT line and deactivating the WRITE GATE line. Inward or outward movement is selected by activating or deactivating, respectively, the DIRECTION SELECT line. The STEP line is pulsed once for each track that the read/write heads are moved.

Writing data to the SA4008 is initiated by activating the DRIVE SELECT line, selecting the desired read/write head by activating the HEAD SELECT lines, and providing a clock signal on the WRITE CLOCK line. The WRITE GATE line is then activated and the data to be written is transmitted on the WRITE DATA line. The WRITE GATE line is deactivated to terminate writing.

Reading data from the SA4008 is initiated by activating the DRIVE SELECT line and selecting the desired read/write head by activating the HEAD SELECT lines. The READ GATE line is then activated and the data is read on the READ DATA line using the PLO CLOCK signal to sample the data. The READ GATE line is deactivated to terminate reading.

IV. DISK CONTROLLER OPERATIONS

By using an 8089, the disk controller becomes an intelligent interface between the host processor and the disk drive. The host issues a single high level command for the desired operation and the 8089 implements the operation through task program control.

The 8089-based disk controller described in this application note implements four basic disk control operations: seek track, format track, write data record, and read data record. The previous section described the three functional operations of the SA4008 drive: track accessing, write data, and read data. The controller uses these three drive operations to implement the four high level operations. An overview of the four operations is now presented. This serves as an introduction to the disk controller before hardware and software details are described.

Seek Track

The seek track operation is implemented primarily through task program control with minimal use of special hardware. Based on the cylinder which is presently accessed by the read/write head mechanism, the task program determines which direction (inward or outward) the head mechanism must be moved. The number of cylinder positions that the heads must be moved is also determined. The task program writes data to an octal latch which transmits the DIRECTION SELECT and STEP signals to the SA4008 drive. By writing the proper data sequence to the octal latch, DIRECTION SELECT is asserted and STEP is pulsed the required number of times. Finally the task program asserts the drive's head select (HEAD SELECT 1, 2, 4 and 8) signals to access the desired track.

Format Track

The format track, write data record, and read data record operations are implemented by a task program which controls special hardware. Details of the special hardware are described in the next section.

The timing overview of the format track operation is presented in Figure 16. The INDEX, SECTOR and READ DATA signals from the drive and the WRITE GATE, WRITE DATA, and READ GATE signals to the drive are shown. 8089 channel activity is also shown. The READ GATE and READ DATA signals remain inactive during the format track operation.

Channel 1 begins the format track operation by initializing the registers for the DMA transfer which writes sector 0's ID data on the track. Serial/parallel conversion hardware is used to convert the 8089's parallel data to serial so that it can be received by the drive. The hardware is initialized with zeros so that when the WRITE GATE is activated, zeros are written on the track. Next a dummy DMA transfer is used to wait for the INDEX pulse which indicates the beginning of the track.

When the INDEX pulse is received, channel 1 resumes execution. The INDEX pulse also activates the WRITE GATE signal to the drive and zeros are written on the track. Timing hardware which was started by the SECTOR pulse determines when to stop writing zeros and begin the write ID field DMA transfer. A synch character is written on the track before the ID field and CRC word after the ID field. After the ID data for sector 0 has been written on the track, the hardware resumes writing zeros.

Channel 1 next initializes the DMA registers for writing ID data to the next sector. A dummy DMA transfer is started to wait for the SECTOR pulse. Channel 1 now idles while it waits for the SECTOR pulse. Note that zeros continue to be written on the track between ID data.

ID data for the remaining 29 sectors is written on the track identically to the first sector. After ID data is written for the last sector, channel 1 deactivates the WRITE GATE signal. WRITE GATE deactivation is delayed so that zeros are written into the data field. This ensures that after a data record has been written (in the last sector), the required zeros are present before and after the data field.

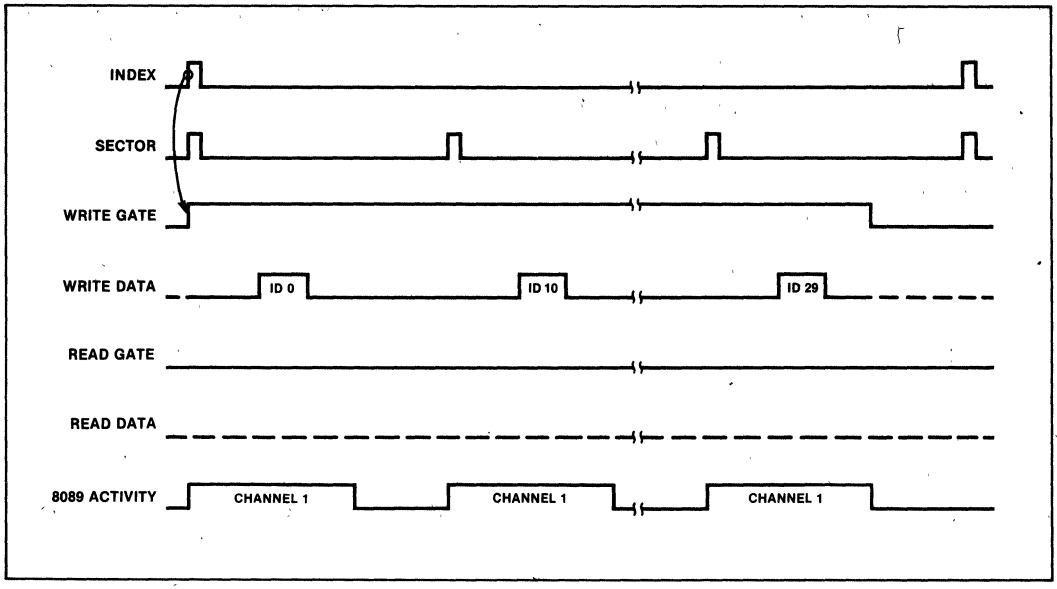


Figure 16. Format Track Timing Overview

Write Data Record

The data transfer operations (write data record and read data record) are implemented with both 8089 channels (Fig. 17). Channel 2 searches for the desired sector by comparing the ID field information read from the track with the desired ID field information. The comparison is performed by a hardware comparator. One input of the comparator accepts ID information read from the track while the other input accepts the desired ID information transferred from channel 2 using DMA transfers. Upon locating the desired sector, channel 1 transfers the data record to or from the track using DMA transfers. Both channels perform DMA transfers using the technique described earlier which allows two DMA transfers in rapid succession.

Higher data capacity is achieved with this two channel approach than with a single channel approach. With two channels, all DMA registers are initialized before either DMA transfer is started. No register re-initialization is required betwen the DMA transfers for the two channel approach. To allow for register re-initialization between DMA transfers in the single channel approach, a larger gap between the ID and data fields is required. This results in lower data capacity per track and therefore lower data capacity per drive.

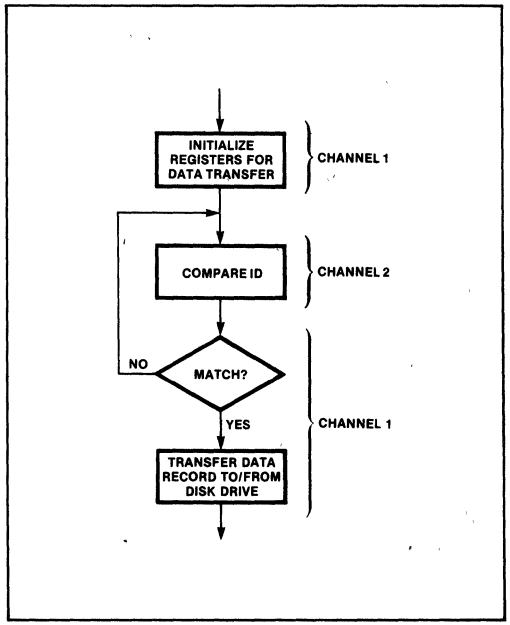


Figure 17. Sector Search and Data Transfer

The write data record operation begins with channel 1 initializing DMA registers used to transfer the data record to the track (Fig. 17) and starting a dummy DMA transfer. Next channel 2 initializes its DMA registers used to transfer the desired ID information to the hardware comparator. Channel 2 waits for a SECTOR pulse with a dummy DMA transfer. When the SECTOR pulse is detected, channel 2 performs the "compare" DMA transfer, activates channel 1's EXT input and halts.

Activation of EXT terminates channel 1's dummy DMA transfer and resumes task program execution. The hardware comparator is tested to determine if the desired sector is found (i.e., the compare is successful). If not found, the ID field comparison is repeated for the subsequent sector. If the desired sector is found, the data record is written in the data field which follows the ID field.

The timing overview of Figure 18 shows the sector activity when the desired sector is found. Channel 2's dummy DMA transfer is terminated by the SECTOR pulse and the READ GATE signal is activated. This allows the serial/parallel conversion hardware to read the serial data from the track and convert it to a parallel format. The beginning of the ID field is found by hardware that searches for a synch character. When detected, channel 2's DMA transfer moves the desired ID information to the hardware comparator synchronously with the ID information from the track arriving at the comparator. Finally, channel 2 activates channel 1's EXT input and halts. Channel 2's sector search activity is the same for all sectors.

Channel 1 resumes execution, tests the hardware comparator, and deactivates the READ GATE signal. Figure 18 shows that channel 1 then activates the WRITE GATE signal and zeros are written on the track. Timing hardware which was started by the detection of the ID field's synch character determines when to stop writing zeros and begin the write data record DMA transfer. A synch character precedes the data record and an CRC word follows the data record. Finally channel 1 deactivates the WRITE GATE signal and halts.

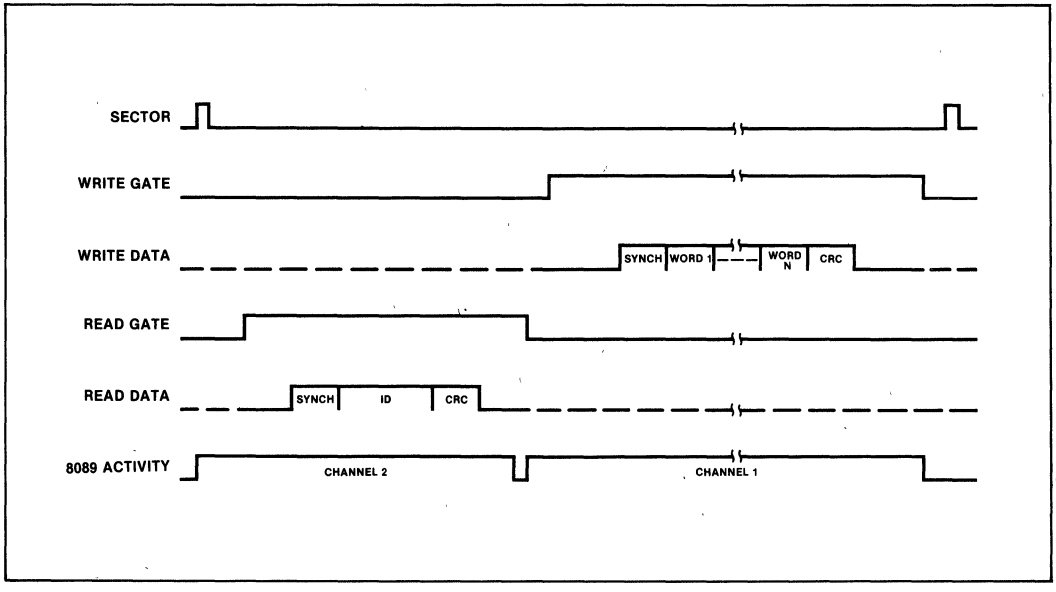


Figure 18. Write Data Record Timing Overview

Read Data Record

The read data record operation is similar to the write data record operation. The sector search activity is identical. Only channel 1's activity after locating the desired sector is different.

The timing overview of Figure 19 shows that when channel 1 resumes execution the hardware comparator is tested and the READ GATE signal is deactivated. Next the READ GATE is again activated and the hardware searches for the synch character. The READ GATE signal is momentarily deactivated so that the disk drive does not read where the WRITE GATE has been activated (during a previous write data record operation). This ensures that the drive's data separator decodes data properly. When the synch character is detected, channel 1's DMA transfer reads the data record from the track. Finally, channel 1 checks for a CRC error, deactivates the READ GATE signal, and halts.

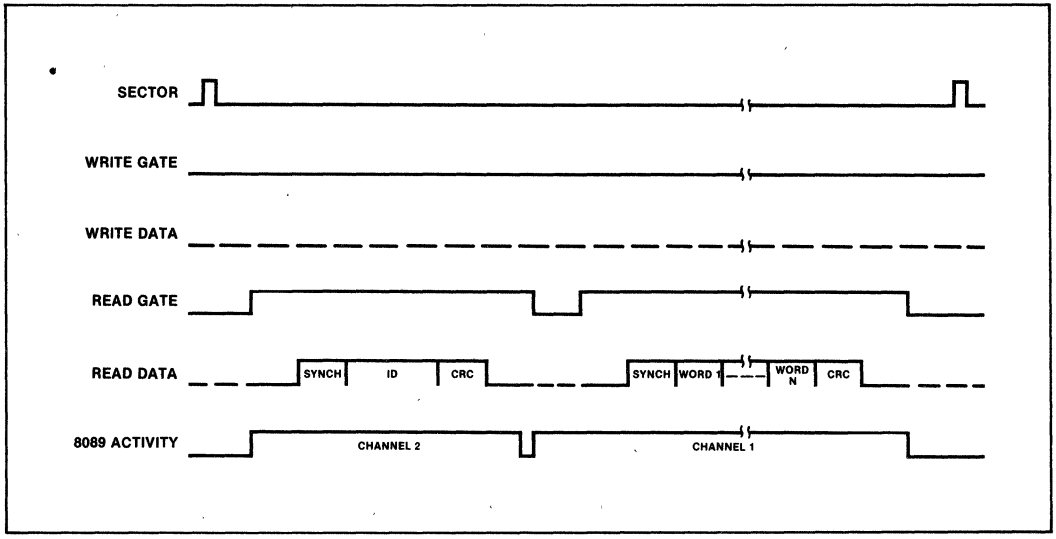


Figure 19. Read Data Record Timing Overview

V. HARDWARE DESIGN

The controller was designed to be compatible with Multibus, an industry-standard multiprocessor system bus. It was constructed on an iSBC 905 Universal Prototype board using wirewrap interconnections. Seventy-five IC packages reside on this 6-3/4 by 12 inch board. The development environment consisted of the controller board, an iSBC 86/12A single board computer (based on the iAPX 86/10) which served as the host processor, and an iSBC 604 cardcage which provided a Multibus interconnect between the two boards. Other development tools used were an ICE-86 in-circuit emulator and the RBF-89 real-time breakpoint facility.

A block diagram of the disk controller is shown in Figure 20. The hardware is divided into four major sections—I/O processor, Multibus interface, timing and control, and data transfer. The 8089 I/O processor along with the timing and control circuitry supervise all disk control operations. The 8089's interface to the timing and control circuitry is through control and status registers which are part of the timing and control section.

I/O Processor

The I/O processor section (Figure 21) consists of the 8089, support circuitry, local bus interface, and local memory. Support circuitry includes the 8284A clock generator and the 8288 bus controller. The clock generator is configured in asynchronous mode since ready signals are generated asynchronously with respect to the 8089's clock signal. The 8089's local bus read signal, $\overline{IORD}$, is generated from the bus controller's $\overline{IORC}$ and $\overline{INTA}$ commands since $\overline{INTA}$ is activated whenever the 8089 fetches instructions from its local bus. Both bus controller I/O write commands, advanced ($\overline{AIOWC}$) and normal ($\overline{IOWC}$), are used. The advanced command is used to write to all local devices except the two 8282 control ports. The normal command is used when writing to these control ports to prevent glitching of the 8282's output signals. This command prevents glitches since its timing guarantees that the write data is valid before the command's leading edge.

The channel attention (CA) signal is generated by decoding Multibus I/O writes to ports 0 and 1 allowing the host processor to start channel 1 and 2, respectively. A CA signal for channel 2 is also generated when the 8089 accesses local bus port 4070H allowing channel 1 to start channel 2.

The local bus interface is implemented with two 8282 octal latches and two 8286 octal transceivers. Two 8205 one-of-eight decoders provide the local bus address decoding for memory and I/O devices. Two 2716-1 EPROM components provide 4K bytes of program storage addressable from 2000H to 2FFFH. If more program storage is required, the 2716-1s can be replaced with 2732As or 2764s to provide 8K or 16K bytes, respectively, of program memory. 4K bytes of read/write memory for storing program variables and buffering disk sector data are provided with four 2142-3 static RAM components addressable from 0 to 7FFH.

Multibus™ Interface

The Multibus interface (Fig. 21) is implemented with three 8283 octal latches, three 8287 octal transceivers, 8289 bus arbiter, and byte swap circuitry. The 8089 has access to the full 1 megabyte Multibus memory address space since all 20 address signals are latched with the three address latches. Memory read and write commands ($\overline{MRDC}$ and $\overline{AMWC}$) from the 8288 bus controller are used to access shared system memory. The 8289 bus arbiter provides the system bus access functions for the 8089. The iSBC 604 cardcage is configured for serial priority resolution with the iSBC 86/12A having priority over the disk controller board. The priorities can be changed by simply swapping the cardcage slot locations of the two boards.

The 8089's $\overline{LOCK}$ output is connected to the bus arbiter's $\overline{LOCK}$ input. While $\overline{LOCK}$ is active, the bus arbiter will not relinquish the shared system bus to another processor regardless of its priority. A channel activates the $\overline{LOCK}$ output when a test and set while locked instruction, TSL, is executed (semaphore access). A channel may also activate $\overline{LOCK}$ for the entire duration of a DMA transfer by setting the LOCK bit in its channel control register (CC). This ensures that once the system bus is acquired, the DMA transfer is completed as quickly as possible.

Three data transceivers and associated byte swap circuitry provide 8- and 16-bit Multibus compatibility. Since Multibus convention states that all 8-bit transfers must occur on the lower half of the 16-bit data bus ($\overline{DAT0}$ to $\overline{DAT7}$), all 8089 designs which access Multibus must provide byte swap circuitry. Even though the system bus is defined as 16 bits wide during the initialization of the 8089, the 8089 may perform byte references to odd-addressed memory locations. This results in the high byte of a 16-bit word being transferred over the lower half of the data bus.

Timing and Control

The timing and control section (Fig. 22) receives signals from the 8089 and disk drive to control all disk operations. The interface with the 8089 is via two 8-bit control ports and one 8-bit status port. Control ports 1 and 2 are implemented with 8282 latches and have addresses 4010H and 4021H, respectively. The two control ports are the primary interface from the software to the hard-

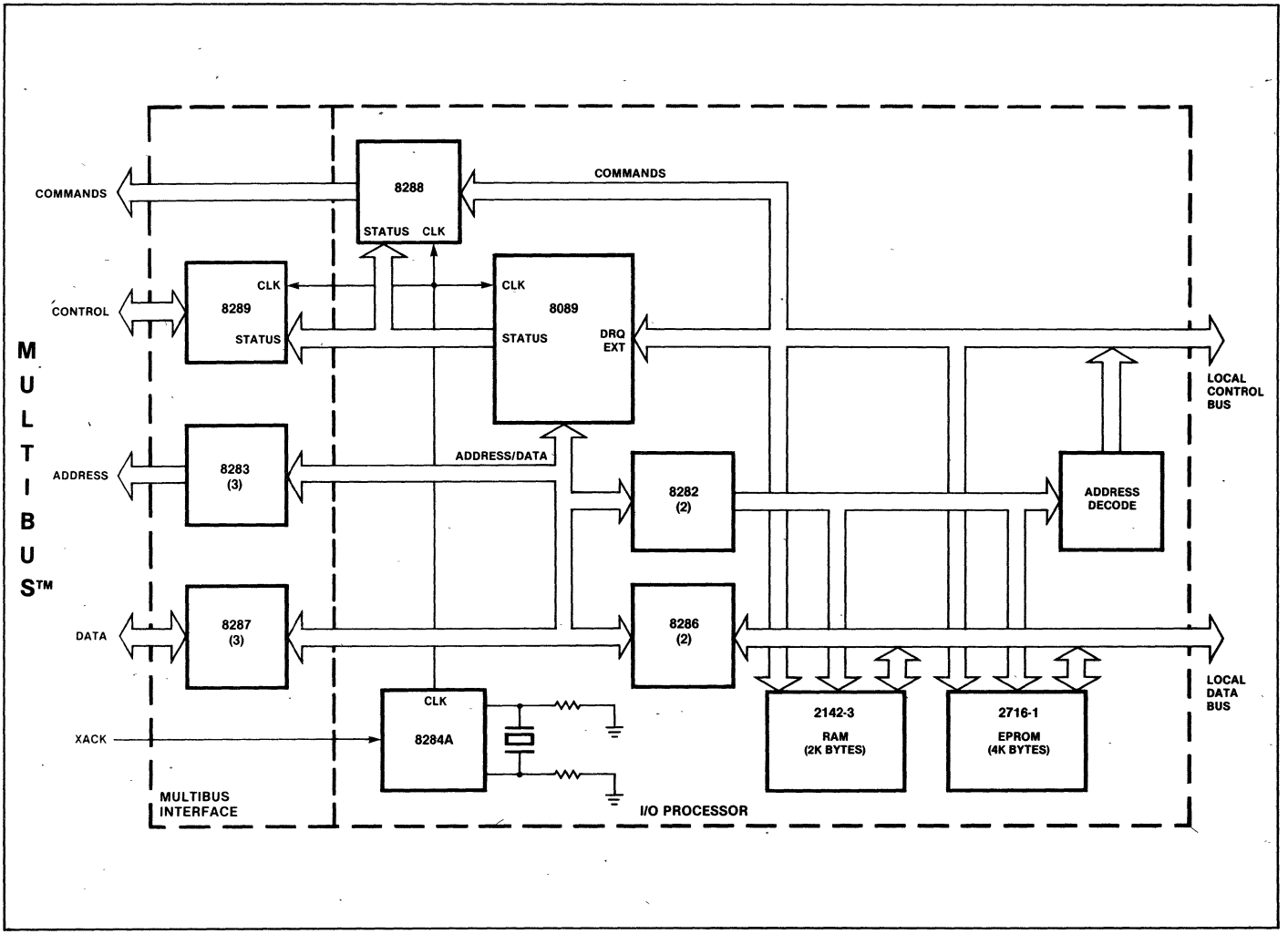


Figure 20. Disk Controller Block Diagram (Sheet 1 of 2)

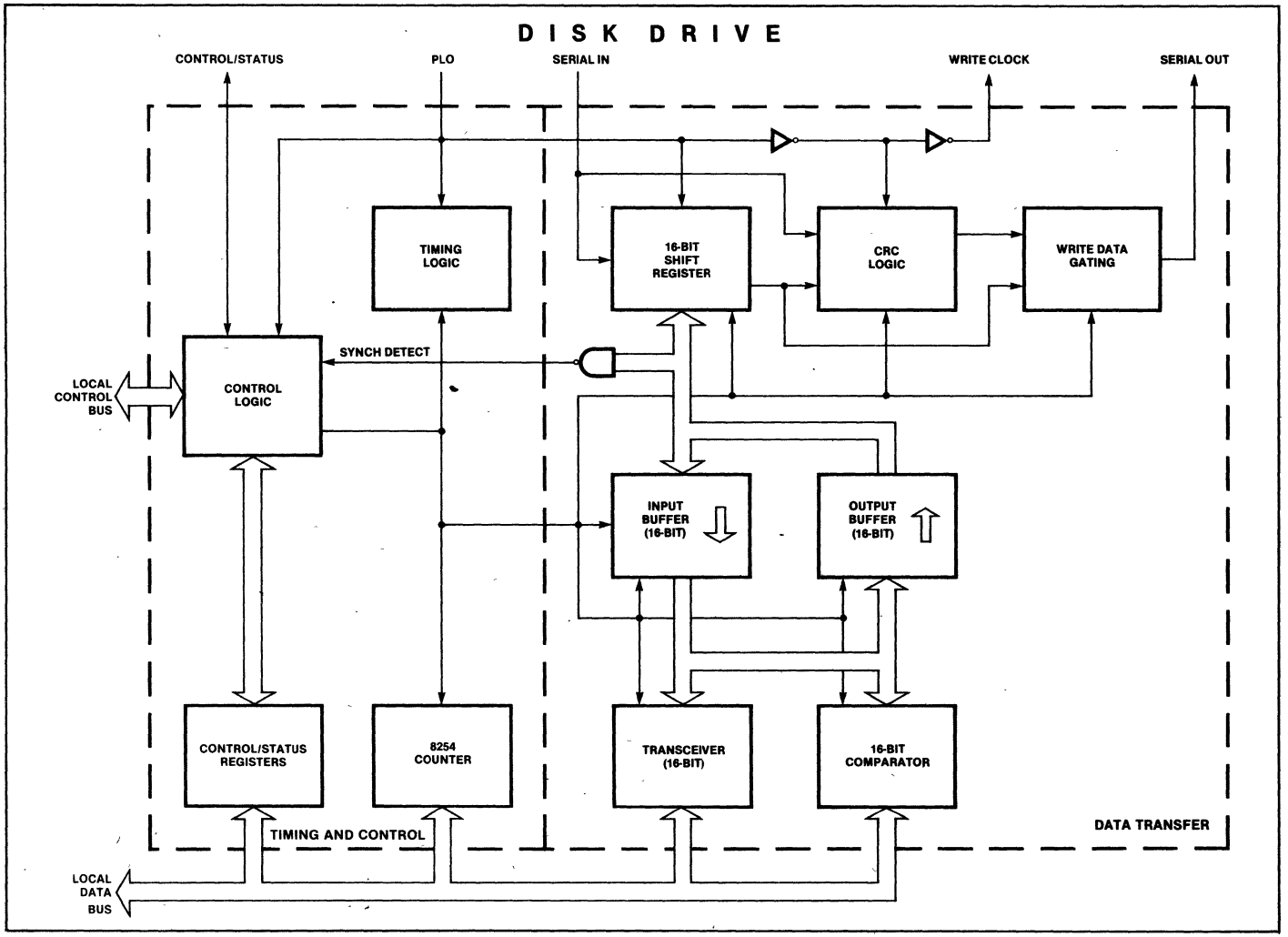


Figure 20. Disk Controller Block Diagram (Sheet 2 of 2)

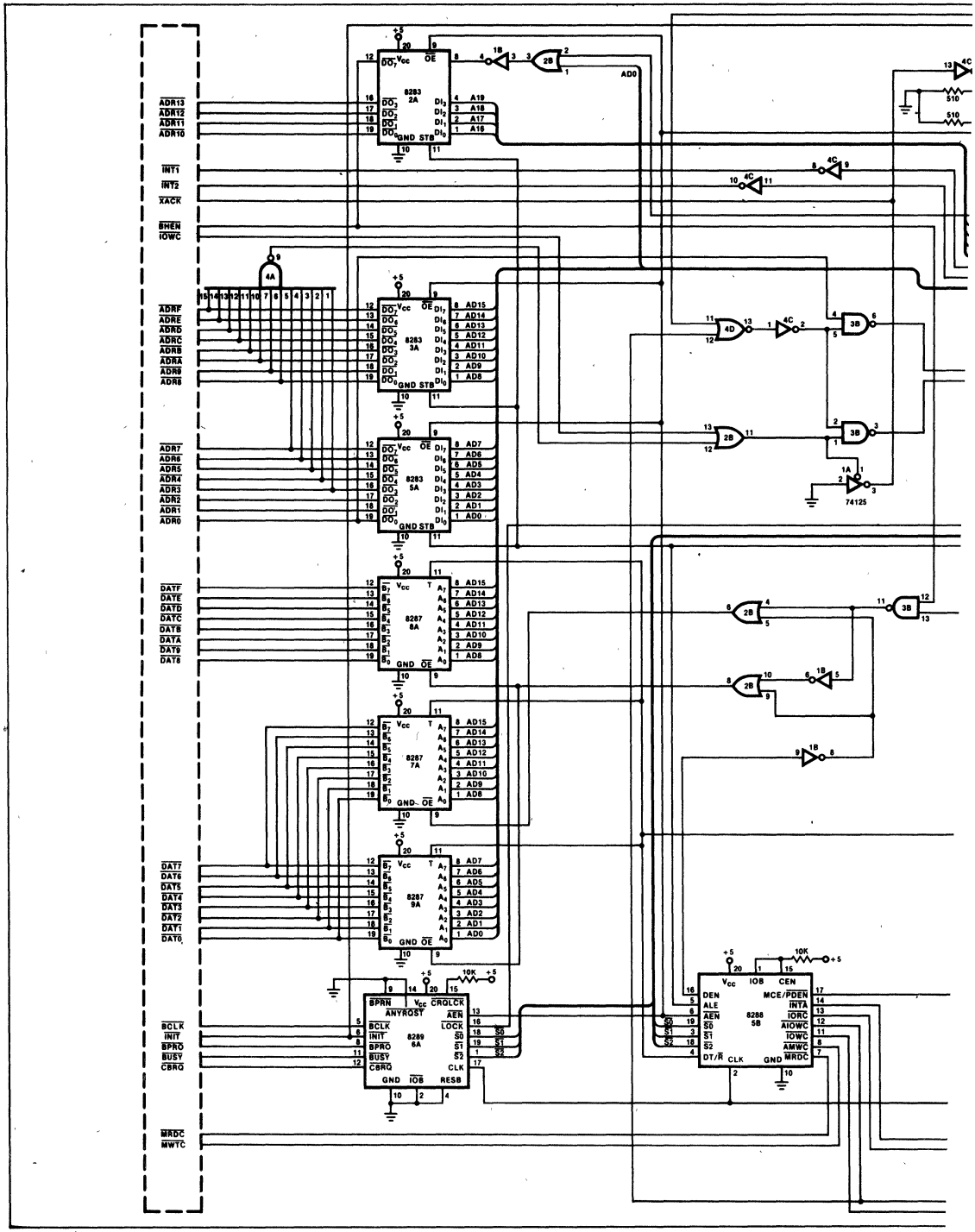


Figure 21. I/O Processor and Multibus Interface

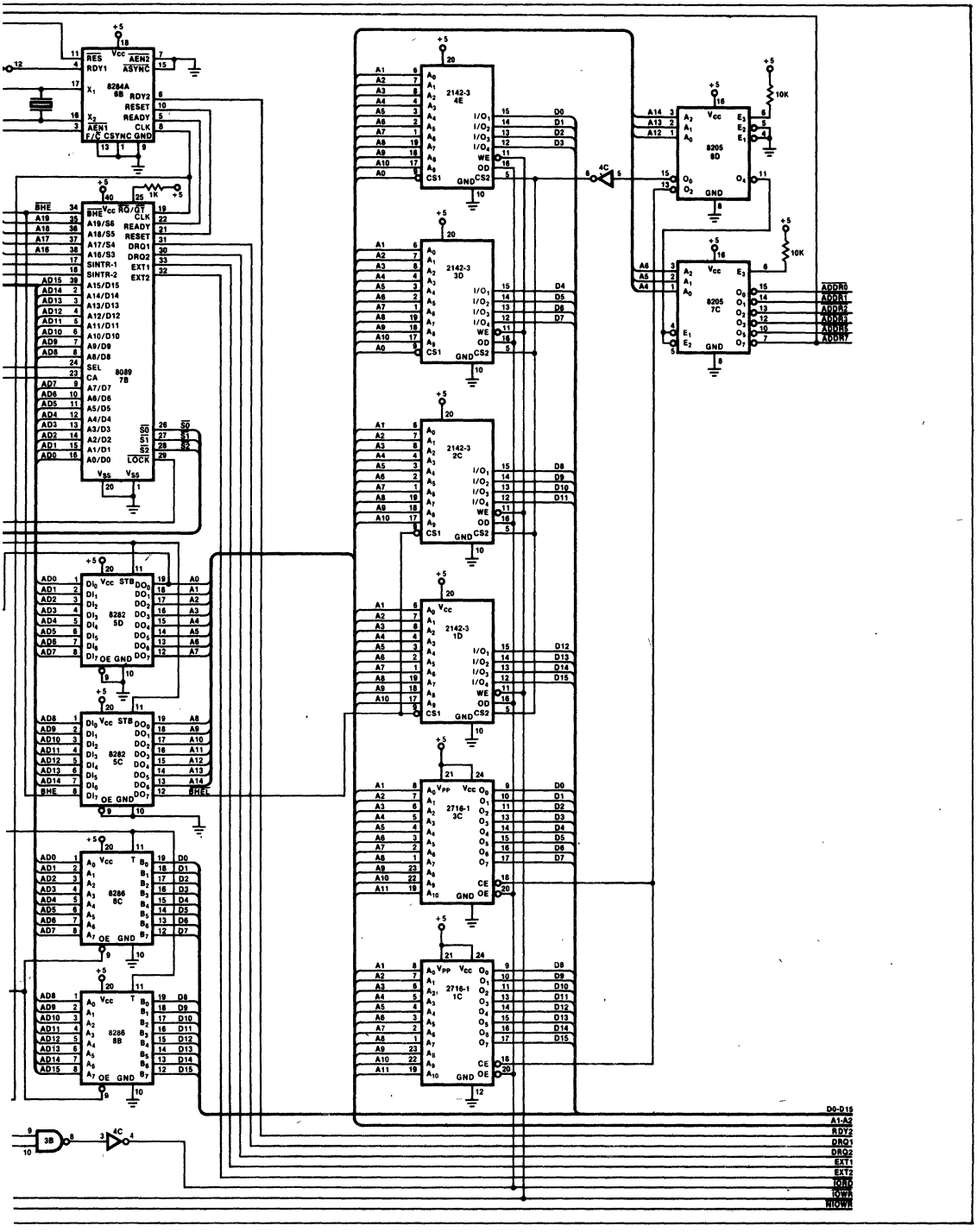


Figure 21. I/O Processor and Multibus Interface

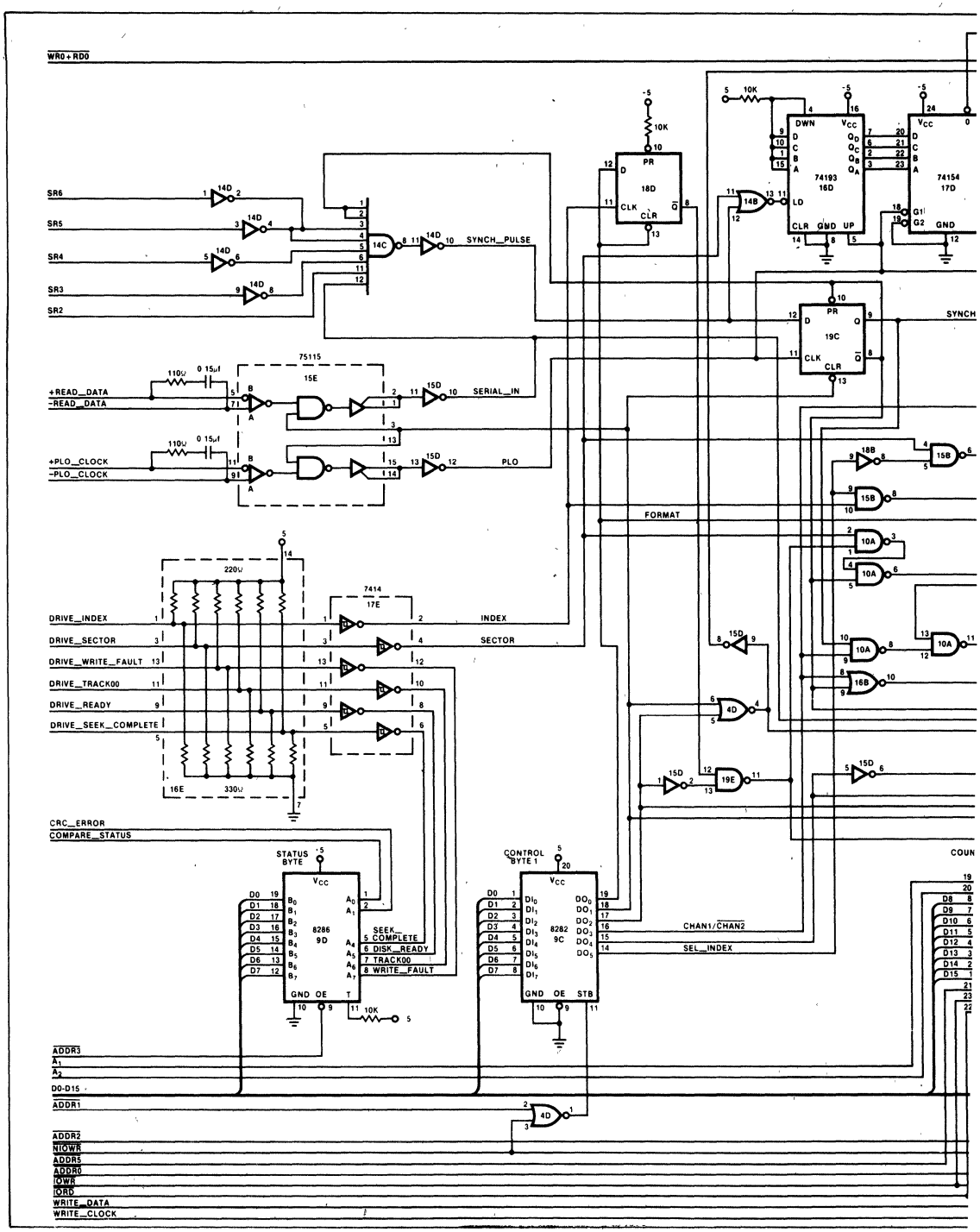


Figure 22. Timing and Control

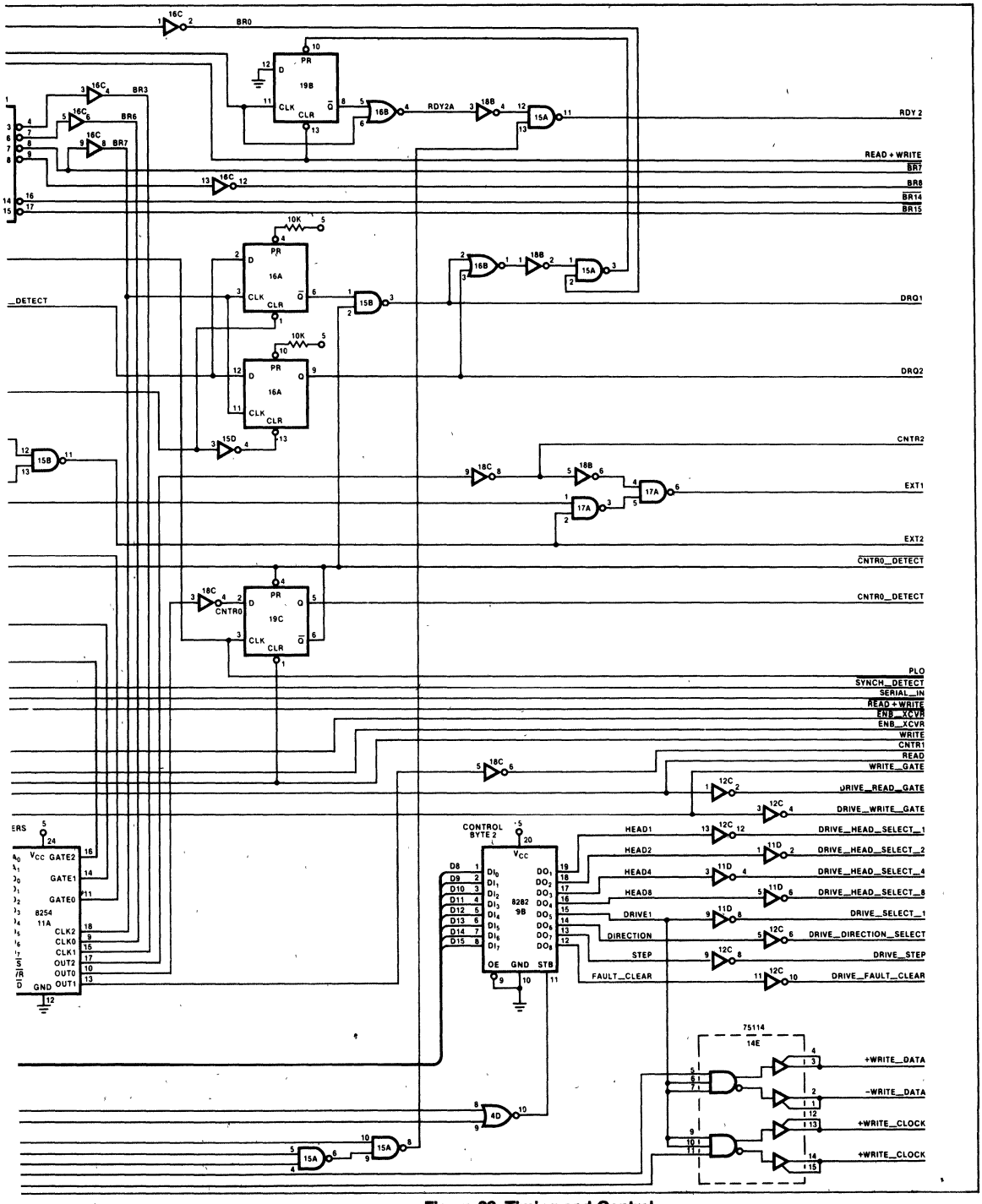


Figure 22. Timing and Control

ware and allow the 8089's task program to control all disk drive activity. The status port is implemented with an 8286 transceiver, has 4030H as an address and allows the task program to monitor drive activity. The 8288 bus controller's normal I/O write command is used to write to the control ports. This prevents the outputs from glitching, which can occur if the advanced I/O write command is used.

Output signals from control port 1 are used to control the special hardware. The FORMAT signal is active when formatting a track. The READ signal is active when reading an ID or data field. The WRITE signal is active when writing an ID or data field. The CHAN1/$\overline{\text{CHAN2}}$ signal enables generation of the proper DMA request (DRQ) signal. The ENB_XCVR signal enables transceivers when reading or writing sector data or disables them when comparing an ID field. The SEL_INDEX signal selects the drive's INDEX or SECTOR pulse for terminating dummy DMA transfers.

Output signals from control port 2 are transmitted to the disk drive. The head select (HEAD1, HEAD2, HEAD4, and HEAD8), drive select (DRIVE1), seek track (DIRECTION and STEP), and FAULT_CLEAR signals are generated by control port 2.

The status port receives signals from the special hardware and the disk drive. From the special hardware are COMPARE_STATUS and CRC_ERROR which indicate the status' of the ID field compare and data read, respectively. The SEEK_COMPLETE, DISK_READY, TRACK00, and WRITE_FAULT signals are from the SA4008.

The interface with the disk drive involves both digital and analog signals. All control signals are digital while the READ_DATA, WRITE_DATA, PLO_CLOCK, and WRITE_CLOCK are differential signals. Control signals from the drive are resistor terminated and conditioned with 7414 schmitt-trigger inverters. The control signals to the drive are driven with 7406 open-collector inverting drivers. The READ_DATA and PLO_CLOCK inputs are received with a 75115 dual differential receiver while the WRITE_DATA and WRITE_CLOCK outputs are driven with a 75114 dual differential driver.

A 16-bit ring counter is used to provide bit resolution timing. Only one of the sixteen outputs is active at any time. As 16-bit words are being serially received from or transmitted to the drive, the active ring counter output corresponds to a bit received or transmitted. When data is received from the drive, output 0 (BR0) corresponds to bit 0 of the received word, BR7 to bit 7, and BR15 to bit 15. When data is transmitted to the drive, BR8 corresponds to bit 0 of the transmitted word, BR15 to bit 7, and BR7 to bit 15. The different relationships between received and transmitted words are a result of simplified ready circuitry (to be discussed later).

The ring counter is implemented with a 74193 binary up/down counter and a 74154 four-to-sixteen decoder. The drive's PLO clock is used as the count input signal. The ring counter is reset whenever a SECTOR pulse or a synch character is detected allowing BR0 to be activated on the next count. A ring counter provides a great deal of design flexibility. Disk control actions can be fine tuned with the availability of 16 outputs. Some of these key actions are reading from and writing to the serial/parallel conversion circuitry, generating ready and DMA request signals, and transmitting and checking CRC words.

An 8254 programmable interval timer provides timing delays. The 8254 must be used, rather than an 8253, due to the short output pulse widths (approximately 140 nsec) of the ring counter. The 8254 has three independent 16-bit counters which are initialized by the software to operate in the hardware triggered strobe mode (mode 5). Each counter accepts CLK and GATE inputs and provides a single OUT output. Each counter's count register is initialized with a count value and when the GATE input is activated, the count register is decremented with each CLK pulse received. When the count register is decremented to zero, a pulse is generated on the OUT output.

The three 8254 output signals are designated CNTR0, CNTR1, and CNTR2 and are associated with their respective counter. Details of the time delays are discussed later. In general, CNTR0 signals the start of an ID field during the format track operation or the start of a data field during the write data record operation. CNTR1 signals the end of the DMA transfer when the format track operation writes the ID field, when the write data record operation writes the data field, or when the read data record operation reads the data field. During both the read or write data record operations, CNTR2 signals the end of the DMA transfer used to compare the ID field (sector search).

When the 8254's counter 0 times out, the CNTR0_DETECT flip-flop is set. The CNTR0_DETECT signal enables the 8089's DMA transfer (write ID field or data field) and is reset by channel 1's task program at the completion of the transfer.

A synch character (0FH for ID field and 0DH for data field) must be detected to begin comparing an ID field or reading a data field. Only a single AND gate is required to detect the synch character since the DRIVE_READ_GATE signal is activated when the read/write heads are over a gap written with zeros.

Upon detection, the SYNCH_DETECT flip-flop is set. The SYNCH_DETECT signal enables the 8089's DMA transfer (write desired ID information or read data field) and is reset by channel 1's task program at the completion of the transfer.

When an I/O device's transfer rate approaches the 8089's maximum transfer rate (1.25 megabytes/sec), the DMA request (DRQ) input cannot be used to synchronize each byte or word transferred due to the 1.2 μsec maximum (at 5 MHz) latency of this input. The disk controller uses the 8089's ready signal (and wait states) to synchronize the SA4008's 889 kilobyte/sec transfer rate with the 8089's transfer rate. The DRQ inputs are used to enable DMA transfers while the ready signal is used to synchronize individual word transfers. Channel 1's DMA request signal, DRQ1, is activated when CNTR0_DETECT becomes valid (write ID field or data field) or 8 bit times after SYNCH_DETECT becomes valid (read data field). The 8-bit delay time allows the first word to be converted from serial to parallel before the 8089's DMA transfer begin. Channel 2's DMA request signal, DRQ2, is also activated 8 bit times after SYNCH_DETECT becomes valid (write desired ID information). DRQ1 and DRQ2 are deactivated by the task program upon completion of the DMA transfer.

The 8284A clock generator synchronizes ready signals from two buses. RDY1 is the ready signal from the Multibus and RDY2 is the ready signal from the local bus. Both ready inputs are normally inactive. When accessing memory or I/O devices, one ready input is activated to complete the bus transfer cycle. Depending on when the ready input is activated, wait states may or may not be inserted. In the disk controller, the 8089 may require wait states only when accessing the 16-bit disk data port. Wait states are not required when accessing other I/O devices or memory devices on the local bus. For these devices requiring no wait states, RDY2 is generated by the I/O read or write command.

Accessing the disk data port may require wait states to synchronize 8089 transfers with the drive. For this case, BR0 is used to set a flip-flop. The flip-flop's output enables RDY2 generation by the I/O read or write command. This ensures that previous data has been transferred to or from the serial/parallel converter before writing to the output buffer or reading the input buffer, respectively. Using only BR0 involved changing the relationships between ring counter outputs and actual data bits transmitted to the drive. A transmitted bit 0 corresponds to BR8 while a received bit 0 corresponds to BR0. This is required since a transmitted word must be preloaded into the output buffer (at data bit 8 time) before being transferred to the serial/parallel converter (to prevent underrun errors). On the other hand, a received word must be transferred from the serial/parallel converter to the input buffer before being read (at data bit 0 time). The input and output buffers are described later.

The external DMA termination signals, EXT1 and EXT2, are used to terminate dummy DMA transfers. EXT1 is generated whenever the 8254's counter 2 times out (signifying the end of ID field comparison) or whenever the drive's SECTOR or INDEX pulse is detected. The SEL_INDEX signal which is controlled by the task program selects which pulse generates EXT1 (0 for SECTOR and 1 for INDEX). This allows the SECTOR or INDEX pulse to terminate the dummy DMA transfer. EXT2 is also generated by either the SECTOR or INDEX pulse, qualified with SEL_INDEX.

Data Transfer

The data transfer section (Fig. 23) provides serial/parallel conversion, ID field comparison, and CRC generation and checking functions. Serial/parallel conversion is performed with a 16-bit shift register implemented with two 74S299 8-bit shift registers. Data read from the drive is converted from serial to parallel while data written to the drive is converted from parallel to serial.

A double buffered technique is used here. A 16-bit input buffer receives read data from the shift register and a 16-bit output buffer transmits write data to the shift register. Each buffer is implemented with a pair of 8282 octal latches. Two 8286 octal transceivers provide the interface between the local data bus and the input and output buffers. These transceivers are enabled when writing an ID field or a data field or when reading a data field. They are disabled during the ID field comparison.

The 16-bit comparator is implemented with four 74LS85 4-bit comparators and one 4-input NAND gate. During the ID field comparison, the transceivers are disabled allowing the input buffer which contains the ID information read from the disk to drive one input of the 16-bit comparator while the ID information written by the 8089 drives the other input. The comparator output is sampled during each 16-bit comparison. The first mismatch is latched (until reset) for channel 1's task program to examine later. This permits the length of the ID field to be any multiple of words.

The input buffer, output buffer, and comparator are all accessed via port 4000H. The CRC circuitry uses a 9401 CRC generator/checker strapped to use the CRC-CCITT polynomial, $X^{16}+X^{12}+X^5+1$. Immediately after reading the CRC word, the 9401's error output is latched allowing channel 1's task program to examine the CRC error status later.

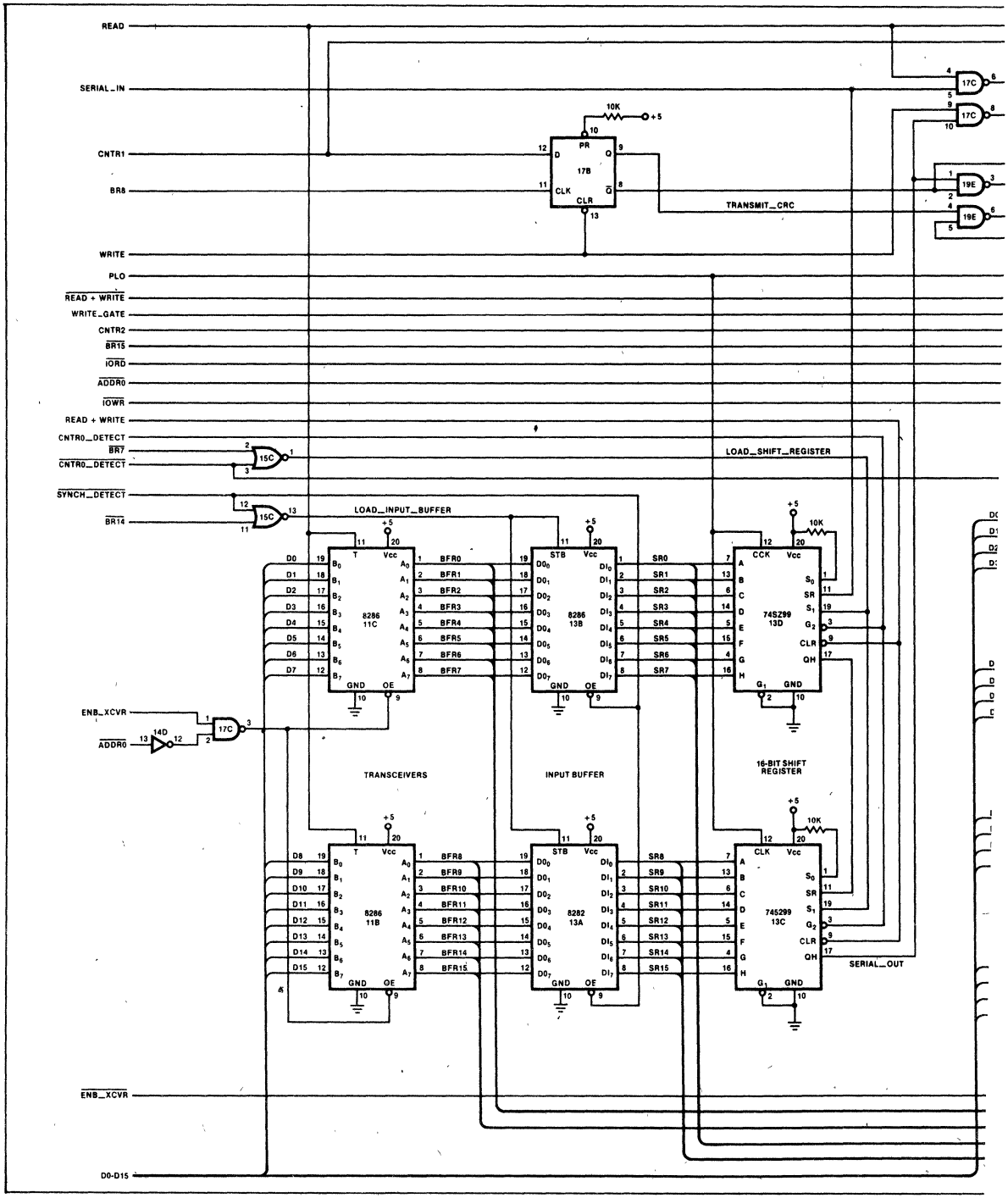


Figure 23. Data Transfer

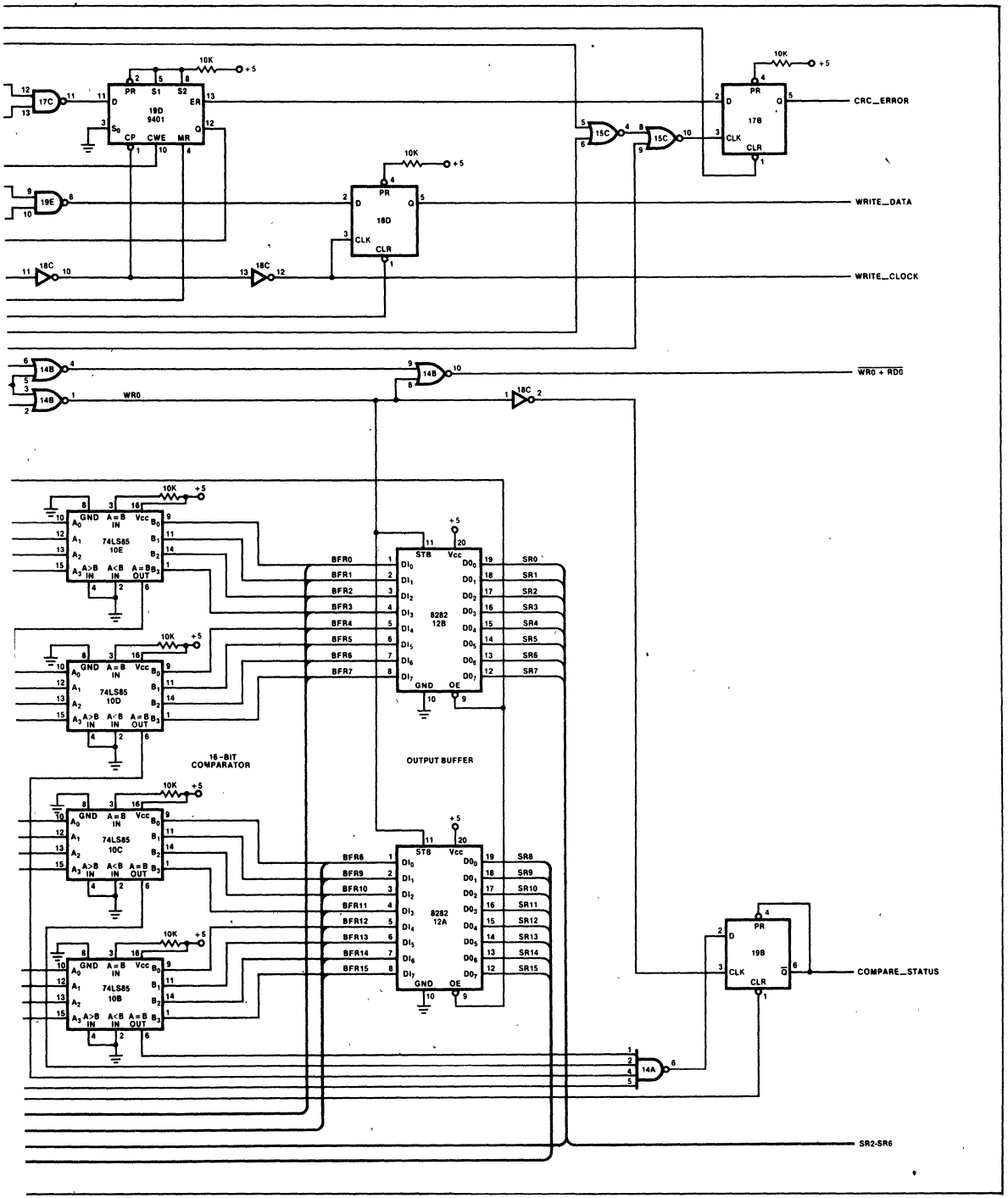


Figure 23. Data Transfer

VI. HARDWARE OPERATION

Now that an overview of the four disk control operations and the details of the hardware components have been presented, the detailed disk control operations will be discussed. The interaction of hardware components, the relative timing of signals, and the data flow are described for the format track, write data record, and read data record operations. The seek track operation is primarily implemented with software. The channel 1 and 2 task programs are discussed in the section on software operation. This discussion is focused on how the hardware operates. While reading the detailed description, it may be helpful to refer to the hardware schematics (Figs. 21, 22, and 23).

Format Track

The format track operation is preceded by a seek track operation where the proper cylinder is accessed and the proper head is selected. Upon detecting the INDEX pulse, the format track operation writes the ID data for 30 sectors and writes zeros everywhere else including the data field areas. Channel 1 controls the entire operation without assistance from channel 2.

The overall timing of the format operation is shown in Figure 24. The INDEX and SECTOR signals from the drive and the WRITE_GATE signal to the drive are shown. Also presented are the signals controlled by channel 1's task program—FORMAT, WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR. In addition, the activity of the 8254's counters is shown.

Channel 1 begins the format track operation by initializing the 8254 counters and its DMA registers used to transfer the ID data to the drive. The FORMAT signal is activated and a dummy DMA transfer is started to wait for the INDEX pulse. When the INDEX pulse is detected (Fig. 24), the hardware activates the WRITE_GATE signal and zeros are written on the track. A SECTOR pulse which coincides with the INDEX pulse starts counter 0. Counter 0 provides the time delay from the SECTOR pulse to the start of the ID field and indicates when to start writing the ID data. This provides the proper-sized gap between the SECTOR pulse and ID field.

Detection of the INDEX pulse also resumes channel 1's program execution and the WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR signals are activated (Figs. 24 and 25). Next the destination synchronized DMA transfer is started, the synch character word is prefetched from memory, and channel 1 waits for DMA request.

When counter 0 times out, the CNTR0_DETECT flip-flop is set (Fig. 25). CNTR0_DETECT is transmitted to the 8089's DMA request input, DRQ1. This starts the 8089 bus cycle which writes the synch character to the output buffer. CNTR0_DETECT is also transmitted to counter 1's gate input, GATE1, which allows counter 1 to start counting BR3 ring counter pulses. Counter 1 provides the time delay from the start to the end of the ID field and indicates when to append a CRC word.

The $\overline{\text{WR0}}$ signal is activated when the 8089 writes to the output buffer or the hardware comparator and is used by the ready circuitry to generate RDY2A. RDY2A is activated by BR0 or $\overline{\text{WR0}}$, whichever occurs last. This ensures that previous data has been transferred from the output buffer to the shift register before writing new data to the output buffer. When RDY2A is activated, the write bus cycle completes and the synch character is latched in the output buffer with the rising edge of $\overline{\text{WR0}}$. The synch character is next loaded into the shift register with BR7 and written to the drive.

The DMA activity repeats until four words have been transferred—synch character, first ID word, second ID word, and zero word. As the zero word is being written, counter 1 times out after counting four BR3 pulses. The TRANSMIT_CRC flip-flop latches CNTR1 with BR8 and remains active for one word time. The active TRANSMIT_CRC signal allows a CRC word to be serially transmitted to the drive from the 9401 CRC generator/checker. When TRANSMIT_CRC goes inactive, zeros are shifted out of the shift register to the drive. Zeros are written on the track until the next ID field since WRITE_GATE is held active until all 30 ID fields have been written.

After the four word DMA transfer, channel 1 initializes DMA registers in preparation for writing the next sector's ID data and starts a dummy DMA transfer to wait for the next SECTOR pulse. The same procedure is repeated until ID data has been written for all 30 sectors. The format track operation concludes with channel 1 deactivating FORMAT, WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR. The FORMAT signal deactivates WRITE_GATE which stops writing zeros to the drive.

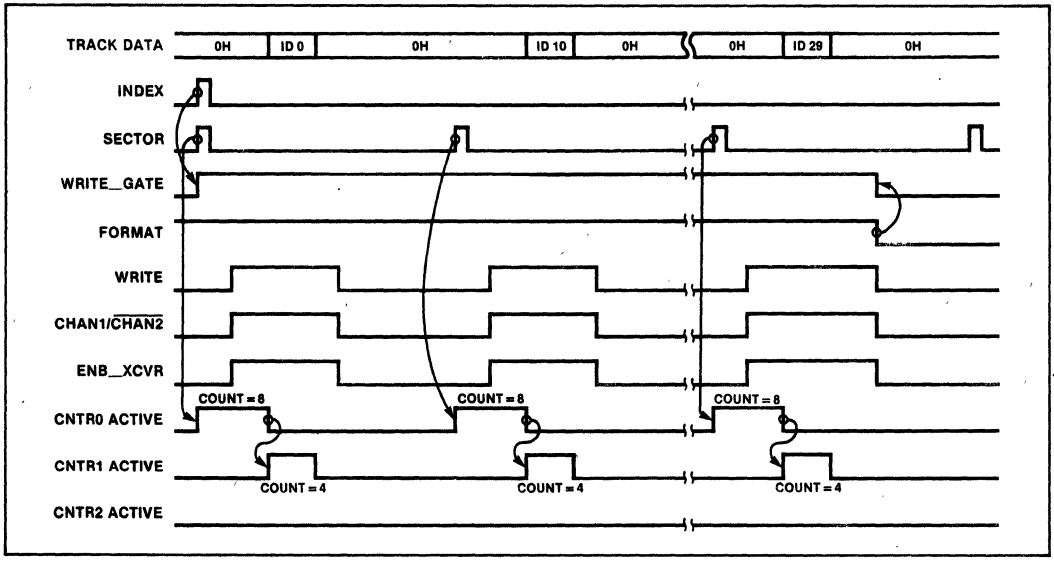


Figure 24. Format Track Operation

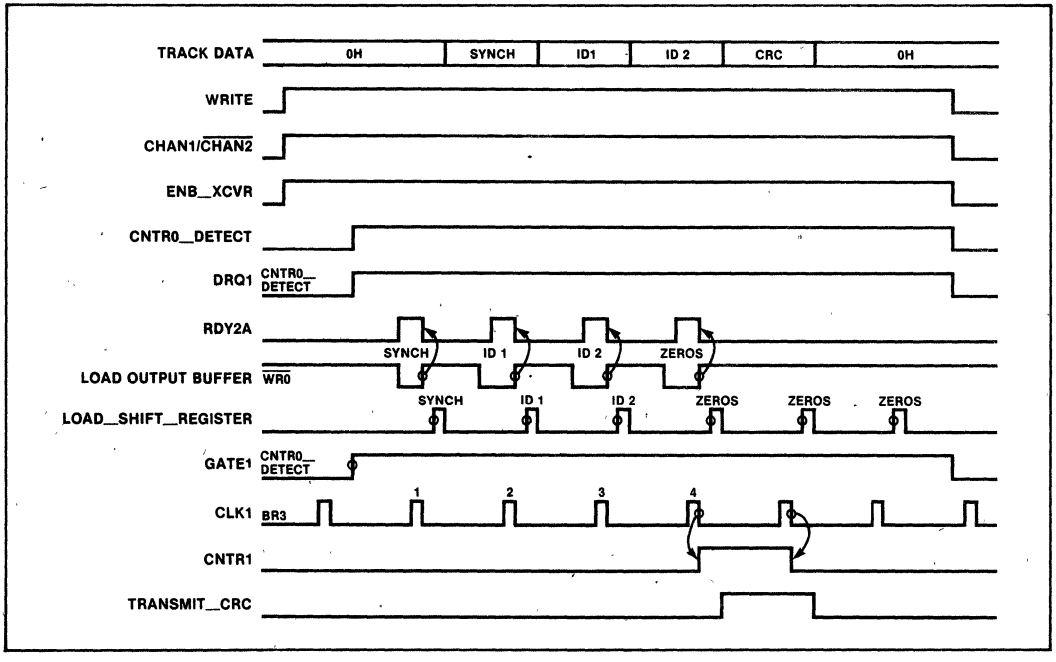


Figure 25. Write ID Field

AFN02057A

Write Data Record

The write data record operation consists of two phases—sector search and write data field. Channel 1's task program supervises this operation with assistance from channel 2. The sector search phase begins with the first complete sector that passes under the read/write heads and ends either when the desired sector is located or when all 30 sectors on the track have been compared without a match. If no match occurs, channel 1 aborts the operation and reports the error to the host processor. Upon locating the desired sector, the write data field phase begins. Two types of DMA transfers are performed during the write data record operation—channel 2 transfers the desired ID field information to the 16-bit comparator and channel 1 transfers the data record to the drive.

The overall timing of the write record operation is shown in Figure 26. The drive signals (SECTOR, READ_GATE, and WRITE_GATE), signals controlled by 8089 task programs (READ, WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR), and activity of the 8254 counters are displayed.

Channel 1 begins the write data record operation by initializing the 8254 counters and its DMA registers used to transfer the data record to the drive. Next channel 1 starts channel 2, initiates a dummy DMA transfer, and executes idle cycles. Channel 2 begins execution and initializes its DMA registers used to transfer the desired ID data to the 16-bit comparator. Next channel 2 starts a dummy DMA transfer to wait for a SECTOR pulse.

When the SECTOR pulse is detected (Fig. 26), channel 2 activates the READ signal. The destination synchronized DMA transfer is started, ID word 1 is prefetched from memory, and channel 2 waits for DMA request. The READ signal activates the drive's READ_GATE signal and the synch character detection circuitry reads data from the track. When the synch character is detected, the SYNCH_DETECT flip-flop is set (Fig. 27). The SYNCH_DETECT signal is used to start counters 0 and 2 (Figs. 26 and 27). Counter 2 provides the time delay from the start to the end of the ID field and indicates when to check for CRC errors. Counter 0 provides the time delay from the start of the ID field to the start of the data field and indicates when to start writing the data field. This provides the proper-sized gap between the ID and data fields.

SYNCH_DETECT also allows the DMA request signal, DRQ2, to be activated with BR7. This starts the 8089 bus cycle which writes ID word 1 to one input of the 16-bit comparator. BR14 is used to generate the LOAD_INPUT_BUFFER signal which latches the drive's ID word 1 in the input buffer (from the shift register). The input buffer drives the other comparator input. Note that the ENB_XCVR signal is inactive and the transceivers between the local data bus and the double-buffered serial/parallel converter are off. CNTR0_DETECT is also inactive which deactivates the output buffer.

The ready circuitry operates in an identical way as during the format operation. When RDY2A is activated, the write bus cycle completes and the COMPARE_STATUS is latched with the rising edge of $\overline{\text{WR0}}$. The COMPARE_STATUS flip-flop keeps the first mismatch latched until reset.

Counter 2 was set up to count three BR7 pulses. After both ID words have been compared, counter 2 times out. The CNTR2 signal allows the 9401 CRC generator/checker's error output to be latched in the CRC_ ERROR flip-flop with BR7. Channel 2 halts after the DMA transfer. The CNTR2 signal is also used to activate channel 1's external terminate input, EXT1. Channel 1 resumes execution, examines the COMPARE_STATUS and CRC_ERROR flip-flops, and deactivates the READ signal.

Upon detecting a match without CRC error, channel 1 begins the write data field phase by activating WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR (Fig. 26). The destination synchronized DMA transfer is started, the synch character word is prefetched from memory, and channel 1 waits for DMA request. When counter 0 times out, CNTR0_DETECT is activated and counter 1 is started. Counter 1 provides the time delay from the start to the end of the data field and indicates when to append a CRC word. CNTR0_DETECT is also transmitted to the 8089's DMA request input, DRQ1, which starts the data record transfer to the drive (Fig. 28). The data record is written on the track almost identically to the way that the ID data is written on the track during the format track operation. The only hardware operational difference is that more words are written on the track for the data record than for the ID field. The earlier discussion explains the operation of Figure 28 and therefore will not be repeated here. The write data record operation concludes with channel 1 deactivating WRITE, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR.

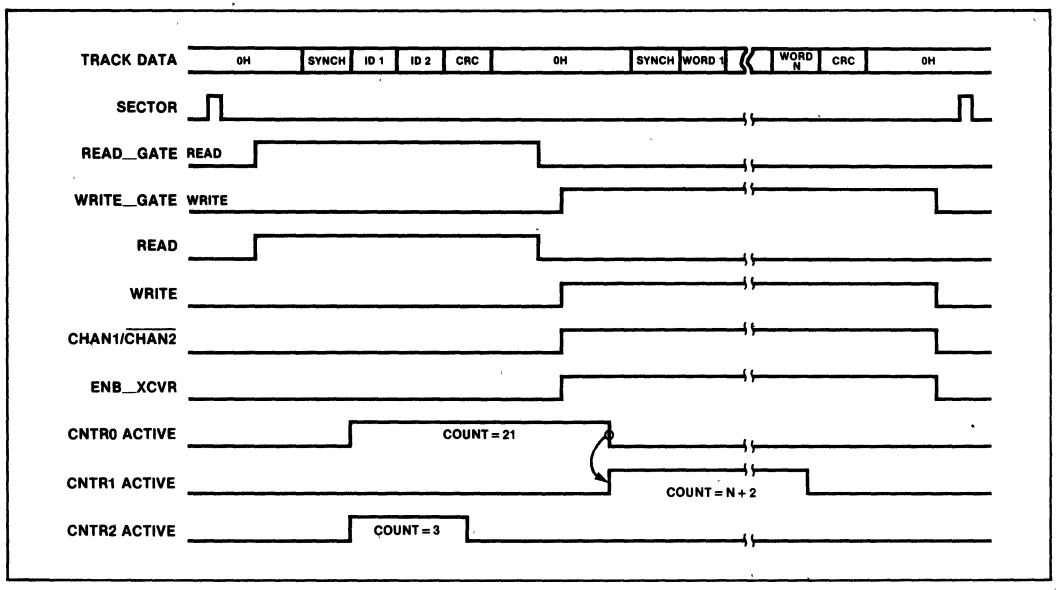


Figure 26. Write Data Record Operation

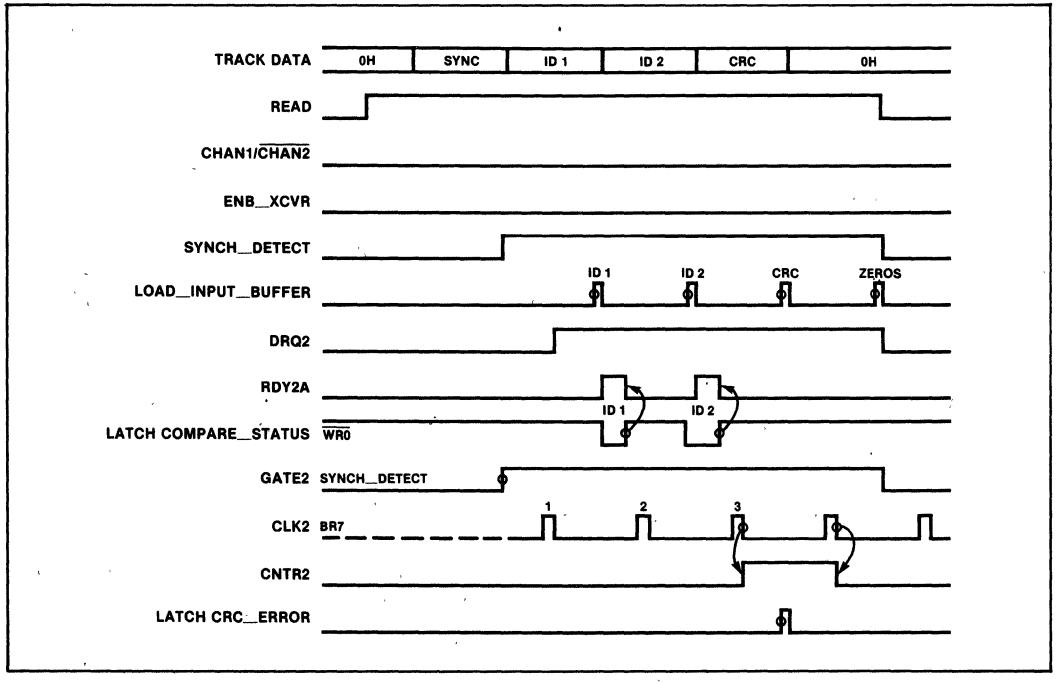


Figure 27. Compare ID Field

AFN02057A

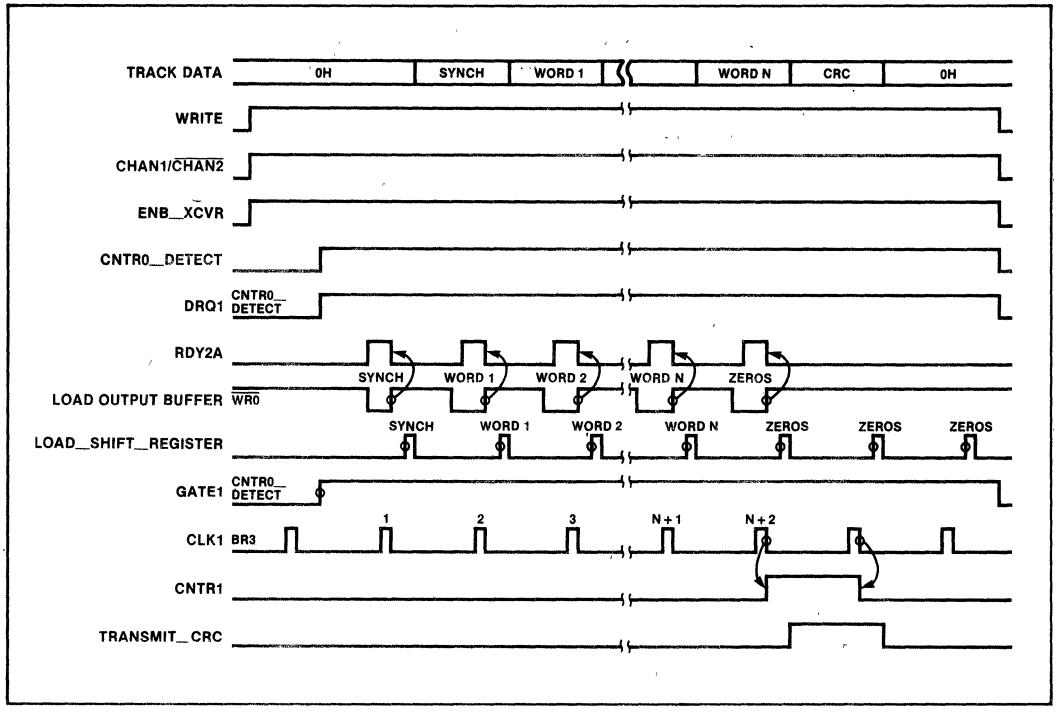


Figure 28. Write Data Field

Read Data Record

The read data record operation (Fig. 29) is similar to the write data record operation. Although counter 0 is activated, it is not used during this operation. The sector search activity by channel 1 and 2 is identical to that of the write record operation. Only channel 1's activity after locating the desired sector is different. Channel 1 reads the data record instead of writing it.

After the desired sector is located without a CRC error, channel 1 begins the read data field phase by activating READ, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR. The source synchronized DMA transfer is started and channel 1 waits for DMA request. The READ signal activates the drive's READ_GATE signal and the synch character detection circuitry reads data from the track. When the synch character is detected, the SYNCH_DETECT flip-flop is set (Fig. 30). The SYNCH_DETECT signal is used to start counter 1. Counter 1 provides the time delay from the start to the end of the data field and indicates when to check for CRC errors.

BR14 is used to generate the LOAD_INPUT_BUFFER signal which latches the first data word in the input buffer (from the shift register). SYNCH_DETECT also allows the DMA request signal, DRQ1, to be activated with BR7. This starts the 8089 bus cycle which reads the first data word from the input buffer.

The $\overline{\text{RD0}}$ signal is activated when the 8089 reads the input buffer and is used by the ready circuitry to generate RDY2A. RDY2A is activated by BR0 or $\overline{\text{RD0}}$, whichever occurs last. This ensures that data has been loaded into the input buffer before reading it. Recall that during the format and write record operations, the ready circuitry used $\overline{\text{WR0}}$ instead of $\overline{\text{RD0}}$. When RDY2A is activated, the read bus cycle completes and the 8089 stores the data word in memory.

The DMA activity repeats until all data words have been read. Counter 1 times out and CNTR1 allows the 9401 CRC generator/checker's error output to be latched in the CRC_ERROR flip-flop with BR7. The DMA transfer terminates and channel 1 examines the CRC_ERROR flip-flop. The read data record operation concludes with channel 1 deactivating READ, CHAN1/$\overline{\text{CHAN2}}$, and ENB_XCVR.

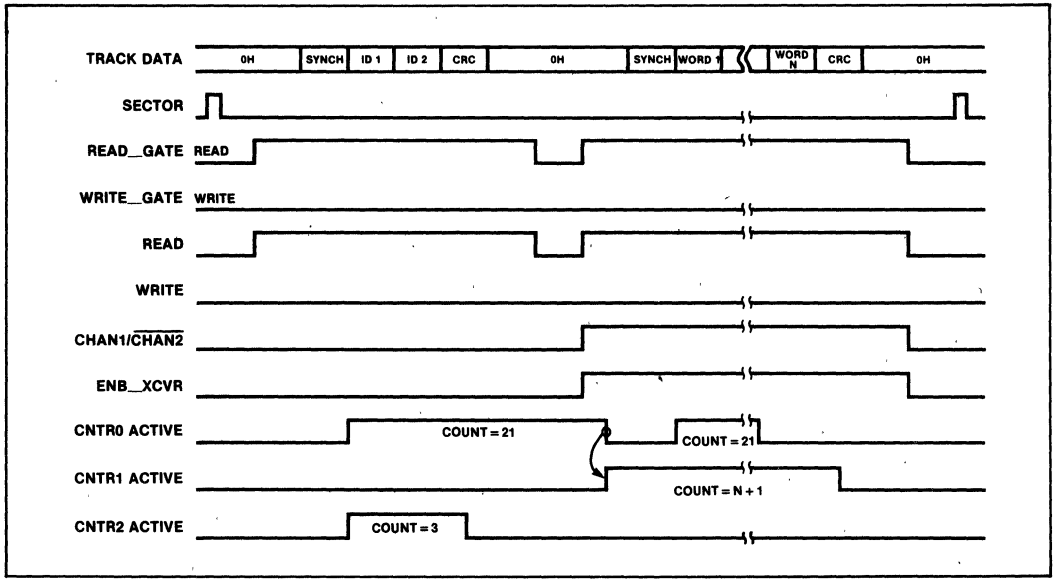


Figure 29. Read Data Record Operation

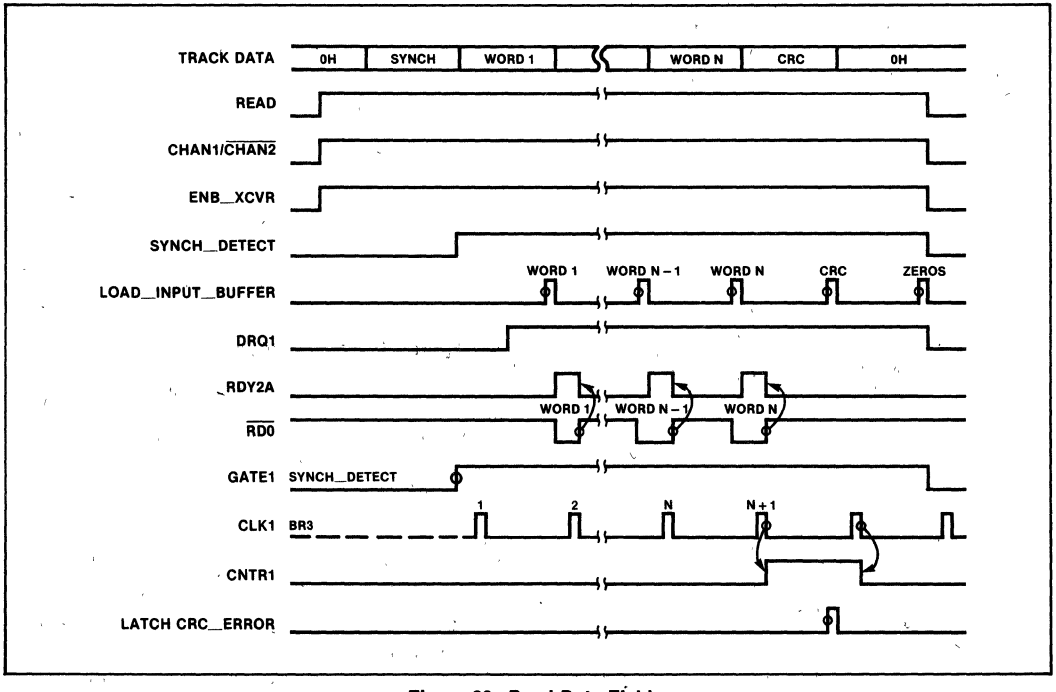


Figure 30. Read Data Field

AFN02057A

VII. SOFTWARE DESIGN

The host processor communicates with and starts only channel 1 and subsequently channel 1 starts channel 2. Although the 8089's architecture and the controller hardware permit the host processor to control and start both channels, this design restricts the host's interactions with channel 1.

In a previous section, the linked blocks of the memory-based communication structure are described. The system configuration pointer and the system configuration block are used only during 8089 initialization after reset. The channel control block (CB) is used for 8089 initialization and to control channel operation. Before starting channel operation, the host processor initializes the channel control word and the parameter block offset and segment base in the proper half of the channel control block (Fig. 8). This section describes the parameter and task blocks used in the disk controller design.

Parameter Blocks

The parameter block for channel 1 is shown in Figure 31. The TB1 offset points to channel 1's task program which resides in local memory. If the task program resides in system memory, such as during initial debugging, TB1 segment base is also used to generate the pointer. Note that the 8089's architecture requires that the first parameter in the PB be the task program's address. All other parameters are user-defined allowing parameters to be tailored for a specific I/O task. Other PB1 parameters that are passed to the 8089 in this application are the data buffer's address, function, cylinder, head, sector and pointers to the CB and PB2. The only parameter passed back to the host is status.

Normally the host processor starts channel 2 and is responsible for initializing parameters in the CB and PB2. In this design channel 1 starts channel 2. The CB and PB2 pointers received from the host in PB1 allow channel 1 to initialize the proper parameters before starting channel 2.

In this disk controller design, channel 2 is essentially a slave of channel 1. Prior to starting channel 2, channel 1 initializes channel 2's CCW and PB2 offset and segment base in the second half of the channel control block. Next channel 1 initializes three parameters in channel 2's parameter block (Fig. 32). The first parameter is the address of channel 2's task program. The function code and the data buffer's address are the other two parameters. Although parameter block 2's structure allows the task program and data buffer to reside in system or local memory, this design places them both in local memory. Therefore, only TB2 offset and data buffer offset are initialized by channel 1 and the segment bases are not used. Channel 2 provides no status information back to channel 1 via parameter block 2.

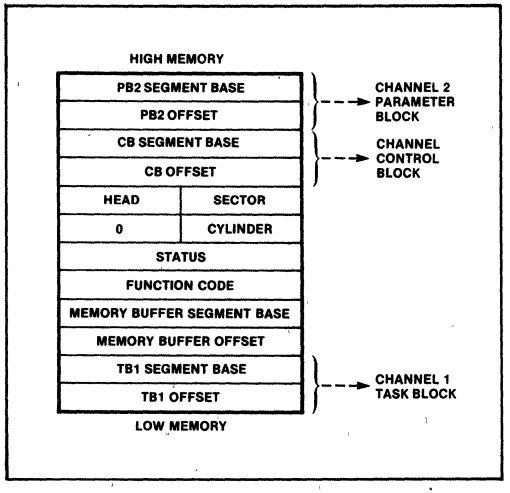


Figure 31. Channel 1 Parameter Block

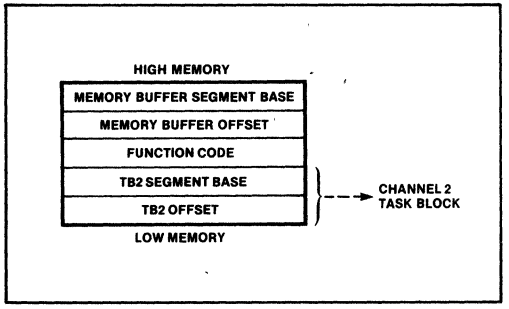


Figure 32. Channel 2 Parameter Block

Software Organization

The disk controller software is organized as several modules with a three-level hierarchy (Fig. 33). When the 8089 receives a channel attention from the host processor, module TBLK1 begins execution (level 1). Control is next transferred to one of the level 2 modules (INIT, SEEK, FMAT, WDATA, or RDATA) based on which function was specified in the parameter block. For read or write data record functions, TBLK2, which is the lone level 3 module, is also executed.

The details of each software module are now described. While reading the detailed description, it may be helpful to refer to the ASM89 assembly language source code in Appendix B.

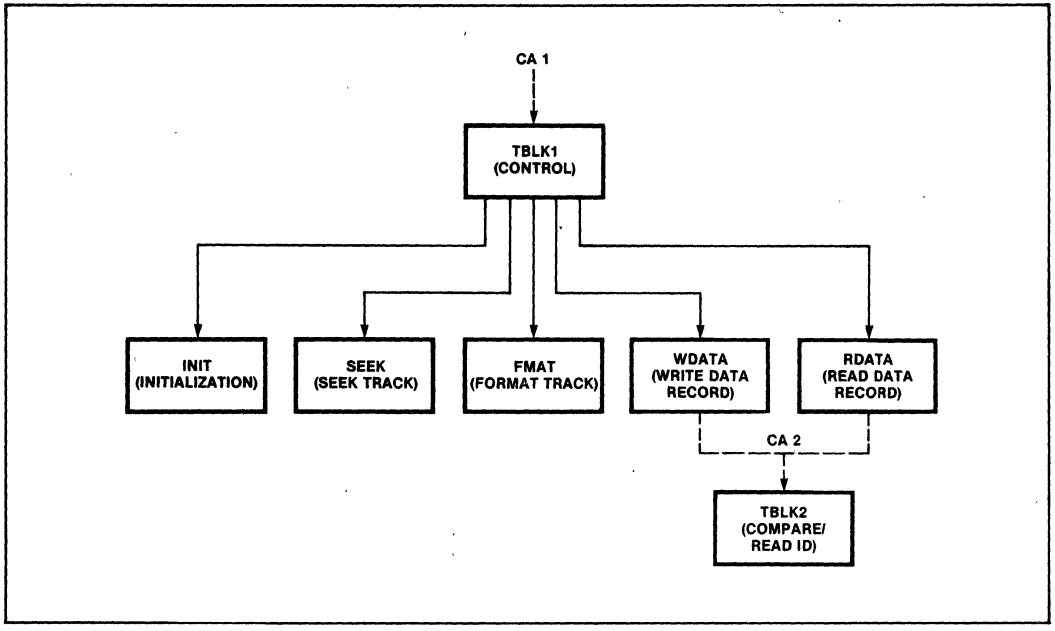


Figure 33. Disk Controller Software Organization

Control Program (TBLK1)

After the host processor initializes parameters in the channel control and parameter blocks, a channel attention is generated which starts module TBLK1. Registers GA and GC are first initialized. GA is used as a pointer to the start of local RAM and GC is used as a pointer to control port 1. The FORMAT, READ, WRITE, CHAN1/$\overline{\text{CHAN2}}$, ENB_XCVR, and SEL_INDEX control signals are generated by writing to control port 1. In general, the controller software uses GA as a base pointer when accessing variables in local memory and GC as a base pointer when accessing I/O ports.

Next TBLK1 examines the function code in the parameter block to determine which function has been specified. A unique bit in the function code is used to specify each of the five functions. This allows the 8089's bit test and branch instructions to be used. If a valid function is specified, control is transferred to the proper level 2 module. If not, the BAD_CODE error bit in the parameter block's status word is set, the host is interrupted, and channel 1 halts.

Initialization (INIT)

The initialization module, INIT, is used to place the controller in a known state after applying power to the system. It is also used to reset the drive's write fault signal. Control ports 1 and 2 are first cleared and then the drive select line is activated. Any pending write faults are reset. The heads are next positioned over cylinder 0 and the three 8254 counters are initialized in preparation for other disk drive operations. Counter 0 is initialized to count 8 pulses, counter 1 to count 4 pulses, and counter 2 to count 3 pulses. Finally, the host is interrupted and the channel halts.

Seek Track (SEEK)

The seek track module, SEEK, is used to position the heads over a specified cylinder and to select one of the eight read/write heads. This module first checks if the controller is initialized. Since INIT selects the drive, an active ready signal from the drive indicates that the controller has been initialized. If not initialized, the NOT_READY error bit in the status word is set, the host is interrupted, and the channel halts. In order to minimize unnecessary accesses to Multibus, a status word in local memory is updated as errors are encountered. Prior to halting, a module will copy this local status word to the parameter block in Multibus's shared memory.

If the drive is initialized, execution of the SEEK module continues. The cylinder and head values are copied from the parameter block to local memory. These variables

are stored in local memory to minimize Multibus access. The cylinder and head values are checked to determine whether they exceed the maximum values of the drive. If one or both does, the BAD_CYLINDER and/or BAD_HEAD error bits are set and the channel halts.

With valid input parameters, head movement is next determined using a local variable, PRESENT_CYL, which specifies which cylinder is presently being accessed. By subtracting the present cylinder value from the new cylinder value, the head movement is determined. A zero result means no movement, a positive result means inward movement, and a negative result means outward movement. A non-zero result also specifies how many cylinders inward or outward the set of heads must be moved. Although the 8089 does not have a subtract instruction, the subtract operation is easily implemented by complementing the subtrahend before adding it to the minuend. If head movement is necessary, the drive's direction line is activated (1 for inward and 0 for outward) and a string of pulses equal to the number of cylinders to be moved is transmitted to the drive.

Next the PRESENT_CYL variable is updated and a 20 msec delay loop is executed. This delay is required by the drive to allow the head positions to stabilize. Finally, the host is interrupted and channel 1 halts.

Format Track (FMAT)

Before information can be stored on a track, the ID fields must be written. This is the function of the format track module, FMAT. Similar to the SEEK module, controller initialization is first checked. Next the count registers for the 8254 counters 0 and 1 are initialized to 8 and 4, respectively. A format table is generated which contains four words of information that are written on the track for each of the 30 sectors. The four words contain the ID synch character, cylinder number, head and sector numbers, and a word of zeros. The zero word is used to write zeros on the track between ID fields. This area contains the gap between ID and data fields, the data field, the gap after the data field, and the gap after the subsequent SECTOR pulse. Only one zero word is needed since the 16-bit shift register continues to shift out zeros until it is reloaded.

The format table is generated in three steps: an array containing the 30 interleaved sector numbers is constructed, the head number is loaded into the upper half of the MC register, and then four words for each sector are assembled in the table. Loading the head number into MC's upper half is effectively done by shifting the data from MC's lower half to its upper half. Although the 8089 has no shift instruction, the shift left operation can be implemented by adding a number to itself. Shifting the head number left 8 bits is easily accomplished with a loop containing just a few lines of code.

After the format table has been constructed, the information is written to the drive using high speed DMA transfers. Channel 1 performs the entire format operation without assistance from channel 2. Dummy DMA transfers are used to synchronize the format operation with INDEX or SECTOR pulses received from the drive. The byte count (BC) register is initialized with the actual byte count plus two since the dummy DMA transfer decrements BC (refer to the section on Special Design Considerations). After the synchronization signal is received, four words from the format table are written on the track with DMA transfers. The first sector's ID field is written after the INDEX pulse is detected and the ID fields of the remaining 29 sectors are written after SECTOR pulses are detected.

After each of the 30 ID fields has been written on the track, the drive's write fault signal is examined. If a fault is detected, the BAD_WRITE error bit is set and the channel halts. If no faults are detected, the channel halts after all 30 ID fields have been written.

Write Data Record (WDATA)

The WDATA module begins execution whenever a data record is written to the drive. Channel 1 begins by transferring the desired sector's ID information from the parameter block to a local memory buffer. This local buffer will be used by channel 2 during the ID field compare. The sector number is checked to determine whether it exceeds the maximum value. If so, the BAD_SECTOR error bit is set and channel 1 halts. If no error is detected, the 8254's count registers for counters 0 and 1 are initialized to 21 and 258, respectively.

Channel 1 next enters the DMA mode and transfers the data record from the system memory buffer to a local memory buffer. The data synch character is inserted into this local buffer before the data record and a zero word is inserted after the data record. The zero word causes zeros to be written after the data record and CRC word.

Preparation for starting channel 2 is next performed. Channel 2's half of the channel control block is loaded with the channel control word to start task program exe-

cution in local memory and with the offset and segment base values of parameter block 2's address. Channel 2's task program address, the function code for compare ID field, and the address of the buffer containing the desired sector's ID information are then loaded into channel 2's parameter block. Next channel 1's registers for the write data record DMA transfer are initialized, a channel attention signal to start channel 2 is generated, and channel 1 starts a dummy DMA transfer. Note that two must be added to BC since it is decremented during the dummy DMA transfer.

Channel 1 now idles while channel 2 detects a SECTOR pulse and transfers the desired sector's ID information to the 16-bit comparator. As channel 2 completes its DMA transfer and halts, counter 2 times out which terminates channel 1's dummy DMA transfer. Channel 1 resumes execution and examines the compare status and CRC error flip-flops simultaneously. This is accomplished using the 8089's jump if masked compare not equal (JMCNE) instruction which uses the MC register to test both flip-flop outputs and jumps if a mismatch and/or CRC error is detected. If a match without CRC error is detected, channel 1 enters the DMA mode and writes the data record on the disk.

If a mismatch and/or CRC error is detected, the CRC error flip-flop is checked individually. The detection of a CRC error causes the BAD_ID_CRC error bit to be set and the channel to halt. Detecting no CRC error means that only a mismatch occurred. In this case, the next sector's ID field is compared by starting channel 2 again.

Assuming that no CRC errors are detected, the sector search is repeated until a match is found or all 30 ID fields have been compared, whichever comes first. This technique allows the sector search to begin with the first complete sector encountered rather than starting at the beginning of the track when the INDEX pulse is detected.

After detecting a match and writing the data record on the track, the drive's write fault signal is examined. The BAD_WRITE error bit is set if a fault is detected. Otherwise, channel 1 halts. For the case where all 30 sectors have been searched and the desired sector is not found, the BAD_SEARCH error bit is set and channel 1 halts.

Read Data Record (RDATA)

Whenever a data record is to be read from the drive the RDATA module is executed. Much of the actions performed by this module are identical to that of the WDATA module. Channel 1 also begins by transferring the desired sector's ID information from the parameter block to a local memory buffer, checking the sector number, and initializing the 8254 count registers. Identical action continues by updating channel 2's communication blocks, initializing channel 1's registers for the DMA transfer, generating a channel attention signal to start channel 2, and starting a dummy DMA transfer. Since the read data record DMA transfer is source synchronized, the BC register is not modified during the dummy DMA transfer and therefore no adjustment is needed when initializing BC.

Channel 2 next performs the ID field compare and halts. Channel 1 resumes execution when counter 2 times out. Identically with WDATA, channel 1 examines the compare status and CRC error flip-flop simultaneously. Detecting a match without CRC error causes channel 1 to enter the DMA mode and read the data record. The CRC error flip-flop is again examined and if no error is detected, the data record just read into a local memory buffer is transferred to the system memory buffer with DMA transfers and channel 1 halts. If a CRC error was detected during the reading of the data record, the BAD_DATA_CRC error bit is set and channel 1 halts.

Detection of a mismatch and/or CRC error after the ID field compare causes the CRC error flip-flop to be checked individually. Encountering a CRC error will set the BAD_ID_CRC error bit and halt channel 1. Otherwise channel 1 will repeat the sector search until a match is found, all 30 ID fields are compared, or a CRC error is detected. Any one of these conditions will cause channel 1 to read the data record and halt or set an error bit and halt.

Compare or Read ID (TBLK2)

Channel 2's task program, TBLK2, is executed whenever the ID field is compared or read. Note that the code to read the ID field is included in TBLK2 but is not used in this version of the software. Channel 2 begins by reading the function code to determine whether to compare the ID field or to read it. In either case, the major actions are similar. Channel 2's DMA registers are initialized, a dummy DMA transfer is started to wait for the SECTOR pulse, the data transfer DMA mode is entered, and finally channel 2 halts. During an ID field compare, the data transfer DMA mode writes information to the 16-bit comparator while during an ID field read, information is read from the serial/parallel conversion circuitry. The BC register must be adjusted during the ID compare but not during the ID read.

VIII. POSSIBLE ENHANCEMENTS

As discussed earlier, the main purpose of this application note is to present basic design information on implementing a disk controller with the 8089 I/O processor. Although the design described here does not exhibit many intelligent features, the controller does allow software enhancements to provide the desired features.

The present design requires a separate track seek operation before a read or write data record operation. Adding the capability to perform the seek operation prior to reading or writing the data record is simple. Separate bits in the function code word are used to specify each function. This allows the host to select multiple functions. Recall that the function code is included in the parameter block and is initialized by the host processor.

The SEEK software module can be modified to examine the read and write function code bits after completing the seek operation. If only one function (read or write, but not both) is specified, control is transferred to the proper module, either RDATA or WDATA. Otherwise, an error bit is set and the channel halts. Note that this same technique can be used to perform a seek operation prior to the format track operation.

Another possible enhancement is the ability to retry an operation when a CRC error is detected. This feature applies whenever the ID field (during sector search) or data field (during read data record) is read. The software can be modified to reposition the heads at the failing sector (by counting SECTOR pulses) and retry the search or read operation. If several more CRC errors are detected, the operation is terminated, an error bit is set, and the channel halts. The number of retries can be preset in the task program or received as a variable from the host processor via the parameter block.

The ability to transfer multiple sectors of data is another desirable feature. A new variable called record count must be added to the parameter block. Sequential logical sectors are transferred from the starting logical sector specified in the parameter block. As many sectors as specified by the record count are transferred. This could also include head switching from one track to another (without a seek operation) to access data across track boundaries.

The transferred data is buffered in local memory and the interleaved scheme allows two physical sector times for the 8089 to transfer the data from system memory to local memory (write operation) or from local memory to system memory (read operation). Data is transferred to or from the multiple sector system memory buffer starting at the location specified by the parameter block variables. Another parameter block variable may be created which returns the last sector number transferred to the host. This information can be used by the host during an error to determine how many sectors were successfully transferred.

The ability to perform linked operations might be useful. For example, a track seek and the reading of five data records can be followed by another track seek and the writing of two data records. To include this feature, the parameter block could be modified to pass a set of parameters for each operation or multiple parameter blocks could be linked together. Variables such as function code, data buffer's address, cylinder, head, sector, record count, status, and last sector transferred are provided for each operation. As many sets of parameters as desired can be specified. The controller software would sequence through these sets of parameters, perform the required operations, and halt when a special function code, such as one with no functions selected, is detected.

It was pointed out earlier that the controller hardware includes provisions for reading the ID field. In addition, the software module TBLK2, channel 2's task program, can either compare the ID field or read it, depending on the function code that channel 1 provides. Therefore, the software can be modified to read the ID field information and verify track position. The 30 ID fields can also be read to verify a format track operation. In addition, sophisticated access methods which require reading the ID field may be implemented.

Another enhancement is to verify a data record just written to the drive. Here the same circuitry used to compare ID fields is used to compare data fields. The good data is written to one input of the hardware comparator while data read from the drive is applied to the other input. The first mismatch is latched in the compare status flip-flop for examination later.

The software can also be enhanced to manage a file structure. The host processor would refer to data records by logical file names rather than physical disk locations (cylinder, head, and sector). By maintaining a disk directory, the software would determine where the record is located or will be located and perform the data record access. The 8089's general instruction set, although oriented towards I/O processing, supports data processing of this complexity.

The 8089's flexible memory-based communication structure allows enhancements to be easily implemented. Modifying the parameter block to accommodate any additional parameters is a simple task. All variables in the parameter block except for the task program address are defined by the user based on the I/O processing task to be performed.

IX. CONCLUSIONS

This application note has provided a detailed description of a hard disk controller design based on the Intel 8089 I/O processor. The features provided by the 8089 make it well suited for disk control applications. The 1.25 megabyte/sec DMA transfer rate allows interfacing with high speed Winchester disk drives. The two channels provided in a single 40-pin package permit back-to-back DMA transfers in rapid succession to minimize gaps between the ID and data fields and provide a higher formatted drive capacity. The bit manipulation instructions simplify the implementation of the disk controller software, typical of I/O processing software. All of these features allow the design of a versatile, intelligent and high performance disk controller compatible with high performance microprocessors and disk drives available today.

An 8089-based disk controller maximizes overall system throughput. The host processor and 8089 operate concurrently due to the 8089's local bus which is used to access the controller circuitry, task programs, and local data variables and buffers. Shared system bus accesses are kept to a minimum which minimizes system bus contention. System throughput is also maximized by offloading disk control overhead tasks from the host and having the 8089 perform these tasks in parallel with the host. This frees host processor time for data processing.

A versatile disk controller with many intelligent features is easily implemented with an 8089. The host initiates a single high level command to perform track seek, data record transfers, error checking, and any retries. Other controller features such as multiple sector transfers, linked operations, and data record verification can also be provided. The 8089 provides flexible system bus interfacing. The controller described here has a Multibus interface with byte swap circuitry that permits interfacing with 8- or 16-bit system memory. Since the system bus width is defined during 8089 initialization, no controller hardware or software changes are necessary. Memory based communications allow both 8- and 16-bit host processors to use this controller.

Use of the 8089 promotes modular subsystem development. Memory based communication blocks provide a simple software interface with the host processor. Once the parameter block structure is defined, host and 8089 software development proceeds in parallel. Future enhancements are also easily incorporated with possible additions to the parameter block. The hardware interface is also straightforward. A system bus interface, such as Multibus, allows the use of address signals to generate the CA and SEL signals received by the 8089 and the use of the interrupt lines to route interrupts back to the host processor. Such a simple interface permits the disk controller hardware to be developed concurrently with other hardware subsystems. Also, note that the entire 8089 subsystem may be changed with minimal impact, if any, to the host processor software. For example, the subsystem could be upgraded to support higher capacity disk drives or a bubble memory subsystem could be implemented using a similar software interface.

Finally, the 8089 allows a compact disk controller to be implemented. The design here is constructed on a 6-3/4 by 12 inch board with 75 IC packages. By combining attributes of a CPU and an intelligent DMA controller in a single 40-pin package, the 8089 I/O processor allows versatile, high performance, and compact I/O subsystems to be implemented.

APPENDIX A

SHUGART SA4000 PERFORMANCE AND FUNCTIONAL SPECIFICATIONS

MODEL	4004	4008
No. of Disk Surfaces	2	4
No. of Heads	4	8
No. of Cylinders	202	202
No. of Tracks	808	1616
Gross Capacity (M bytes)	14.54	29.08
Access Time including seek settle of 20 ms (Milliseconds)		
One Track	20	20
Average (67 Track Seek)	65	65
Maximum (201 Track Seek)	140	140
Disk Speed	2964 RPM	
Recording Mode	MFM	
Recording Density	5534 BPI	
Flux Density	5534 FCI	
Track Capacity	18000 Bytes	
Track Density	172 TPI	
Transfer Rate	7.11×10^6 bits/sec. 889×10^3 bytes/sec.	
Sectors	Programmable	
Start Time	1.5 minutes	

APPENDIX B

```
8089 MACRO ASSEMBLER      *** 8089-BASED DISK CNTLR ***

ISIS-II 8089 MACRO ASSEMBLER X202 ASSEMBLY OF MODULE HDC89
OBJECT MODULE PLACED IN :F1:HDC89.OBJ
ASSEMBLER INVOKED BY:  :F1:ASM89  :F1:HDC89.A89  DATE(7-20-81)

LINE SOURCE

   1 $TITLE(*** 8089-BASED DISK CNTLR ***)
   2          ;
   3          ;         8089-BASED HARD DISK CONTROLLER
   4          ;
   5          ;
   6 HDC89    SEGMENT
   7          ;
   8          ;
   9 $INCLUDE(:F1:EQU89.A89)
  10          ;
  11          ;
  12          ;         CHANNEL 1 PARAMETER BLOCK OFFSETS
  13          ;
  14          PB1_TB1_OFF              EQU     00H
  15          PB1_TB1_SEG              EQU     02H
  16          PB1_BUFR_OFF             EQU     04H
  17          PB1_BUFR_SEG             EQU     06H
  18          PB1_FUNCTION             EQU     08H
  19          PB1_STATUS               EQU     0AH
  20          PB1_CYLINDER             EQU     0CH
  21          PB1_HEAD_SECTOR          EQU     0EH
  22          PB1_CCB_OFF              EQU     10H
  23          PB1_CCB_SEG              EQU     12H
  24          PB1_PB2_OFF              EQU     14H
  25          PB1_PB2_SEG              EQU     16H
  26          ;
  27          ;
  28          ;         CHANNEL 2 PARAMETER BLOCK OFFSETS
  29          ;
  30          PB2_TB2_OFF              EQU     00H
  31          PB2_TB2_SEG              EQU     02H
  32          PB2_FUNCTION             EQU     04H
  33          PB2_BUFR_OFF             EQU     06H
  34          PB2_BUFR_SEG             EQU     08H
  35          ;
  36          ;
  37          ;         CHANNEL 2 FUNCTION CODES
  38          ;
  39          CMP_ID                   EQU     00H
  40          READ_ID                  EQU     01H
  41          ;
  42 $EJECT
```

```
43                ;
44                ;
45                ;       8089 CHANNEL CONTROL REGISTER BIT MASKS
46                ;
47                PORT_TO_PORT            EQU     0000000000000000B
48                BLOCK_TO_PORT           EQU     0100000000000000B
49                PORT_TO_BLOCK           EQU     1000000000000000B
50                BLOCK_TO_BLOCK          EQU     1100000000000000B
51                ;
52                TRANSLATE               EQU     0010000000000000B
53                ;
54                SOURCE_SYNCH            EQU     0000100000000000B
55                DEST_SYNCH              EQU     0001000000000000B
56                ;
57                GA_SOURCE               EQU     0000000000000000B
58                GB_SOURCE               EQU     0000010000000000B
59                ;
60                LOCKED_CONTROL          EQU     0000001000000000B
61                ;
62                CHAINED_MODE            EQU     0000000100000000B
63                ;
64                SINGLE_XFER             EQU     0000000010000000B
65                ;
66                EXT_TERM_0              EQU     0000000000100000B
67                EXT_TERM_4              EQU     0000000001000000B
68                EXT_TERM_8              EQU     0000000001100000B
69                ;
70                BC_TERM_0               EQU     0000000000001000B
71                BC_TERM_4               EQU     0000000000010000B
72                BC_TERM_8               EQU     0000000000011000B
73                ;
74                UNTIL_MC_TERM_0         EQU     0000000000000001B
75                UNTIL_MC_TERM_4         EQU     0000000000000010B
76                UNTIL_MC_TERM_8         EQU     0000000000000011B
77                ;
78                WHILE_MC_TERM_0         EQU     0000000000000101B
79                WHILE_MC_TERM_4         EQU     0000000000000110B
80                WHILE_MC_TERM_8         EQU     0000000000000111B
81                ;
82  $EJECT
83                ;
84                ;
85                ;       CONTROLLER ADDRESSES
86                ;
87                RAM_BASE                EQU     0000H
88                ROM_BASE                EQU     2000H
89                DATA_PORT               EQU     4000H
90                CNTL_PORT_1             EQU     4010H
91                CNTL_PORT_2             EQU     4021H
92                STATUS_PORT             EQU     4030H
93                CHAN2_CA_PORT           EQU     4070H
94                ;
95                LD_CNTR0_54             EQU     4051H
96                LD_CNTR1_54             EQU     4053H
97                LD_CNTR2_54             EQU     4055H
98                MODE_54                 EQU     4057H
```

```
 99          RD_CNTR0_54                 EQU     4051H
100          RD_CNTR1_54                 EQU     4053H
101          RD_CNTR2_54                 EQU     4055H
102          ;
103          ;
104          ;          OFFSET VALUES FROM CNTL_PORT_1 = 4010H
105          ;
106          CNTL2                       EQU     011H
107          STATUS                      EQU     020H
108          CA2                         EQU     060H
109          ;
110          ;
111          ;          8254 CONTROL WORD BIT MASKS
112          ;
113          SEL_CNTR0_54                EQU     00000000B
114          SEL_CNTR1_54                EQU     01000000B
115          SEL_CNTR2_54                EQU     10000000B
116          ;
117          RD_LD_LATCH_54              EQU     00000000B
118          RD_LD_MSB_54                EQU     00100000B
119          RD_LD_LSB_54                EQU     00010000B
120          RD_LD_WORD_54               EQU     00110000B
121          ;
122          MODE0_54                    EQU     00000000B
123          MODE1_54                    EQU     00000010B
124          MODE2_54                    EQU     00000100B
125          MODE3_54                    EQU     00000110B
126          MODE4_54                    EQU     00001000B
127          MODE5_54                    EQU     00001010B
128          ;
129          BCD_COUNT_54                EQU     00000001B
130          ;
131 $EJECT
132          ;
133          ;
134          ;          CNTL_PORT_1 BIT MASKS
135          ;
136          CLEAR                       EQU     00000000B
137          FORMAT                      EQU     00000001B
138          READ                        EQU     00000010B
139          WRITE                       EQU     00000100B
140          CHAN1                       EQU     00001000B
141          CHAN2                       EQU     00000000B
142          ENB_XCVR                    EQU     00010000B
143          SEL_INDEX                   EQU     00100000B
144          ;
145          ;
146          ;          CNTL_PORT_2 BIT MASKS
147          ;
148          HEAD1                       EQU     00000001B
149          HEAD2                       EQU     00000010B
150          HEAD4                       EQU     00000100B
151          HEAD8                       EQU     00001000B
152          DRIVE1                      EQU     00010000B
153          INWARD                      EQU     00100000B
154          OUTWARD                     EQU     00000000B
```

```
155             STEP                    EQU     01000000B
156             FAULT_CLEAR             EQU     10000000B
157             ;
158             ;
159             ;       STATUS_PORT BIT POSITIONS
160             ;
161             COMPARE_STATUS          EQU     0
162             CRC_ERROR               EQU     1
163             SEEK_COMPLETE           EQU     4
164             DRIVE_READY             EQU     5
165             TRACK00                 EQU     6
166             WRITE_FAULT             EQU     7
167             ;
168             ;
169             ;       MASK-COMPARE (MC) PATTERNS
170             ;
171             TEST_SECTOR_FOUND       EQU     0301H
172             ;
173 $EJECT
174             ;
175             ;
176             ;       FUNCTION CODE BIT POSITIONS
177             ;
178             INIT_CODE               EQU     0
179             SEEK_CODE               EQU     1
180             FMAT_CODE               EQU     2
181             WRITE_CODE              EQU     3
182             READ_CODE               EQU     4
183             ;
184             LOOP_CODE               EQU     7
185             ;
186             ;
187             ;       ERROR CODE BIT POSITIONS
188             ;
189             BAD_CODE                EQU     0
190             NOT_READY               EQU     1
191             BAD_CYLINDER            EQU     2
192             BAD_HEAD                EQU     3
193             BAD_SECTOR              EQU     4
194             BAD_WRITE               EQU     5
195             BAD_SEARCH              EQU     6
196             BAD_ID_CRC              EQU     7
197             BAD_DATA_CRC            EQU     0
198             ;
199             ;
200             ;       OTHER CONSTANTS
201             ;
202             MAX_CYLINDER            EQU     202
203             MAX_HEAD                EQU     8
204             MAX_SECTOR              EQU     30
205             ;
206             ID_SYNCH                EQU     0FH
207             DATA_SYNCH              EQU     0DH
208             ;
209             ID_SIZE                 EQU     4
210             WORD_COUNT              EQU     256
```

```
211           BYTE_COUNT                EQU     WORD_COUNT + WORD_COUNT
212           ;
213           START_SYS_CCW             EQU     083H
214           START_LOC_CCW             EQU     081H
215           ;
216 $EJECT
217           ;-------------------------------------------------
218           ;
219           ;         DATA VARIABLE DEFINITIONS
220           ;
221           ;-------------------------------------------------
222           ;
223           ;
224           ORG       RAM_BASE
225           ;
226           CYLINDER:                 DW      0         ; IN LOW BYTE
227           HEAD:                     DW      0         ; IN LOW BYTE
228           SECTOR:                   DW      0         ; IN LOW BYTE
229           ;
230           FUNCTION:                 DW      0
231           ;
232           PRESENT_CYL:              DW      0         ; IN LOW BYTE
233           ;
234           FIND_SECTOR:              DW      0,0,0,0
235           ;
236           TEMP_STATUS:              DW      0
237           ;
238           TEMP:                     DW      0
239           ;
240           ;
241           ;
242           ORG       RAM_BASE + 05F0H
243           ;
244           SECTOR_BUFFER:            DS      512
245           ;
246 ;
247 $EJECT
248           ;-------------------------------------------------
249           ;
250           ;         C H A N N E L     1
251           ;
252           ;-------------------------------------------------
253           ;
254           ;
255           ;         CONTROL PROGRAM
256           ;
257           ORG       RAM_BASE + 040H
258           ;
259 TBLK1:    MOVI      GA,RAM_BASE               ; GA = RAM BASE PTR
260           MOVI      [GA].TEMP_STATUS,0H       ; STATUS = NO ERROR
261           MOVI      GC,CNTL_PORT_1            ; GC = I/O BASE PTR
262           MOV       [GA].FUNCTION,[PP].PB1_FUNCTION ; GET FUNCTION
263                                                     ;     CODE
264           LJBT      [GA].FUNCTION,INIT_CODE,INIT    ; JUMP IF INIT
265                                                     ;
266           LJBT      [GA].FUNCTION,SEEK_CODE,SEEK    ; JUMP IF SEEK
```

```
267                                                     ;
268             LJBT    [GA].FUNCTION,FMAT_CODE,FMAT    ; JUMP IF FMAT
269                                                     ;
270             LJBT    [GA].FUNCTION,WRITE_CODE,WDATA  ; JUMP IF WRITE
271                                                     ;
272             LJBT    [GA].FUNCTION,READ_CODE,RDATA   ; JUMP IF READ
273                                                     ;
274             SETB    [GA].TEMP_STATUS,BAD_CODE ; ERROR, INVALID
275             MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS ; FUNCTION
276             SINTR                                   ; SET INTERRUPT
277             HLT
278             ;
279 $EJECT
280             ;
281             ;
282             ;       INITIALIZATION
283             ;
284             ;-------------------------------------------------
285             ;
286 INIT:       MOVBI   [GC],CLEAR                      ; ZERO CONTROL PORTS
287             MOVBI   [GC].CNTL2,CLEAR
288             MOVBI   [GC].CNTL2,DRIVE1               ; SELECT DRIVE
289             ;
290 I10:        JNBT    [GC].STATUS,DRIVE_READY,I10 ; WAIT FOR DRIVE
291                                                 ;     READY
292             ;
293             ;
294             ;       RESET WRITE FAULT (IF ANY)
295             ;
296             JNBT    [GC].STATUS,WRITE_FAULT,I15
297             MOVBI   [GC].CNTL2,DRIVE1+FAULT_CLEAR
298             MOVBI   [GC].CNTL2,DRIVE1
299             ;
300             ;
301             ;       POSITION HEADS OVER TRACK00
302             ;
303 I15:        JBT     [GC].STATUS,TRACK00,I40
304 I20:        MOVBI   [GC].CNTL2,DRIVE1+OUTWARD+STEP
305             MOVBI   [GC].CNTL2,DRIVE1+OUTWARD
306 I30:        JNBT    [GC].STATUS,SEEK_COMPLETE,I30
307             JNBT    [GC].STATUS,TRACK00,I20
308             ;
309 I40:        MOVI    [GA].PRESENT_CYL,0H         ; INIT PRESENT_CYL
310             MOVI    [GA].CYLINDER,0H            ; ZERO VARIABLES
311             MOVI    [GA].HEAD,0H
312             MOVI    [GA].SECTOR,0H
313             ;
314             ;
315             ;       INITIALIZE 8254 CNTR0, CNTR1, AND CNTR2
316             ;
317             MOVI    GA,MODE_54
318             MOVBI   [GA],SEL_CNTR0_54 + RD_LD_WORD_54 + MODE5_54
319             MOVBI   [GA],SEL_CNTR1_54 + RD_LD_WORD_54 + MODE5_54
320             MOVBI   [GA],SEL_CNTR2_54 + RD_LD_WORD_54 + MODE5_54
321             MOVI    GA,LD_CNTR0_54
322             MOVBI   [GA],07                         ; CNTR0 COUNT = 8 PULSES
```

```
323             MOVBI   [GA],0
324             MOVI    GA,LD_CNTR1_54
325             MOVBI   [GA],03                 ; CNTR1 COUNT     PULSES
326             MOVBI   [GA],0
327             MOVI    GA,LD_CNTR2_54
328             MOVBI   [GA],02                 ; CNTR2 COUNT = 3 PULSES
329             MOVBI   [GA],0
330             ;
331             MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
332             MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS
333             SINTR                           ; SET INTERRUPT
334             HLT
335             ;
336 $EJECT
337             ;
338             ;
339             ;       SEEK TRACK
340             ;
341             ;-------------------------------------------------
342             ;
343             ;       CHECK IF DRIVE IS INITIALIZED
344             ;
345 SEEK:       JBT     [GC].STATUS,DRIVE_READY,S10 ; JMP IF DRIVE RDY
346             SETB    [GA].TEMP_STATUS,NOT_READY  ; SET ERROR BIT
347             LJMP    S80
348             ;
349             ;
350             ;       INITIALIZE VARIABLES: CYLINDER AND HEAD
351             ;
352 S10:        MOV     [GA].CYLINDER,[PP].PB1_CYLINDER
353             MOVB    GB,[PP].PB1_HEAD_SECTOR+1
354             MOV     [GA].HEAD,GB
355                                             ;
356                                             ; CHECK CYLINDER PARAM
357             MOVI    [GA].TEMP,MAX_CYLINDER-1 ; SUBTRACT FROM MAX
358             MOV     IX,[GA].CYLINDER         ;     VALUE
359             NOT     IX
360             INC     IX
361             ADD     [GA].TEMP,IX
362             JNBT    [GA].TEMP+1,7,S13       ; JUMP IF POSITIVE
363             SETB    [GA].TEMP_STATUS,BAD_CYLINDER ; SET ERROR BIT
364                                             ;
365                                             ; CHECK HEAD PARAM
366 S13:        MOVI    [GA].TEMP,MAX_HEAD-1    ; SUBTRACT FROM MAX
367             MOV     IX,[GA].HEAD            ;     VALUE
368             NOT     IX
369             INC     IX
370             ADD     [GA].TEMP,IX
371             JNBT    [GA].TEMP+1,7,S16       ; JUMP IF POSITIVE
372             SETB    [GA].TEMP_STATUS,BAD_HEAD ; SET ERROR BIT
373 S16:        JNZ     [GA].TEMP_STATUS,S80    ; JUMP IF ERROR
374             ;
375             ;
376             ;       DETERMINE HEAD MOVEMENT: INWARD, OUTWARD,
377             ;       OR NONE
378             ;
```

```
379             MOV     [GA].TEMP,[GA].CYLINDER ; SUBTRACT PRESENT CYL
380             MOV     IX,[GA].PRESENT_CYL     ;     FROM NEW CYLINDER
381             NOT     IX
382             INC     IX
383             ADD     [GA].TEMP,IX
384             JZ      [GA].TEMP,S60           ; JUMP IF DELTA ZERO
385             JBT     [GA].TEMP+1,7,S30       ; JUMP IF DELTA NEGATIVE
386             ;
387 $EJECT
388             ;
389             ;
390             ;       MOVE HEADS INWARD (POSITIVE DELTA)
391             ;
392             MOV     BC,[GA].TEMP            ; GET CYLINDER COUNT
393 S20:        MOVBI   [GC].CNTL2,DRIVE1+INWARD+STEP ; PULSE
394             MOVBI   [GC].CNTL2,DRIVE1+INWARD
395             DEC     BC                      ; DECREMENT COUNT AND
396             JNZ     BC,S20                  ;     REPEAT IF <> 0
397             JMP     S50
398             ;
399             ;
400             ;       MOVE HEADS OUTWARD (NEGATIVE DELTA)
401             ;
402 S30:        MOV     BC,[GA].TEMP            ; GET AND COMPLEMENT
403             NOT     BC                      ;     CYLINDER COUNT
404             INC     BC
405 S40:        MOVBI   [GC].CNTL2,DRIVE1+OUTWARD+STEP ; PULSE
406             MOVBI   [GC].CNTL2,DRIVE1+OUTWARD
407             DEC     BC                      ; DECREMENT COUNT AND
408             JNZ     BC,S40                  ;     REPEAT IF <> 0
409             ;
410 S50:        JNBT    [GC].STATUS,SEEK_COMPLETE,S50 ; WAIT FOR SEEK
411                                             ;     COMPLETE SIG
412             ;
413             ;
414             ;       UPDATE PRESENT_CYL VARIABLE
415             ;
416             MOV     [GA].PRESENT_CYL,[GA].CYLINDER
417             ;
418             ;
419             ;       SELECT HEAD:  ACTIVATE HEAD SIGNALS TO DRIVE
420             ;
421 S60:        MOV     IX,[GA].HEAD
422             ORI     IX,DRIVE1
423             MOVB    [GC].CNTL2,IX
424             ;
425             ;
426             ;       20 MSEC TIME DELAY
427             ;
428             MOVI    IX,3448
429 S70:        DEC     IX
430             JNZ     IX,S70
431             ;
432             ;
433 S80:        MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS
434             SINTR                           ; SET INTERRUPT
```

```
435             HLT
436             ;
437 $EJECT
438             ;
439             ;
440             ;       FORMAT TRACK
441             ;
442             ;-----------------------------------------------
443             ;
444             ;       CHECK IF DRIVE IS INITIALIZED
445             ;
446 FMAT:       JBT     [GC].STATUS,DRIVE_READY,F05 ; JMP IF DRIVE RDY
447             SETB    [GA].TEMP_STATUS,NOT_READY  ; SET ERROR BIT
448             LJMP    F50
449             ;
450             ;
451             ;       INITIALIZE 8254 FOR FORMAT
452
453 F05:        MOVI    GA,LD_CNTR0_54
454             MOVBI   [GA],07                     ; CNTR0 COUNT = 8
455             MOVBI   [GA],0
456             MOVI    GA,LD_CNTR1_54
457             MOVBI   [GA],03                     ; CNTR1 COUNT = 4
458             MOVBI   [GA],0
459             ;
460             ;
461             ;       GENERATE BYTE ARRAY, SECTOR(30), WHICH CONTAINS
462             ;       THE INTERLEAVED SECTOR NUMBERS STARTING AT
463             ;       ADDRESS = SECTOR_BUFFER + 100H
464             ;
465             MOVI    GA,RAM_BASE                 ; GA = RAM BASE PTR
466             MOVI    GB,SECTOR_BUFFER + 100H
467             MOVI    [GA].TEMP,0H                ; J = 0
468 F10:        MOV     BC,[GA].TEMP                ; SECTOR(I) = J
469             MOV     [GB].0H,BC
470             ADDI    BC,10                       ; SECTOR(I+1) = J+10
471             MOV     [GB].1H,BC
472             ADDI    BC,10                       ; SECTOR(I+2) = J+20
473             MOV     [GB].2H,BC
474             ADDI    GB,3                        ; I = I+3
475             INC     [GA].TEMP                   ; J = J+1
476             MOV     BC,[GA].TEMP                ; REPEAT IF J <> 10
477             NOT     BC
478             INC     BC
479             ADDI    BC,10
480             JNZ     BC,F10
481             ;
482             ;
483             ;       LOAD MC REGISTER WITH HEAD DATA IN UPPER
484             ;       BYTE (BITS 8-15)
485             ;
486             MOVI    BC,8H                       ; SHIFT COUNT = 8
487             MOV     [GA].TEMP,[GA].HEAD         ; GET HEAD DATA
488 F15:        MOV     MC,[GA].TEMP                ; SHIFT LEFT BY ADDING
489             ADD     [GA].TEMP,MC                ;    TO ITSELF
490             DEC     BC                          ; DECREMENT SHIFT COUNT
```

```
491             JNZ     BC,F15                  ;     & REPEAT IF <> 0
492             MOV     MC,[GA].TEMP
493             ;
494 $EJECT
495             ;
496             ;
497             ;       GENERATE SECTOR FORMAT TABLE STARTING
498             ;       AT ADDRESS = SECTOR_BUFFER
499             ;
500             MOVI    GB,SECTOR_BUFFER + 100H
501             MOVI    [GA].TEMP,0H            ; SECTOR COUNT = 0
502             MOVI    IX,SECTOR_BUFFER
503 F20:       MOVI    [GA+IX+],ID_SYNCH       ; SYNCH CHARACTER
504             MOV     [GA+IX+],[GA].CYLINDER  ; CYLINDER
505             MOV     [GA+IX],MC              ; HEAD / SECTOR
506             MOVB    BC,[GB]
507             OR      [GA+IX+],BC
508             MOVI    [GA+IX+],0H             ; ZEROS
509             INC     GB                      ; INCREMENT SECTOR NO.
510                                             ;    POINTER
511             INC     [GA].TEMP               ; INCREMENT SECTOR COUNT
512             MOV     BC,[GA].TEMP            ;    & REPEAT IF <> MAX
513             NOT     BC
514             INC     BC
515             ADDI    BC,MAX_SECTOR
516             JNZ     BC,F20
517             ;
518             ;
519             ;       FORMAT FIRST SECTOR AFTER INDEX PULSE
520             ;
521 F30:       MOVI    GB,SECTOR_BUFFER        ; SOURCE POINTER
522             MOVI    GA,DATA_PORT            ; DESTINATION POINTER
523             MOVI    BC,ID_SIZE + 6          ; BYTE COUNT
524             MOVI    CC,BLOCK_TO_PORT
525 &                           + DEST_SYNCH
526 &                           + GB_SOURCE
527 &                           + EXT_TERM_0
528 &                           + BC_TERM_0     ; DMA CONTROL
529             WID     16,16                   ; 16-BIT TO 16-BIT DMA
530             XFER                            ; INIT DUMMY DMA TO
531                                             ;    DETECT INDEX PULSE
532             MOVBI   [GC],FORMAT+SEL_INDEX   ; EXT1 = INDEX PULSE
533                                             ; --------------------
534                                             ; WAIT FOR INDEX PULSE
535                                             ; --------------------
536             XFER                            ; START ID DATA TO DRIVE
537                                             ;    DMA
538             MOVBI   [GC],FORMAT
539 &                           + WRITE
540 &                           + CHAN1
541 &                           + ENB_XCVR      ; OUTPUT FORMAT COMMAND
542                                             ; --------------------
543                                             ; DMA OCCURS HERE
544                                             ; --------------------
545             MOVBI   [GC],FORMAT             ; RESET ALL BUT FORMAT
546                                             ;    LINE
```

```
547             ;
548 $EJECT
549             ;
550             MOVI    GA,RAM_BASE                 ; GA = RAM BASE PTR
551             JNBT    [GC].STATUS,WRITE_FAULT,F35 ; JUMP IF NO FAULT
552             SETB    [GA].TEMP_STATUS,BAD_WRITE  ; SET ERROR BIT
553             JMP     F50
554             ;
555             ;
556             ;       FORMAT REMAINING SECTORS
557             ;
558 F35:        MOVI    MC,MAX_SECTOR-1             ; SECTOR_COUNT = MAX-1
559 F40:        MOVI    GA,DATA_PORT                ; DESTINATION POINTER
560             MOVI    BC,ID_SIZE + 6              ; BYTE COUNT
561             MOVI    CC,BLOCK_TO_PORT
562 &                           + DEST_SYNCH
563 &                           + GB_SOURCE
564 &                           + EXT_TERM_0
565 &                           + BC_TERM_0         ; DMA CONTROL
566             XFER                                ; INIT DUMMY DMA TO
567                                                 ;    DETECT SECTOR PULSE
568             WID     16,16                       ; 16-BIT TO 16-BIT DMA
569                                                 ; --------------------
570                                                 ; WAIT FOR SECTOR PULSE
571                                                 ; --------------------
572             XFER                                ; START ID DATA TO DRIVE
573                                                 ;    DMA
574             MOVBI   [GC],FORMAT
575 &                           + WRITE
576 &                           + CHAN1
577 &                           + ENB_XCVR          ; OUTPUT FORMAT COMMAND
578                                                 ; --------------------
579                                                 ; DMA OCCURS HERE
580                                                 ; --------------------
581             MOVBI   [GC],FORMAT                 ; RESET ALL BUT FORMAT
582                                                 ;    LINE
583             MOVI    GA,RAM_BASE                 ; GA = RAM BASE PTR
584             JNBT    [GC].STATUS,WRITE_FAULT,F45 ; JUMP IF NO FAULT
585             SETB    [GA].TEMP_STATUS,BAD_WRITE  ; SET ERROR BIT
586             JMP     F50
587             ;
588             ;
589             ;       DECREMENT SECTOR_COUNT AND JUMP IF <> 0
590             ;
591 F45:        DEC     MC
592             JNZ     MC,F40
593             ;
594 F50:        MOVI    IX,26                       ; 150 MSEC DELAY
595 F55:        DEC     IX                          ;    (FOR WRITE GATE
596             JNZ     IX,F55                      ;     TURN OFF)
597             MOVBI   [GC],CLEAR                  ; CLEAR FORMAT LINE
598             ;
599             MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS
600             SINTR                               ; SET INTERRUPT
601             HLT
602             ;
```

```
603 $EJECT
604             ;
605             ;
606             ;           WRITE SECTOR DATA
607             ;
608             ;-----------------------------------------------
609             ;
610             ;           CHECK IF DRIVE IS INITIALIZED
611             ;
612 WDATA:     JBT         [GC].STATUS,DRIVE_READY,W05 ; JMP IF DRIVE RDY
613             SETB        [GA].TEMP_STATUS,NOT_READY  ; SET ERROR BIT
614             LJMP        W50
615             ;
616             ;
617             ;           INITIALIZE SECTOR VARIABLES
618             ;
619 W05:       MOV         [GA].FIND_SECTOR,[PP].PB1_CYLINDER ; FIND_SECTOR
620             MOV         GB,[PP].PB1_HEAD_SECTOR ; FIND_SECTOR + 2
621             MOV         [GA].FIND_SECTOR+2,GB
622             ANDI        GB,0FFH                 ; SECTOR
623             MOV         [GA].SECTOR,GB
624                                                 ; CHECK SECTOR PARAM
625             MOVI        [GA].TEMP,MAX_SECTOR-1  ; SUBTRACT FROM MAX
626             MOV         IX,[GA].SECTOR          ;    VALUE
627             NOT         IX
628             INC         IX
629             ADD         [GA].TEMP,IX
630             JNBT        [GA].TEMP+1,7,W10       ; JUMP IF POSITIVE
631             SETB        [GA].TEMP_STATUS,BAD_SECTOR ; SET ERROR BIT
632             LJMP        W50
633             ;
634             ;
635             ;           INITIALIZE 8254 FOR WRITE DATA
636             ;
637 W10:       MOVI        GA,LD_CNTR0_54
638             MOVBI       [GA],20                 ; CNTR0 COUNT = 21
639             MOVBI       [GA],0
640             MOVI        GA,LD_CNTR1_54
641             MOVBI       [GA],1                  ; CNTR1 COUNT = 258
642             MOVBI       [GA],1
643             ;
644             ;
645             ;           TRANSFER DATA FROM SYSTEM BUFFER TO
646             ;           LOCAL BUFFER
647             ;
648             LPD         GA,[PP].PB1_BUFR_OFF    ; SOURCE POINTER
649             MOVI        GB,SECTOR_BUFFER        ; DESTINATION POINTER
650             MOVI        [GB],DATA_SYNCH         ; INSERT SYNCH CHAR
651             ADDI        GB,2                    ;    IN LOCAL BUFFER
652             MOVI        BC,BYTE_COUNT           ; BYTE COUNT
653             MOVI        CC,BLOCK_TO_BLOCK
654 &                               + GA_SOURCE
655 &                               + BC_TERM_0     ; DMA CONTROL
656             XFER                                ; INIT DMA
657             WID         16,16                   ; 16-BIT TO 16-BIT DMA
658             MOVI        [GB],0H                 ; INSERT ZEROS
```

```
659             ;
660 $EJECT
661             ;
662             ;
663             ;       PREPARE CHANNEL 2'S CCB AND PB
664             ;
665                                             ; INITIALIZE CCB
666             LPD     GA,[PP].PB1_CCB_OFF     ; GET CCB ADDRESS
667             MOVBI   [GA].08H,START_LOC_CCW  ; INIT CCW
668             MOV     [GA].0AH,[PP].PB1_PB2_OFF ; INIT PB2 OFFSET
669             MOV     [GA].0CH,[PP].PB1_PB2_SEG ; INIT PB2 SEGMENT
670                                             ;
671                                             ; INITIALIZE PB2
672             LPD     GA,[PP].PB1_PB2_OFF     ; GET PB2 ADDRESS
673             MOVI    [GA].PB2_TB2_OFF,TBLK2  ; INIT TB2 ADDRESS
674             MOVI    [GA].PB2_FUNCTION,CMP_ID ; INIT COMPARE CMD
675             MOVI    [GA].PB2_BUFR_OFF,FIND_SECTOR ; INIT BUFFER
676                                                   ;     ADDR
677             ;
678             ;
679             ;       SEARCH FOR SECTOR SPECIFIED IN FIND_SECTOR
680             ;
681 W20:        MOVI    IX,MAX_SECTOR           ; SECTOR_COUNT = MAX
682             ;
683 W30:        MOVI    GA,SECTOR_BUFFER        ; SOURCE POINTER
684             MOVI    GB,DATA_PORT            ; DESTINATION POINTER
685             MOVI    BC,BYTE_COUNT + 6       ; BYTE COUNT
686             MOVI    CC,BLOCK_TO_PORT
687 &                           + DEST_SYNCH
688 &                           + GA_SOURCE
689 &                           + EXT_TERM_0
690 &                           + BC_TERM_0     ; DMA CONTROL
691             WID     16,16                   ; 16-BIT TO 16-BIT DMA
692             MOVI    MC,TEST_SECTOR_FOUND    ; INIT MC
693             XFER                            ; INIT DUMMY DMA TO
694                                             ;    DETECT END OF ID
695                                             ;    COMPARE
696             MOVB    [GC].CA2,BC             ; GENERATE CHANNEL 2
697                                             ;    CA SIGNAL
698                                             ; --------------------
699                                             ; WAIT FOR CHANNEL 2
700                                             ; TO COMPARE ID
701                                             ; --------------------
702             JMCNE   [GC].STATUS,W40         ; JUMP IF NOT FOUND
703             ;
704 $EJECT
705             ;
706             ;
707             ;       WRITE SECTOR DATA ON DISK
708             ;
709             MOVBI   [GC],CLEAR              ; CLEAR READ LINE
710             XFER                            ; START DMA WRITE
711             MOVBI   [GC],WRITE + CHAN1
712 &                           + ENB_XCVR      ; OUTPUT WRITE COMMAND
713                                             ; --------------------
714                                             ; DMA OCCURS HERE
```

```
715                                             ; ---------------------
716             NOP                             ; TIME DELAY
717             NOP
718             NOP
719             NOP
720             NOP
721             MOVBI   [GC],CLEAR              ; CLEAR WRITE LINE
722                                             ;
723             MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
724             JNBT    [GC].STATUS,WRITE_FAULT,W50 ; JUMP IF NO FAULT
725             SETB    [GA].TEMP_STATUS,BAD_WRITE  ; SET ERROR BIT
726             JMP     W50
727             ;
728             ;
729             ;       NO MATCH ON PRESENT SECTOR
730             ;
731 W40:        MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
732             JNBT    [GC].STATUS,CRC_ERROR,W45 ; JUMP IF NO ERROR
733             SETB    [GA].TEMP_STATUS,BAD_ID_CRC ; SET ERROR BIT
734 W45:        MOVBI   [GC],ENB_XCVR           ; RESET COMPARE STATUS
735             MOVBI   [GC],CLEAR              ;     FLIP FLOP
736             JNZ     [GA].TEMP_STATUS,W50    ; JUMP IF ERROR
737                                             ;
738             DEC     IX                      ; DEC SECTOR_COUNT &
739             JNZ     IX,W30                  ;     LOOP IF <> 0
740             SETB    [GA].TEMP_STATUS,BAD_SEARCH ; SET ERROR BIT
741             ;
742 W50:        MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS
743             LJBT    [GA].FUNCTION+1,LOOP_CODE,W20
744             SINTR                           ; SET INTERRUPT
745             HLT
746             ;
747 $EJECT
748             ;
749             ;
750             ;       READ SECTOR DATA
751             ;
752             ;-------------------------------------------------
753             ;
754             ;       CHECK IF DRIVE IS INITIALIZED
755             ;
756 RDATA:      JBT     [GC].STATUS,DRIVE_READY,R05 ; JMP IF DRIVE RDY
757             SETB    [GA].TEMP_STATUS,NOT_READY  ; SET ERROR BIT
758             LJMP    R50
759             ;
760             ;
761             ;       INITIALIZE SECTOR VARIABLES
762             ;
763 R05:        MOV     [GA].FIND_SECTOR,[PP].PB1_CYLINDER ; FIND_SECTOR
764             MOV     GB,[PP].PB1_HEAD_SECTOR ; FIND_SECTOR + 2
765             MOV     [GA].FIND_SECTOR+2,GB
766             ANDI    GB,0FFH                 ; SECTOR
767             MOV     [GA].SECTOR,GB
768                                             ; CHECK SECTOR PARAM
769             MOVI    [GA].TEMP,MAX_SECTOR-1  ; SUBTRACT FROM MAX
770             MOV     IX,[GA].SECTOR          ;     VALUE
```

```
771             NOT     IX
772             INC     IX
773             ADD     [GA].TEMP,IX
774             JNBT    [GA].TEMP+1,7,R07       ; JUMP IF POSITIVE
775             SETB    [GA].TEMP_STATUS,BAD_SECTOR ; SET ERROR BIT
776             LJMP    R50
777             ;
778             ;
779             ;       INITIALIZE 8254 FOR READ DATA
780             ;
781 R07:       MOVI    GA,LD_CNTR1_54
782             MOVBI   [GA],0                  ; CNTR1 COUNT = 257
783             MOVBI   [GA],1
784             ;
785             ;
786             ;       ZERO SECTOR BUFFER
787             ;
788             MOVI    GA,SECTOR_BUFFER
789             MOVI    IX,0
790             MOVI    BC,WORD_COUNT
791 R10:       MOVI    [GA+IX+],0
792             DEC     BC
793             JNZ     BC,R10
794             ;
795 $EJECT
796             ;
797             ;
798             ;       PREPARE CHANNEL 2'S CCB AND PB
799             ;
800                                             ; INITIALIZE CCB
801             LPD     GA,[PP].PB1_CCB_OFF     ; GET CCB ADDRESS
802             MOVBI   [GA].08H,START_LOC_CCW  ; INIT CCW
803             MOV     [GA].0AH,[PP].PB1_PB2_OFF ; INIT PB2 OFFSET
804             MOV     [GA].0CH,[PP].PB1_PB2_SEG ; INIT PB2 SEGMENT
805                                             ;
806                                             ; INITIALIZE PB2
807             LPD     GA,[PP].PB1_PB2_OFF     ; GET PB2 ADDRESS
808             MOVI    [GA].PB2_TB2_OFF,TBLK2  ; INIT TB2 ADDRESS
809             MOVI    [GA].PB2_FUNCTION,CMP_ID ; INIT COMPARE CMD
810             MOVI    [GA].PB2_BUFR_OFF,FIND_SECTOR ; INIT BUFFER
811                                                   ;      ADDR
812             ;
813             ;
814             ;       SEARCH FOR SECTOR SPECIFIED IN FIND_SECTOR
815             ;
816 R20:       MOVI    IX,MAX_SECTOR           ; SECTOR_COUNT = MAX
817             ;
818 R30:       MOVI    GA,SECTOR_BUFFER        ; SOURCE POINTER
819             MOVI    GB,DATA_PORT            ; DESTINATION POINTER
820             MOVI    BC,BYTE_COUNT           ; BYTE COUNT
821             MOVI    CC,PORT_TO_BLOCK
822 &                       + SOURCE_SYNCH
823 &                       + GB_SOURCE
824 &                       + EXT_TERM_0
825 &                       + BC_TERM_0         ; DMA CONTROL
826             WID     16,16                   ; 16-BIT TO 16-BIT DMA
```

```
827           MOVI    MC,TEST_SECTOR_FOUND    ; INIT MC
828           XFER                            ; INIT DUMMY DMA TO
829                                           ;    DETECT END OF ID
830                                           ;    COMPARE
831           MOVB    [GC].CA2,BC             ; GENERATE CHANNEL 2
832                                           ;    CA SIGNAL
833                                           ; --------------------
834                                           ; WAIT FOR CHANNEL 2
835                                           ; TO COMPARE ID
836                                           ; --------------------
837           JMCNE   [GC].STATUS,R40         ; JUMP IF NOT FOUND
838           ;
839 $EJECT
840           ;
841           ;
842           ;       READ SECTOR DATA FROM DISK
843           ;
844           MOVBI   [GC],CLEAR              ; CLEAR READ LINE
845           NOP                             ; TIME DELAY
846           NOP
847           NOP
848           XFER                            ; START DMA READ
849           MOVBI   [GC],READ + CHAN1
850 &                       + ENB_XCVR        ; OUTPUT READ COMMAND
851                                           ; --------------------
852                                           ; DMA OCCURS HERE
853                                           ; --------------------
854           MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
855           JNBT    [GC].STATUS,CRC_ERROR,R35 ; JUMP IF NO ERROR
856           MOVBI   [GC],CLEAR              ; CLEAR READ LINE
857           SETB    [GA].TEMP_STATUS+1,BAD_DATA_CRC ; SET ERROR BIT
858           JMP     R50
859           ;
860           ;
861           ;       TRANSFER DATA FROM LOCAL BUFFER TO
862           ;       SYSTEM BUFFER
863           ;
864 R35:     MOVBI   [GC],CLEAR              ; CLEAR READ LINE
865           MOVI    GA,SECTOR_BUFFER        ; SOURCE POINTER
866           LPD     GB,[PP].PB1_BUFR_OFF    ; DESTINATION POINTER
867           MOVI    BC,BYTE_COUNT           ; BYTE COUNT
868           MOVI    CC,BLOCK_TO_BLOCK
869 &                       + GA_SOURCE
870 &                       + BC_TERM_0       ; DMA CONTROL
871           XFER                            ; INIT DMA
872           WID     16,16                   ; 16-BIT TO 16-BIT DMA
873           MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
874           JMP     R50
875           ;
876           ;
877           ;       NO MATCH ON PRESENT SECTOR
878           ;
879 R40:     MOVI    GA,RAM_BASE             ; GA = RAM BASE PTR
880           JNBT    [GC].STATUS,CRC_ERROR,R45 ; JUMP IF NO ERROR
881           SETB    [GA].TEMP_STATUS,BAD_ID_CRC ; SET ERROR BIT
882 R45:     MOVBI   [GC],ENB_XCVR           ; RESET COMPARE STATUS
```

```
883             MOVBI   [GC],CLEAR                  ;     FLIP FLOP
884             JNZ     [GA].TEMP_STATUS,R50        ; JUMP IF ERROR
885                                                 ;
886             DEC     IX                          ; DEC SECTOR_COUNT &
887             JNZ     IX,R30                      ;     LOOP IF <> 0
888             SETB    [GA].TEMP_STATUS,BAD_SEARCH ; SET ERROR BIT
889             ;
890 R50:        MOV     [PP].PB1_STATUS,[GA].TEMP_STATUS
891             LJBT    [GA].FUNCTION+1,LOOP_CODE,R20
892             SINTR                               ; SET INTERRUPT
893             HLT
894             ;
895 $EJECT
896 $INCLUDE(:F1:CHAN2.A89)
897             ;-----------------------------------------------
898             ;
899             ;       C H A N N E L     2
900             ;
901             ;-----------------------------------------------
902             ;
903             ;
904             ;       DETERMINE OPERATION TO BE PERFORMED
905             ;
906             ;               0 = COMPARE ID FIELD
907             ;
908             ;               1 = READ ID FIELD
909             ;
910             ORG     RAM_BASE + 0580H
911             ;
912 TBLK2:     MOV     IX,[PP].PB2_FUNCTION        ; GET OPERATION CODE
913             JNZ     IX,RD_ID
914             ;
915             ;
916             ;       COMPARE ID FIELD OPERATION
917             ;
918 CP_ID:      MOV     GA,[PP].PB2_BUFR_OFF        ; SOURCE POINTER
919             MOVI    GB,DATA_PORT                ; DESTINATION POINTER
920             MOVI    BC,ID_SIZE + 2              ; BYTE COUNT
921             MOVI    CC,BLOCK_TO_PORT
922 &                           + DEST_SYNCH
923 &                           + GA_SOURCE
924 &                           + EXT_TERM_0
925 &                           + BC_TERM_0
926 &                           + CHAINED_MODE      ; DMA CONTROL
927             MOVI    GC,CNTL_PORT_1              ; CONTROL PORT POINTER
928             XFER                                ; INIT DUMMY DMA TO
929                                                 ;    DETECT SECTOR
930                                                 ;    PULSE
931             WID     16,16                       ; 16-BIT TO 16-BIT DMA
932                                                 ; --------------------
933                                                 ; WAIT FOR SECTOR PULSE
934                                                 ; --------------------
935             XFER                                ; START COMPARE ID FIELD
936                                                 ;    DMA
937             MOVBI   [GC],READ + CHAN2           ; OUTPUT COMMAND
938                                                 ; --------------------
```

```
                                                    ; DMA OCCURS HERE
                                                    ; --------------------
                HLT
                ;
                ;
$EJECT
                ;
                ;
                ;       READ ID FIELD OPERATION
                ;
RD_ID:          MOVI    GA,DATA_PORT                ; SOURCE POINTER
                MOV     GB,[PP].PB2_BUFR_OFF        ; DESTINATION POINTER
                MOVI    BC,ID_SIZE                  ; BYTE COUNT
                MOVI    CC,PORT_TO_BLOCK
&                               + SOURCE_SYNCH
&                               + GA_SOURCE
&                               + EXT_TERM_0
&                               + BC_TERM_0
&                               + CHAINED_MODE      ; DMA CONTROL
                MOVI    GC,CNTL_PORT_1              ; CONTROL PORT POINTER
                XFER                                ; INIT DUMMY DMA TO
                                                    ;    DETECT SECTOR
                                                    ;    PULSE
                WID     16,16                       ; 16-BIT TO 16-BIT DMA
                                                    ; --------------------
                                                    ; WAIT FOR SECTOR PULSE
                                                    ; --------------------
                XFER                                ; START READ ID FIELD DMA
                MOVBI   [GC],READ + CHAN2
&                               + ENB_XCVR          ; OUTPUT COMMAND
                                                    ; --------------------
                                                    ; DMA OCCURS HERE
                                                    ; --------------------
                HLT
                ;
                ;
HDC89           ENDS
                ;
                ;
                ;
                END
```

intel

APPLICATION NOTE

AP-123

March 1982

Graphic CRT Design Using the iAPX 86/11

Hal Kop
Microcomputer Applications

MARCH 1982
ORDER NUMBER: 210355-001

INTRODUCTION

The purpose of this application note is to provide the reader with the conceptual tools and factual information needed to apply the iAPX 86/11 to graphic CRT design. Particular attention will be paid to the requirements of high-resolution, color graphic applications, since these tend to require higher performance than those which do not use color.

The iAPX 86/11 is a microprocessor system which contains an 8086 CPU and an 8089 Input/Output Processor. In the graphic CRT application, the 8089 performs DMA transfers from the display memory to the CRT controller, and also serves as a CPU for functions such as keyboard polling and initialization of the CRT controller chips. The DMA transfers are done in such a manner that they do not tie up the system bus.

The system is organized so that the 8086 and the 8089 can perform concurrent processing on separate buses. Using the inherent ability of the 8089 to execute programs in its own I/O space, the 8086 can successfully delegate many of the chores that have specifically to do with the CRT display and keyboard, thus reducing the 8086's processing overhead. For these reasons, the capabilities of the 8086 as a CPU can be more fully utilized to perform calculations dealing with the material to be displayed. Thus, more complex types of displays can be undertaken, and the terminal will also be more interactive.

This application note is presented in five sections:

1. Introduction
2. Overview of Graphic CRT Systems
3. Overview of the 8089
4. Graphic CRT System Design
5. Conclusions

Section 2 discusses typical CRT designs, shows how performance requirements increase when the capability for color graphics is included, and explains some of the system bottlenecks that can arise. Section 3 describes the capabilities of the 8089, which can be brought to bear to resolve these bottlenecks. Section 4 gives detailed information for a color graphic CRT system using the iAPX 86/11 (8086 and 8089).

The reader may obtain useful background information on the 8086 and 8089 from *iAPX 86,88 User's Manual*. It would also be helpful to read the data sheets on the 8086, 8089, 2118 Dynamic RAM, 8202 Dynamic Ram Controller, 8275 CRT Controller, 8279 Keyboard/Display Interface, and 2732A EPROM.

OVERVIEW OF CRT GRAPHIC SYSTEMS

Typical Design Technique

A typical microprocessor-based CRT terminal is shown in block diagram form in Figure 1. The terminal consists

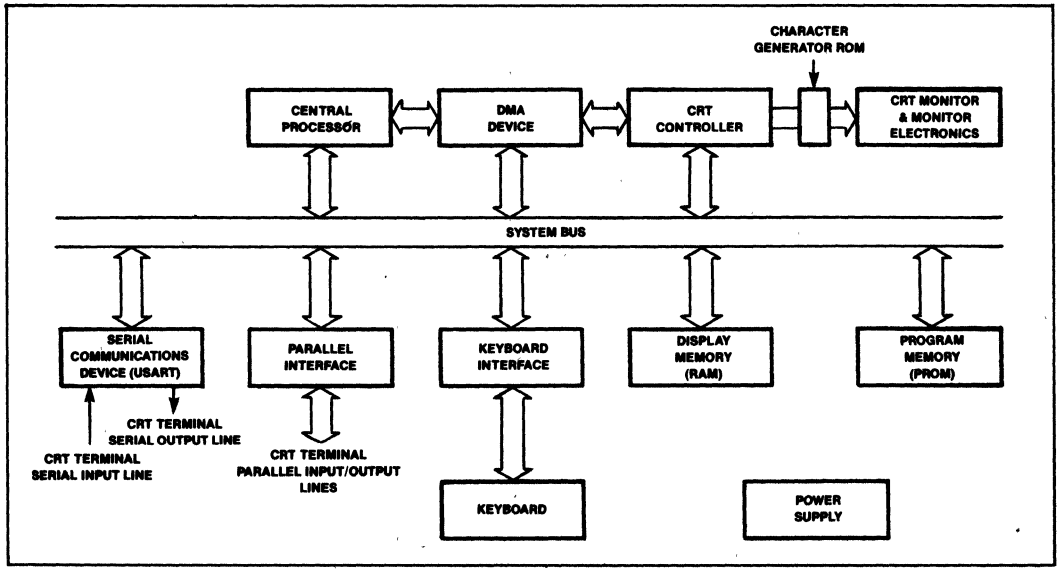


Figure 1. Typical CRT Terminal Block Diagram

of a CRT monitor, monitor electronics, a CRT controller and character generator ROM, display memory, a DMA device, a central processor and associated program memory, a keyboard and keyboard interface, and serial and/or parallel communication devices.

The primary function of the non-graphic CRT controller is to refresh the display. It does this by controlling the periodic transfer of information from display memory to the CRT screen, with the help of the DMA device. The central processing unit (CPU) coordinates the transfer of information to and from the external devices. When information from an external device is received by the terminal, the CPU performs character recognition and handling functions, display memory management functions, and cursor control functions. The CPU also interrogates the keyboard interface device. If a key depression is detected, the ASCII character representing that key is sent to the display memory and/or an external device.

The design shown in Figure 1 could be implemented using Intel LSI products. The CPU could be an 8085, the DMA device an 8237A DMA controller, the CRT controller an 8275, the character generator ROM a 2708, program memory ROM a 2716, display memory 2114s (2K x 8), and the keyboard interface an 8279 keyboard controller. These choices would result in a CRT terminal capable of displaying 25 lines of text containing 80 characters each.

As the design is upgraded to add color and graphics capability, performance requirements increase accordingly. The components most likely to require changing are the CPU, the DMA device, the CRT controller, and the display memory. Thus, it is desirable at this point to examine the operation of these components in more detail to provide a foundation for graphic system operation. Later we shall give a specific example of a more complex display, and examine the performance requirements imposed. Figure 2 is a block diagram showing only those components involved with the non-graphic CRT refresh function, with more detail provided regarding the connecting signal lines.

The refresh function proceeds as follows. The 8275, having been programmed to the specific screen format, generates a series of DMA request signals to the 8237A. This results in the transfer of a row of characters from display memory to one of two row buffers within the 8275. From this row buffer, the characters are sent, via lines CC0–CC6, to the character generator ROM. The dot timing and interface circuitry is then utilized to convert the parallel output data from the character generator ROM into serial signals for the video input of the CRT.

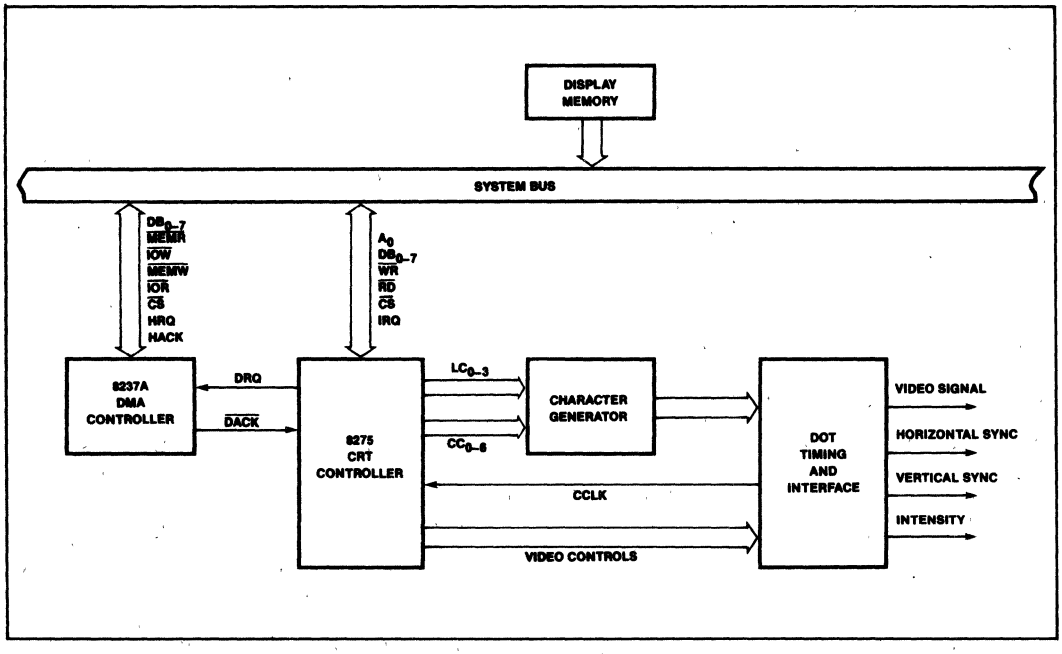


Figure 2. Components Involved in the CRT Refresh Function

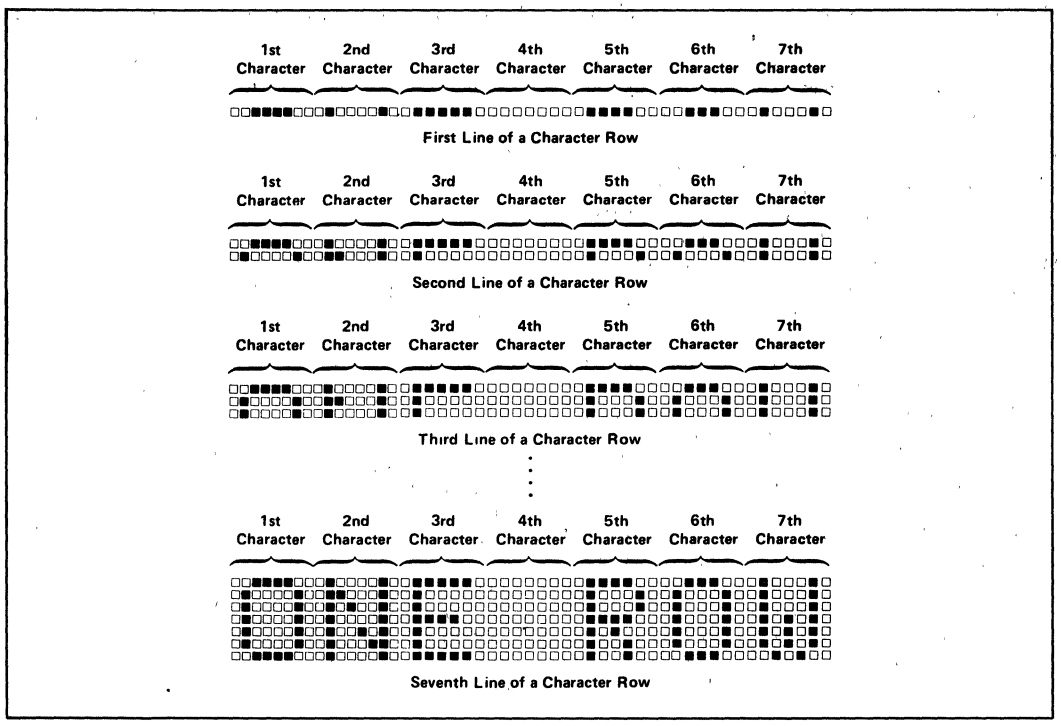


Figure 3. Character Row Display

The character rows are displayed on the CRT one line at a time. Line count signals LC0–LC3 are applied to the character generator ROM by the 8275, to specify the specific line count within the row of characters. This display process is shown in Figure 3, using a seven-line character for purposes of illustration. The entire process is repeated for each row of characters in the display.

At the beginning of the last display row, the 8275 issues an interrupt request via the IRQ output line. This interrupt output is normally connected to the interrupt input of the system CPU. The interrupt causes the CPU to execute an interrupt service subroutine. This subroutine typically reinitializes the DMA controller parameters for the next display refresh cycle, polls the system keyboard controller, and executes other appropriate functions.

Performance Requirements

In the example we have discussed thus far, a display consisting of 25 rows, each containing 80 text characters, with no color or graphic capability, has been assumed. Such a screen can be represented by 80 x 25 = 2000 bytes of data. If the screen is refreshed 60 times per second, then a total of 120,000 bytes will need to be transferred each second from display memory to the 8275 CRT controller. This figure is well within the capability of the 8237A DMA controller, even allowing for vertical retrace time and other overhead. In this application then, both the display memory and the system bus remain available to the system CPU most of the time, and no bottleneck occurs because of the DMA transfer process.

The situation is quite different when a high-resolution, color graphics capability is desired. The performance requirements are obviously much greater. To derive a quantitative requirement it is necessary to choose, even if somewhat arbitrarily, a specific display method and screen format. The display method chosen for the system described in this application note is called the virtual-bit mapping technique. When this technique is used, the graphic material to be displayed is handled on a character basis. Figure 4 shows the structure of the text and graphic characters used. The text character is a

7 x 5 character in an 8 x 5 matrix. The graphic character is a 4 x 5 matrix.

The size of a graphic character is the same as the size of a text character. In addition, the text characters may be in color. The resolution (horizontal) for a graphic character is twice as coarse as the dot spacing for a text character. One of eight colors may be selected for foreground and for background within a particular character.

Figure 5 shows how the display character can be specified using four bytes. The first byte determines whether the character is a text character or a graphic character, and specifies the colors for foreground and background. If it is a text character, the second byte specifies the character with a seven-bit ASCII code, and the remaining two bytes are not used. If it is a graphics character, the second, third, and fourth bytes contain the color specification for each of the twenty distinct picture elements (pixels) within the character. Use of the foreground color is indicated by a one in the respective bit position, while a zero specifies use of the background color.

The screen format chosen has 80 characters per row and 48 rows. Thus the resolution (in terms of picture elements) is 640 x 480 for text characters and 320 x 240 for graphic characters. A full screen contains 80 x 48 = 3840 characters. Thus, a single frame of the display can be represented by 3840 x 4 = 15,360 bytes. If the screen were updated 60 times per second, the CRT refresh function would require a DMA transfer rate of 15,360 x 60 = 921,600 bytes per second.

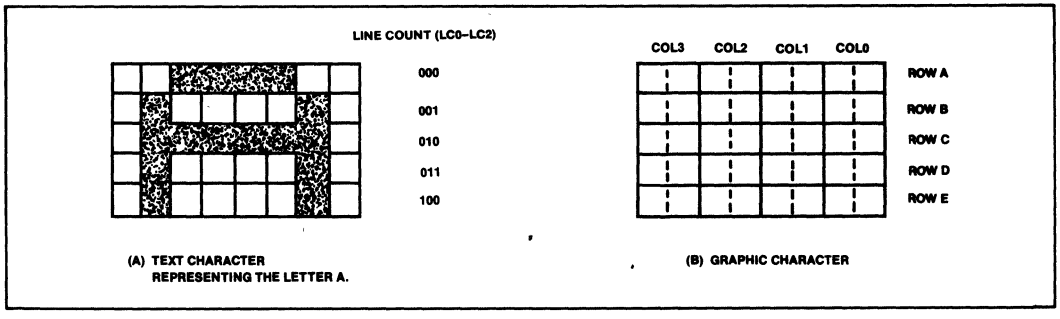


Figure 4. Character Structure

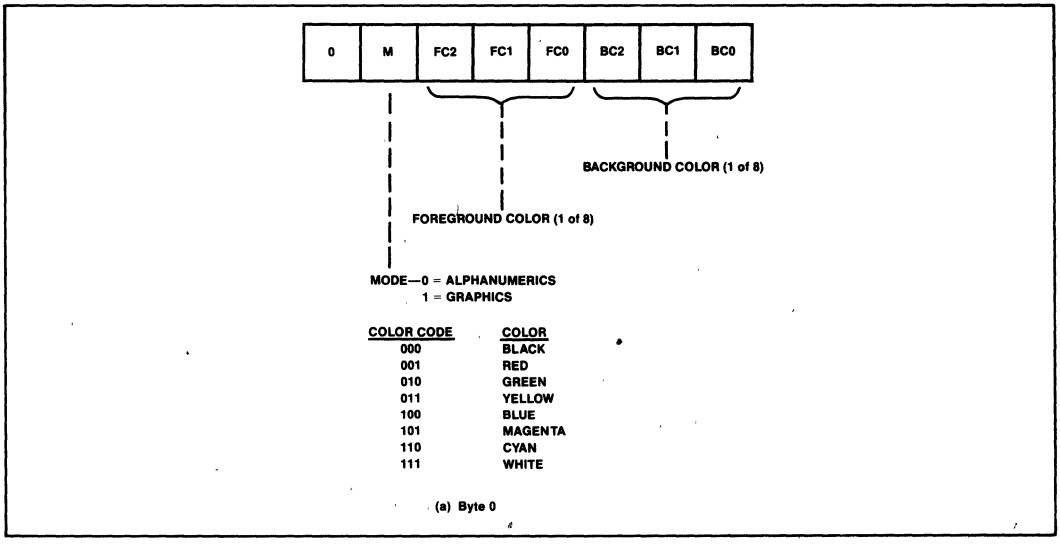
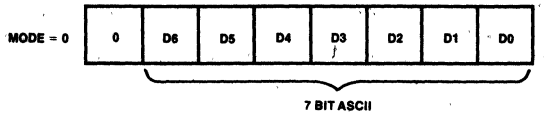
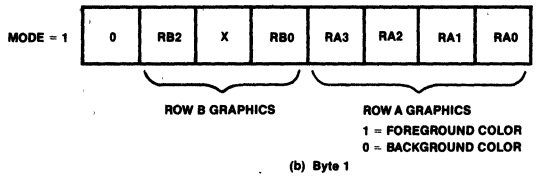


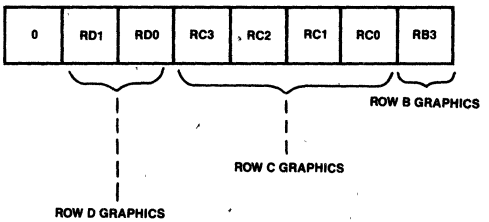
Figure 5. Display Character Specification



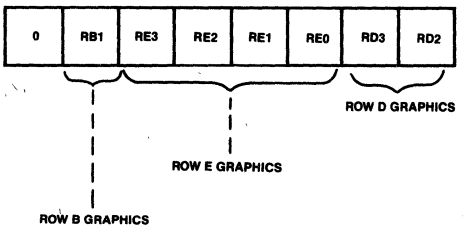


(b) Byte 1

NOTE: RB1 IS INTENTIONALLY MOVED TO BYTE 3 SUCH THAT REPRESENTATION OF A BLANK CHARACTER FOR EITHER TEXT OR GRAPHIC IS THE SAME.



(c) Byte 2



(d) Byte 3

Figure 5. Display Character Specification (Cont.)

System Bottlenecks

It can be seen from the above calculation that nearly one megabyte of data must be transferred per second to effect the CRT refresh function alone. Even with the fastest available DMA controllers, this represents the major part of the bandwidth for such devices. When the design shown in Figure 1 is used, the system bus must also be used by the CRT terminal processor for such functions as keyboard polling and communication with external devices. In addition, any changes made to the material being displayed would require use of the system bus for the purpose of storing the new material in the display memory, and possibly also for access to system memory during the calculation process. It is easy to see, therefore, that severe bottlenecks can occur in terms of system bus utilization. Problems involving bus contention could also be difficult to resolve. Display underruns could become difficult or impossible to avoid in some cases, such as when graphics computations require excessive use of the system bus.

The situation can be improved substantially if provision is made for concurrent processing. One CPU can be doing calculations on the material to be displayed, while another CPU can be managing the CRT terminal functions and the I/O devices simultaneously. Local buses can be used for access to the respective program memories, with the system bus used only for transfer of new display data and for communication between the two processors.

The iAPX 86/11 offers a convenient and economical way of implementing this multiprocessing approach. In particular, the 8089 has unique capabilities that simplify the design process.

OVERVIEW OF THE 8089

Architectural Overview

The 8089 Input/Output Processor is a complete I/O management system on a single chip. It contains two independent I/O channels, each of which has the capabilities of a CPU combined with a programmable DMA controller.

The DMA functions are somewhat more flexible than those of most DMA controllers. For example, a conventional DMA controller transfers data between an I/O device and a memory. The 8089 DMA function can operate between one memory and another, between a memory and an I/O device, or between one I/O device and another. Any device (I/O or memory) can physically reside on the system bus or on the I/O bus. The bus width for the source and destination need not be the same. If the source, for example, is a 16-bit device, while the destination is an 8-bit device, the 8089 will disassemble the 16-bit word automatically as part of the DMA transfer process. The transfer can be synchronized by the source, by the destination, or it can be free running. The 8089 can effect data transfers at rates up to 1.25 megabytes when a 5 MHz clock is used.

Unlike most DMA controllers, the 8089 uses a two-cycle approach to DMA transfer. A fetch cycle reads the data from the source into the 8089, and a store cycle writes the data from the 8089 to the destination. This two-cycle approach enables the 8089 to perform operations on the data being transferred. Typical of such operations are translating bytes from one code to another (for example, EBCDIC to ASCII) or comparing data bytes to a search value.

A variety of conditions can be specified for terminating DMA transfers, including single cycle, byte count (up to 64K), external event, and data-dependent conditions, such as the outcome of a masked compare operation.

The CPU in each channel can execute programs in the system space (from a memory on the system bus) or in the I/O space (from a memory on a separate I/O bus). Thus, complete channel programs can be run by the 8089 without tying up the system bus or interfering with the operation of the system CPU. Figure 6 is a simplified block diagram of the 8089, showing how the 8089 interfaces with these two buses.

The programs that the 8089 executes may be preexisting programs stored in ROM or EPROM, or they may be programs prepared for the 8089 by the system CPU. In the latter case, the programs are typically in modular form, contained in "task blocks" that the system CPU places in a memory location accessible to the 8089. During normal operation, the system CPU then directs the 8089 to the various task blocks, according to which programs are to be executed. The details of how this is done are given below under *Software Interface.*

The 8089 has an addressing capability of 64K bytes in the I/O space, and thus can support multiple peripherals, as illustrated in Figure 7. In the system space, the 8089 supports 1-megabyte addressing, and is directly compatible with the 8086 or 8088, and with Intel's Multibus. The 8089 operates from a single +5V power source, and is housed in a standard 40-pin, dual in-line package. The instruction set for the 8089 IOP is specifically designed and optimized for I/O processing and control. In addition to being able to execute DMA

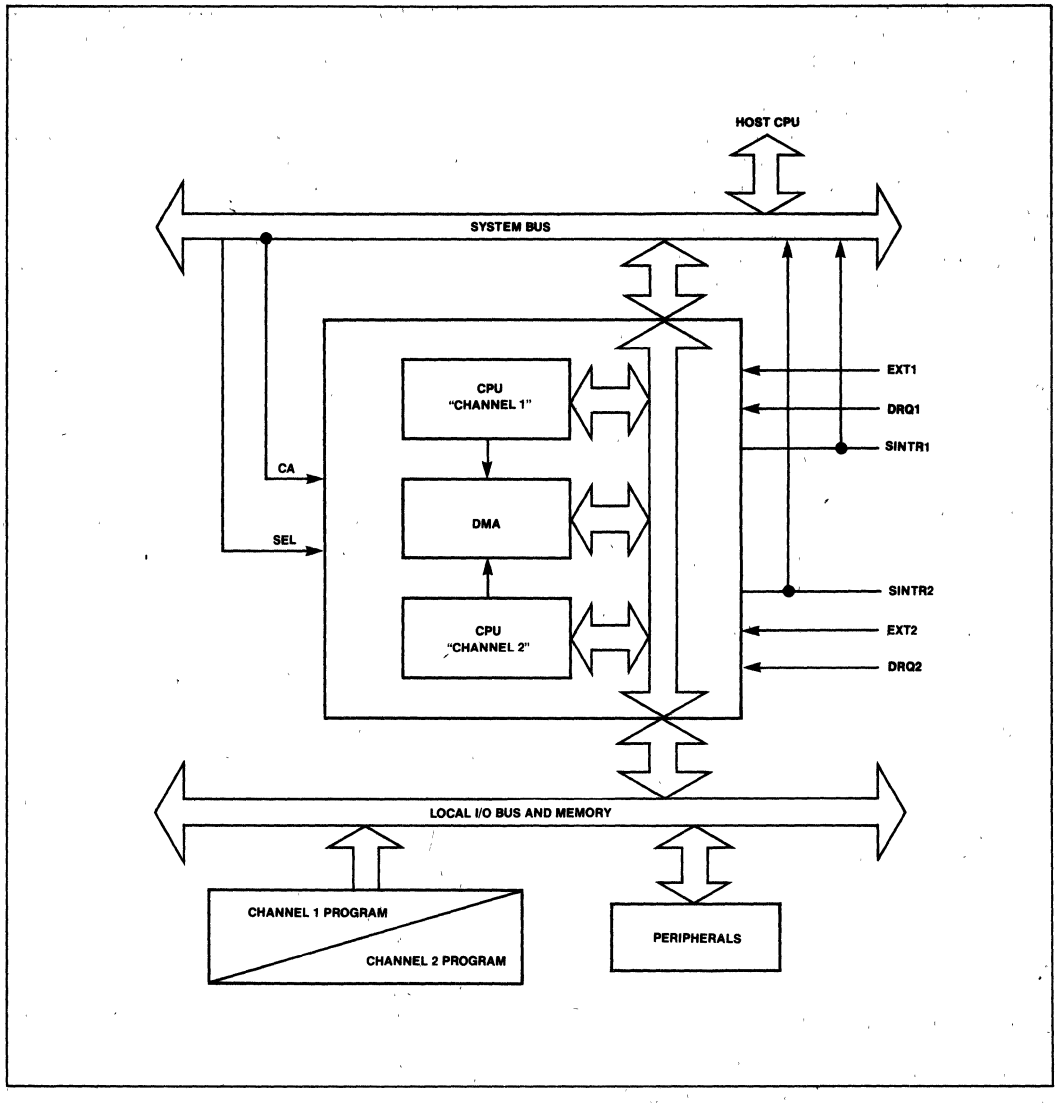


Figure 6. Simplified Block Diagram of the 8089

transfers under a wide variety of operating conditions, the 8089 can perform logic operations, bit manipulations, and elementary arithmetic operations on the data being transferred. A variety of addressing modes may be used, including register indirect, index auto increment, immediate offset, immediate literal, and indexed.

The register set for the 8089 is shown in Figure 8. Each channel has an independent set of these registers, not accessible to the other channel. Table 1 gives a brief summary of how these registers are used during a program execution or during a DMA transfer. Four of the registers can contain memory addresses which refer to either the system space or the I/O space. These registers each have an associated tag bit. Tag = 0 refers to the system space and tag = 1 refers to the I/O space. More details on how the registers are used are given below as part of the *Software Interface* section.

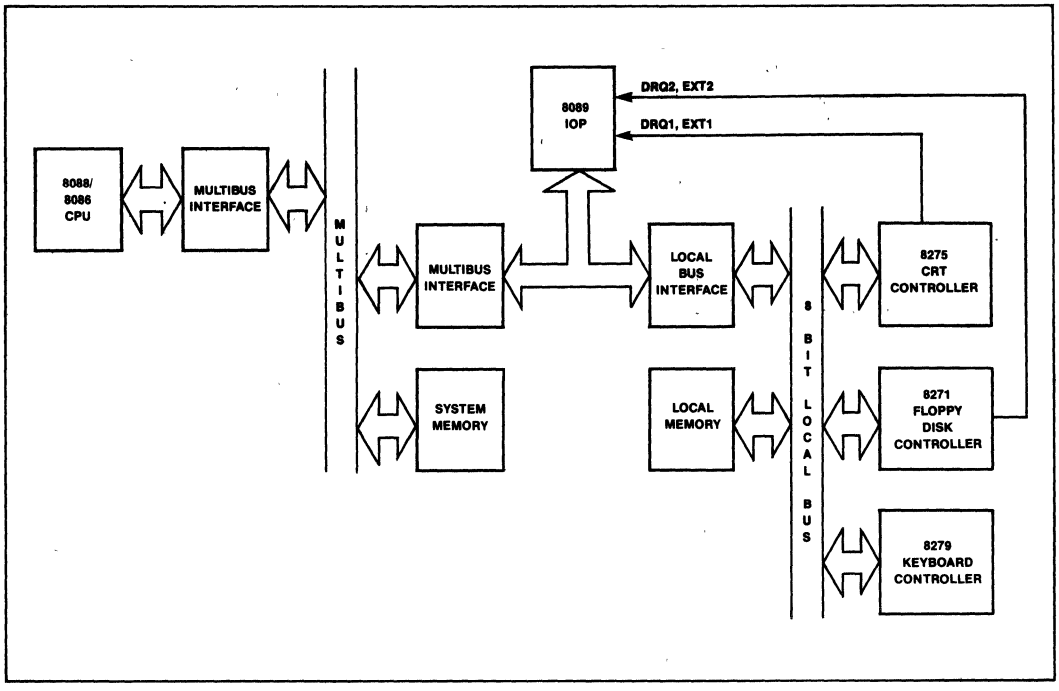


Figure 7. I/O System with Multiple Peripherals

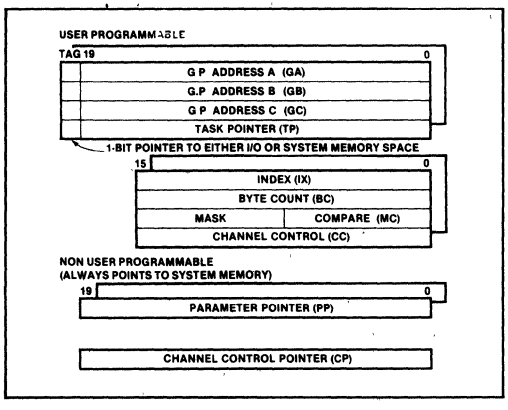


Figure 8. 8089 Register Set

System Configurations

The hardware relationship between the host CPU and the 8089 can take one of two basic forms—local configuration or remote configuration. In local configuration (Figure 9) the IOP shares the system bus interface logic with the host CPU. They reside on the same bus, sharing the same system address buffers, data buffers, and bus timing and control logic. The 8089 requests the use of the bus by activating the request/grant line to the host CPU. When the host relinquishes the bus, the IOP uses all the same hardware, and the host CPU is restricted from accessing the bus until the 8089 returns control of the bus to the host CPU.

The local configuration is a very economical configuration in terms of hardware cost, but it does not allow concurrent processing, and thus it is not able to really take advantage of the 8089's capabilities for independent operation. In the local configuration, the 8089 acts as a local DMA controller for the CPU, providing enhanced DMA capabilities and 1-megabyte addressing.

For applications such as the color graphics terminal, where system bus utilization (and other overhead) due to I/O processing would clearly be excessive in the local configuration, it is far more desirable to use the remote configuration, illustrated in Figure 10. The two processors both access the system bus, but each may have its own local bus in addition. Each of the processors may execute programs from memory on its own local bus, or

Table 1. Channel Register Summary

Register	Size	Program Access	System or I/O Pointer	Use by Channel Programs	Use in DMA Transfers
GA	20	Update	Either	General, base	Source/destination pointer
GB	20	Update	Either	General, base	Source/destination pointer
GC	20	Update	Either	General, base	Translate table pointer
TP	20	Update	Either	Procedure return, instruction pointer	Adjusted to reflect cause of termination
PP	20	Reference	System	Base	N/A
IX	16	Update	N/A	General, auto-increment	N/A
BC	16	Update	N/A	General	Byte counter
MC	16	Update	N/A	General, masked compare	Masked compare
CC	16	Update	N/A	Restricted use recommended	Defines transfer options

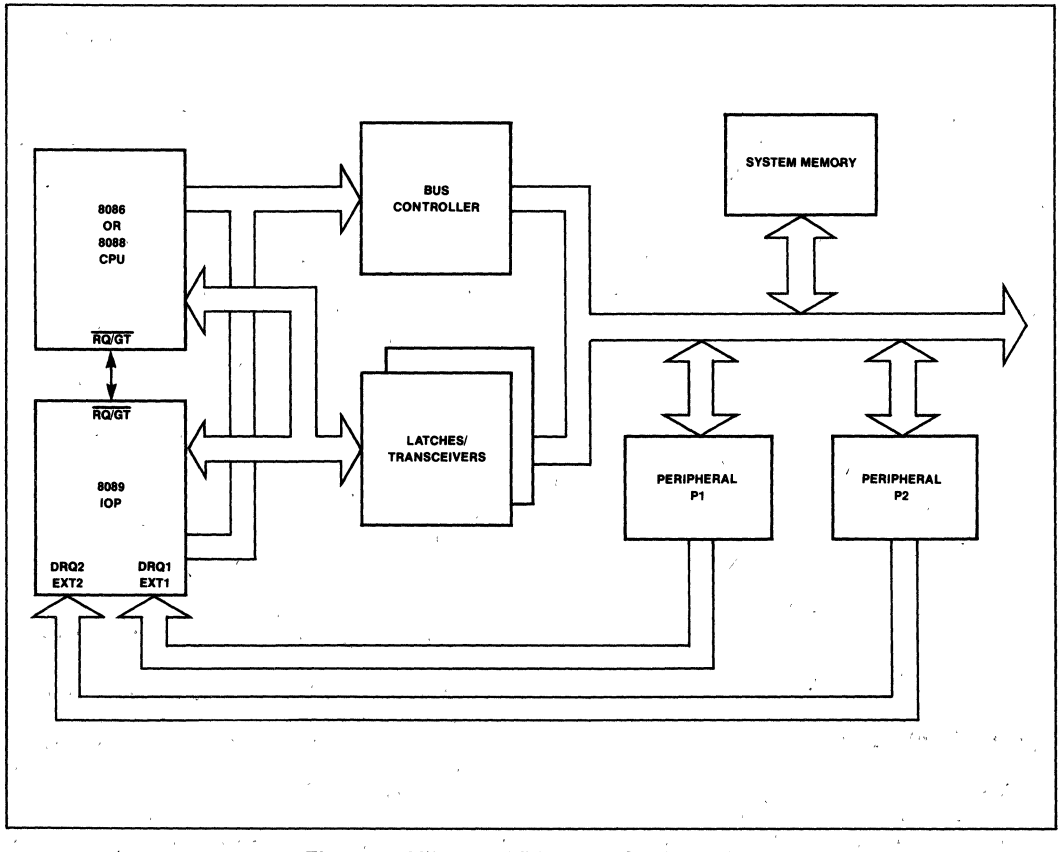


Figure 9. CPU and IOP in Local Configuration

AFN-02172A

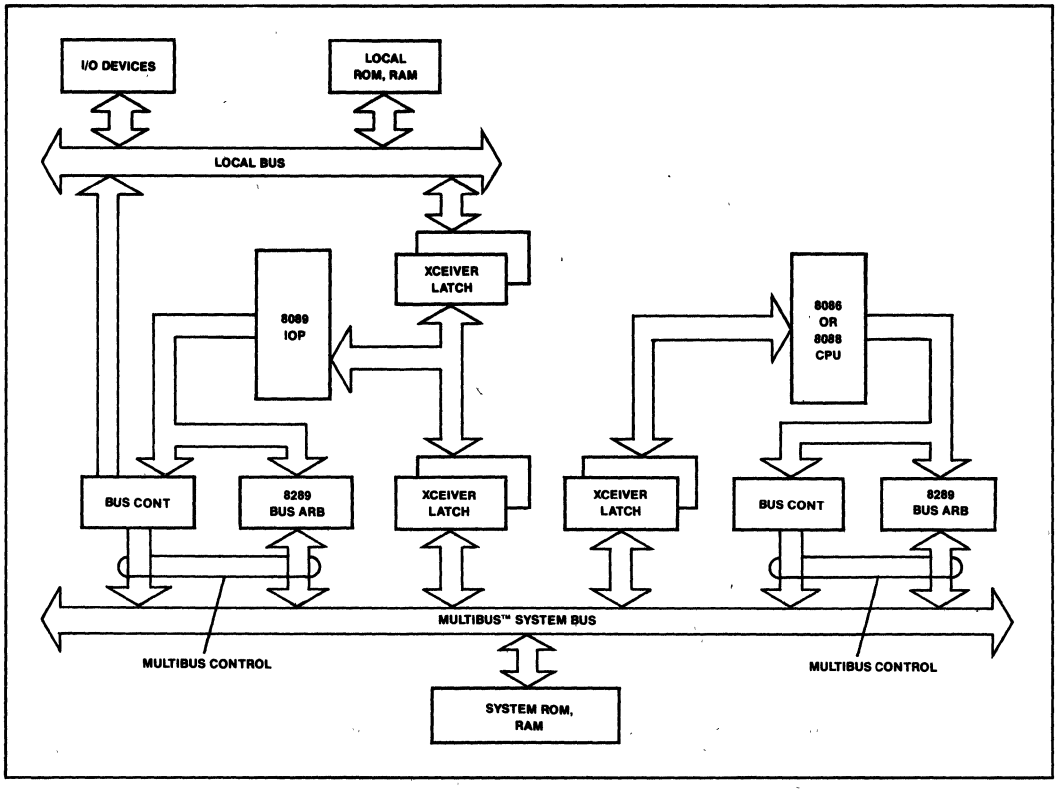


Figure 10. CPU and IOP in Remote Configuration

on the shared system bus. This creates a much more flexible arrangement. Concurrent processing may be used, and it is not necessary to synchronize the processors. An 8086, for example, may run at 8 or 10 MHz while the 8089 operates at 5 MHz. The specific terminal design described later in this application note makes use of one additional technique to further decouple the operation of the two processors. This is a dual-port RAM, which is located between the system bus and the 8089, and serves as display memory and as storage for the task blocks created by the 8086 CPU. Details on how this dual-port RAM operates are given below in the sections describing the terminal design itself.

Software Interface

Although the 8089 is an intelligent device which has a great deal of ability to function independently when managing the course of I/O operations, it typically operates under the overall supervision of the host CPU. Figure 11 illustrates the method of communication between the CPU and the IOP. The CPU communicates to the IOP by placing messages in memory and activating the IOP's channel attention (CA) input. The IOP communicates to the CPU by placing messages in system memory and making an interrupt request on one of its system interrupt request (SINTR-1 or SINTR-2) outputs.

The messages in memory take the form of linked blocks. These blocks are of the following five types:

1. System Configuration Pointer (SCP)
2. System Configuration Block (SCB)
3. Channel Control Block (CCB)
4. Parameter Block (PB)
5. Task Block (TB)

The SCP and SCB blocks are used by the CPU (only after reset) to initialize the 8089. The CCB, PB and TB blocks are used when the CPU wishes to instruct the

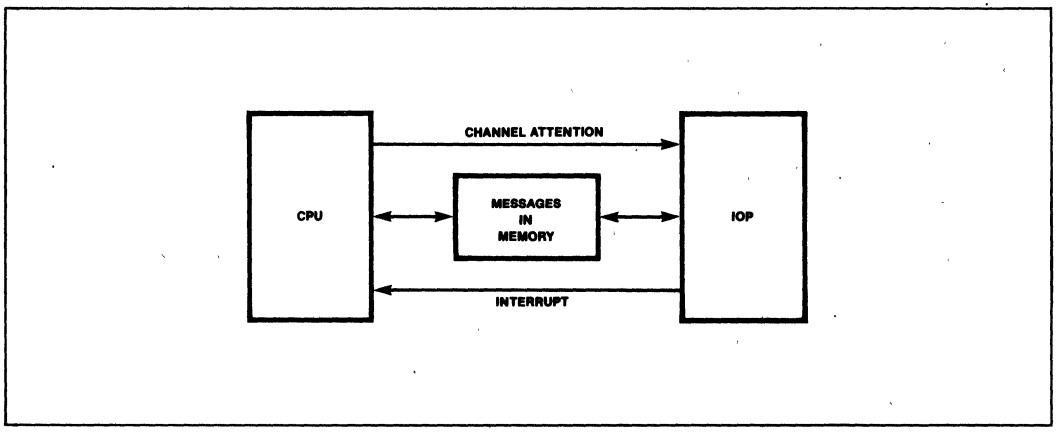


Figure 11. CPU/IOP Communication

IOP to perform a particular sequence of operations. Figure 12 shows these five blocks and how they are linked. The SCP, SCB, CB, and PB must be in memory which is accessible from both the CPU and 8089 (either system memory or for this application note, dual-port memory). The TB may be in either system or 8089 local memory.

The system configuration pointer is always found at the same location (FFFF6) in the system memory. The first time channel attention is activated (after an IOP reset) the 8089 reads the system configuration pointer from this location. The SYSBUS field contains only one significant bit (Bit 0), designated by the letter W. If W = 0, the system bus is an 8-bit bus. W = 1 denotes a 16-bit system bus. The IOP first assumes an 8-bit bus and reads the SYSBUS field. It stores the information as to the physical width of the system bus, then immediately uses this information in the process of fetching the next four bytes, which contain the address of the system configuration block.

The addresses used to link blocks are standard iAPX 86, 88 pointer variables, each occupying two word locations in system memory. The lower-addressed word contains an offset, which is added to the segment base value (left-shifted four places) found in the upper-addressed word to derive the complete 20-bit physical address in system memory. If the block is in an I/O memory (as a task block might be), only the offset value is used.

After thus deriving the address of the system configuration block, the IOP reads this block, starting with the system operation command (SOC) field. Bit 1 of the SOC field specifies the request/grant mode (used in local configuration or in multiple-IOP systems). Bit 0 specifies the I/O bus width (designated I). When I = 0, the I/O bus is an 8-bit bus. I = 1 denotes a 16-bit I/O bus. The IOP then proceeds to read the double-word pointer to the channel control block, converts it to the 20-bit physical address, and stores it in an internal register (the channel control pointer register). This register is loaded only during initialization and is not available to channel programs. For this reason the channel control block cannot be moved unless the IOP is reset and reinitialized.

The initialization is complete when the channel control pointer has been stored. The IOP indicates this by clearing the busy flag in the channel 1 control block (which must be set by the host CPU before the initialization sequence began). The host CPU can monitor this flag to determine when initialization is complete, and then to initialize any other 8089s in the system.

It is the responsibility of the host CPU to make sure that the SCP and SCB have the proper contents before issuing the channel attention (CA) that begins the initialization sequence. After initialization, the host CPU must also assure that the channel control block (CCB), parameter block (PB), and task block (TB) all have the proper contents, before issuing a subsequent CA.

The CA may be issued in the form of an I/O write command to the address of the IOP on the Multibus. Figure 13 shows a typical decoding circuit for this write command. The IOP actually occupies two consecutive address locations on this bus, because the A0 line is tied to the select (SEL) input of the 8089. A zero on the SEL line specifies IOP channel 1 for the impending operation, while a one specifies IOP channel 2.

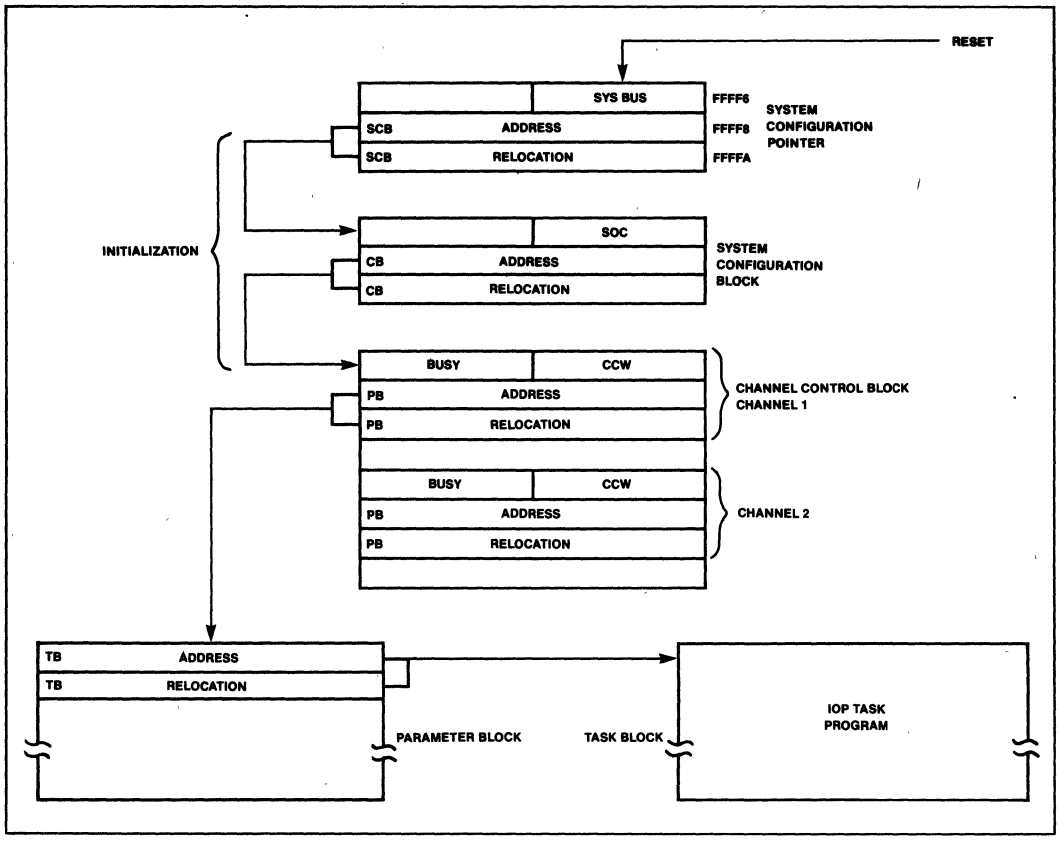


Figure 12. Linked Block Communication Structure

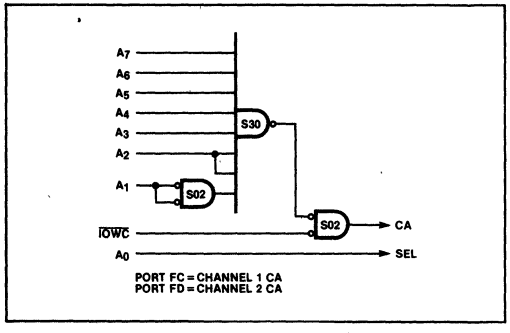
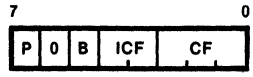


Figure 13. Channel Attention Decoding Circuit

The channel control block has a section for each channel. When the CA is received, the IOP goes to the section corresponding to the selected channel, and reads the channel command word (CCW). It then sets or clears the busy flag (FFH = set, 00H = clear). The encoding of the channel command word is shown in Figure 14. The CCW provides the IOP with a functional command (START in I/O space, HALT, etc.) and specifies some of the operating conditions, such as interrupt handling, bus load limit, or priority relative to the other channel. If the CPU is instructing the IOP to execute a program, it is at this point that the CPU specifies, via the CCW, whether the instructions are to be fetched from the system space or from the 8089's I/O space. Refer to *iAPX 86,88 User's Manual* for specific details on the setting and clearing of the busy flag and on CCW specifications.

After the CCW has been read, the IOP reads (if appropriate to the command) the address of the parameter block associated with the impending operation, and stores the translated address (from the two-word segment and offset pair to the 20-bit physical address) in



CF COMMAND FIELD
000 UPDATE PSW
001 START CHANNEL PROGRAM LOCATED IN I/O SPACE.
010 (RESERVED)
011 START CHANNEL PROGRAM LOCATED IN SYSTEM SPACE.
100 (RESERVED)
101 RESUME SUSPENDED CHANNEL OPERATION
110 SUSPEND CHANNEL OPERATION
111 HALT CHANNEL OPERATION

ICF INTERRUPT CONTROL FIELD
00 IGNORE, NO EFFECT ON INTERRUPTS.
01 REMOVE INTERRUPT REQUEST; INTERRUPT IS ACKNOWLEDGED.
10 ENABLE INTERRUPTS.
11 DISABLE INTERRUPTS.

B BUS LOAD LIMIT
0 NO BUS LOAD LIMIT
1 BUS LOAD LIMIT

P PRIORITY BIT

Figure 14. Channel Command Word Encoding

the parameter pointer (PP) register. PP is another register which is not programmable by the channel program. The IOP then goes to this location in system memory, and fetches the address of the task block itself. The task block contains the actual program to be executed, while the parameter block contains parameters to be used by that program.

Except for the first two words, which contain the task block address, the parameter block format is up to the discretion of the user. Similarly, the task block may have any format whatsoever, as long as the IOP can execute the program. The parameter block is always in system memory, but the task block may be either in system memory or in I/O (local) memory.

The host CPU may prepare as many parameter-block/task-block sets as it wishes. An individual set is then activated for execution by placing its parameter block pointer in the desired channel's control block, loading the appropriate channel control word, and issuing a CA to that channel.

The registers shown in Figure 8 store (in addition to pointer variables) various flags and parameters associated with the IOP's operation. Some of these registers are loaded automatically with information fetched during the initialization sequence or during channel attention processing. Others must be set by executing a program using instructions from the IOP's instruction set that are specifically designed for loading these registers.

Channel programs (task blocks) are written in ASM-89, the 8089 assembly language. About 50 basic instructions are available. The IOP instruction set contains some instructions similar to those found in CPUs, and also other instructions specifically tailored to I/O operations. Data transfer, simple arithmetic, logical, and address manipulation operations are available. Unconditional jump and call instructions are provided so that channel programs can link to each other. An individual register or even a single bit may be set or cleared with a single instruction. Other instructions specify conditional jumps, initiate DMA transfers, perform semaphore operations, and issue interrupt requests to the CPU.

A channel program typically ends by posting the result of an operation to a field supplied in the parameter block, then interrupting the CPU (if interrupts are enabled) and halting. When the channel halts, its associated BUSY flag is cleared in the channel control block. The CPU can poll this flag (as an alternative to being interrupted) to determine when the operation has been completed.

Timing Details

The basic bus timing relationships for the 8089 are identical to those of the 8086 or 8088, in that all cycles consist of four states (assuming no wait states), and use the same time-multiplexing technique for the address/data lines. The address (and ALE signal from the

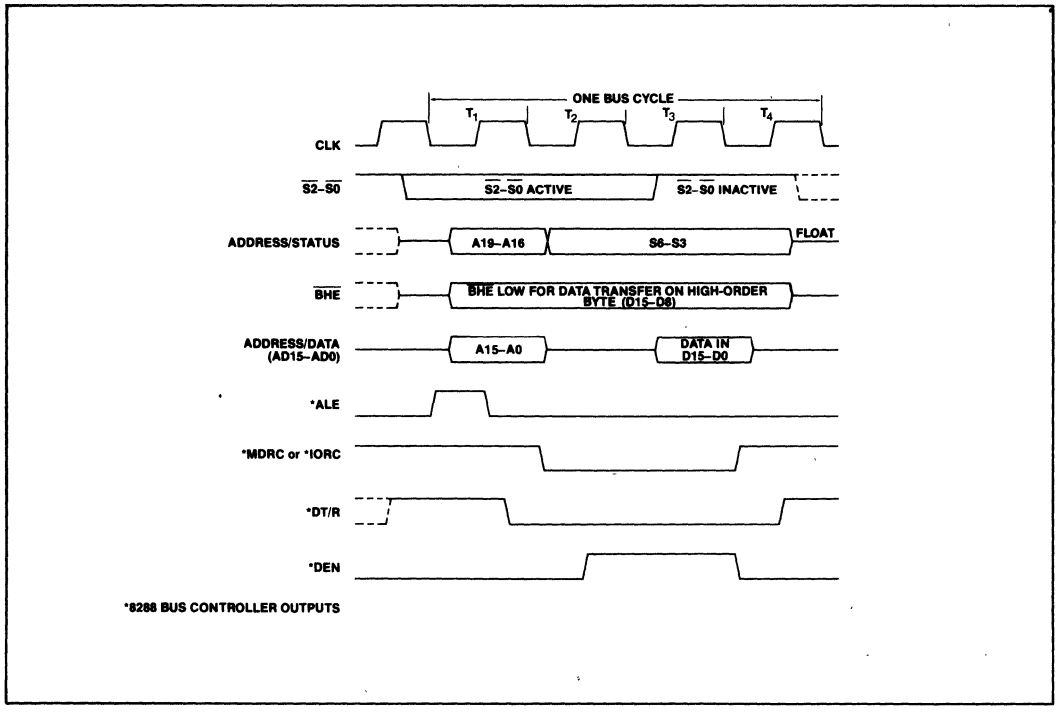


Figure 15. Read Bus Cycle

8288 bus controller) is output during state T1 for either a read or write cycle. During state T2 for a read cycle (Figure 15) the address/data lines are floated. During state T2 for a write cycle (Figure 16) data is output on these lines. During state T3, the write data is maintained or the read data is sampled. The bus cycle is concluded in state T4.

Figure 17 shows some details on the wait state timing and Figure 18 shows the RESET-CA initialization timing.

During DMA transfers, the transfer cycle may be synchronized by either the source or the destination. Figure 19 (source-synchronized transfers) and Figure 20 (destination-synchronized transfers) show the relationships among the basic clock cycles, the DRQ signals, and the DACK signals.

The 8089 does not have a DACK output signal. Rather, it uses the more general process of issuing a command (for example, I/O read or write) to an address on the I/O bus. This command is then hardware decoded to obtain a chip select signal for the addressed device. This method enables the 8089 to relate to a variety of I/O devices in a very flexible manner.

Figures 19 and 20 also show how the 8089 inserts idle clocks to accommodate various DRQ latency conditions. If maximum efficiency (transfer rate) is desired, it is usually possible to remove this latency by techniques such as generating an early DRQ. Another possibility is to use the unsynchronized DMA transfer mode (DRQ is not examined) and to use the READY signal for synchronizing transfers. The early DRQ technique will be discussed later.

GRAPHIC CRT SYSTEM DESIGN

Having examined the requirements for graphic CRT systems in general, and having also discussed the capabilities of the 8089, we can now proceed to describe a specific graphic CRT design using the 8089.

In this design, the system CPU is an 8086. Thus, the entire system is called an iAPX 86/11.

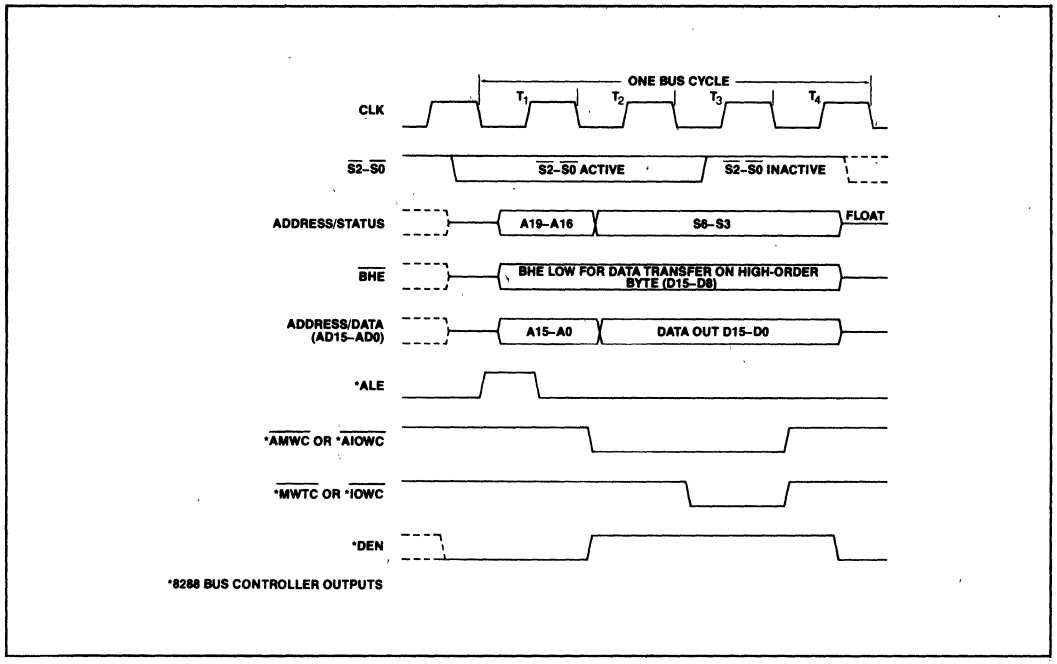


Figure 16. Write Bus Cycle

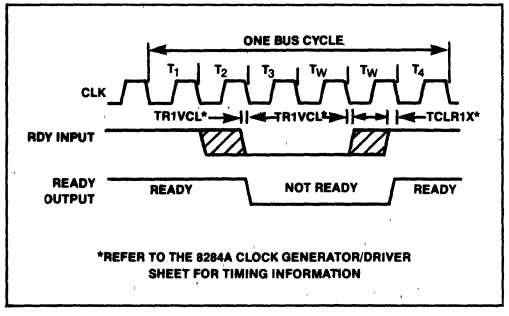


Figure 17. Wait-State Timing (Synchronous RDY Input)

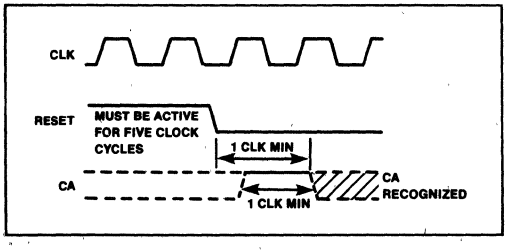


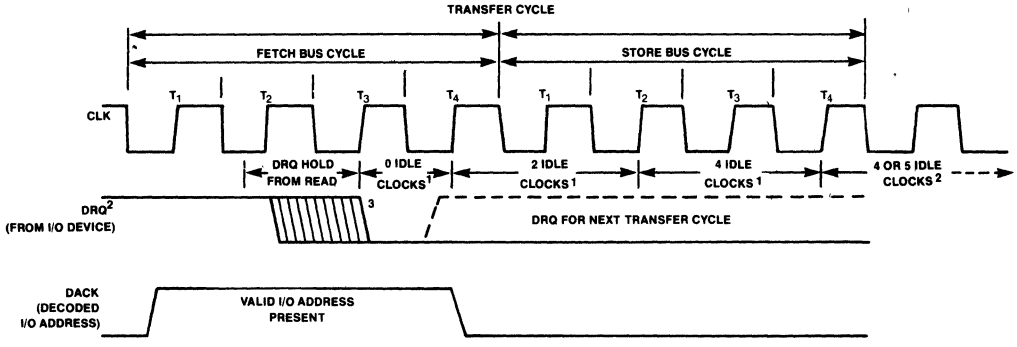
Figure 18. Reset and Channel Attention Timing

System Partitioning

The 8086 and 8089 are arranged in the remote configuration. This assures that concurrent processing can occur. As mentioned earlier, an additional step is taken to further decrease system bus utilization for I/O-related processes. This step is the inclusion in the system of a dual-port RAM, located between the system bus and the 8089. This dual-port RAM contains the display memory and also contains the linked message blocks used for communication between the 8086 and the 8089.

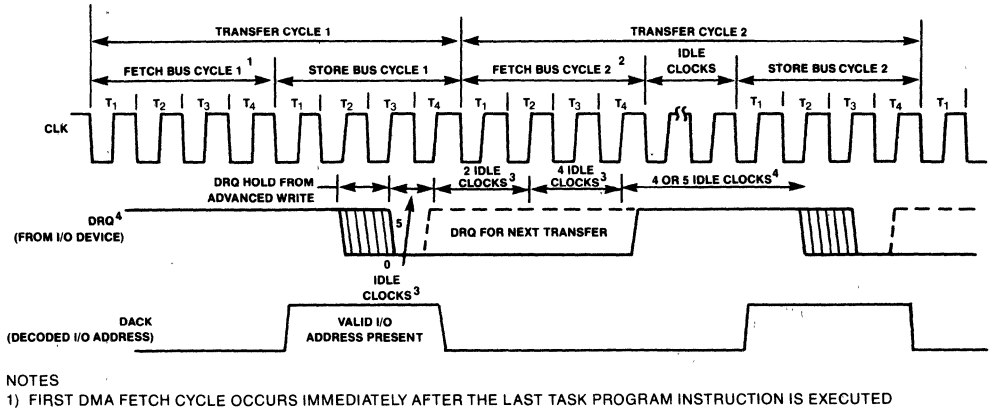
The system configuration then becomes that shown in Figure 21. The dual-port RAM becomes the only data path between the 8086 and the 8089. Access to this memory is time-shared between the 8086 and the 8089, with the 8089 taking less than 50% of the total time available. Since the 8089 does not access the system bus, the host system can enjoy complete freedom to allocate its resources between its own local bus and the system bus. The CPU and the IOP can operate asynchronously, with the 8086 running on an 8 MHz clock and the 8089 on a 5 MHz clock.

The division of responsibility between the 8086 and the 8089 is then very clearly defined. The 8086 initializes the 8089 and specifies the task parameters, storing them in



NOTES
1) INDICATES THE NUMBER OF IDLE CLOCKS INSERTED AFTER T_4 OF THE STORE CYCLE BEFORE THE NEXT TRANSFER CYCLE BEGINS IF DRQ IS RECEIVED BEFORE THE RISING EDGE OF CLK IN THE CURRENT FETCH CYCLE, THE NEXT FETCH BEGINS IMMEDIATELY AFTER THE CURRENT STORE
2) IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FOUR OR FIVE MORE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED TRANSFER CYCLE BEGINS (DRQ IS LATCHED ON THE RISING EDGE OF CLK)
3) TO PREVENT THE START OF THE NEXT TRANSFER CYCLE, DRQ MUST BE BROUGHT LOW BY THE RISING EDGE OF CLK IN T_4 OF THE CURRENT FETCH (FOR B/B→W SOURCE SYNCHRONIZED AND W→B/B DESTINATION SYNCHRONIZED IT MUST BE LOW BY THE RISING EDGE OF CLK IN THE FOURTH CLOCK OF THE CURRENT BUS CYCLE INCLUDING WAIT STATES)

Figure 19. Source-Synchronized Transfer Cycle



NOTES
1) FIRST DMA FETCH CYCLE OCCURS IMMEDIATELY AFTER THE LAST TASK PROGRAM INSTRUCTION IS EXECUTED
2) FETCH CYCLE 2 BEGINS IMMEDIATELY AFTER STORE CYCLE 1
3) INDICATES THE NUMBER OF IDLE CLOCKS INSERTED AFTER T_4 OF THE FETCH BEFORE STORE CYCLE 2 BEGINS IF DRQ IS RECEIVED BEFORE THE RISING EDGE OF CLK IN THE CURRENT STORE CYCLE, THE NEXT STORE BEGINS IMMEDIATELY AFTER THE NEXT FETCH
4) IF THE 8089 IS IDLE WHEN DRQ IS RECOGNIZED, FOUR OR FIVE MORE IDLE CLOCK CYCLES OCCUR BEFORE THE ASSOCIATED STORE CYCLE BEGINS (DRQ IS LATCHED ON THE RISING EDGE OF CLK)
5) TO PREVENT THE NEXT STORE CYCLE FROM OCCURRING, DRQ MUST BE BROUGHT LOW BY THE RISING EDGE OF CLK IN T_4 OF THE CURRENT STORE (FOR B/B→W SOURCE SYNCHRONIZED AND W→B/B DESTINATION SYNCHRONIZED, IT MUST BE LOW BY THE RISING EDGE OF CLK IN THE FOURTH CLOCK OF THE CURRENT STORE CYCLE INCLUDING WAIT STATES)

Figure 20. Destination-Synchronized Transfer Cycle

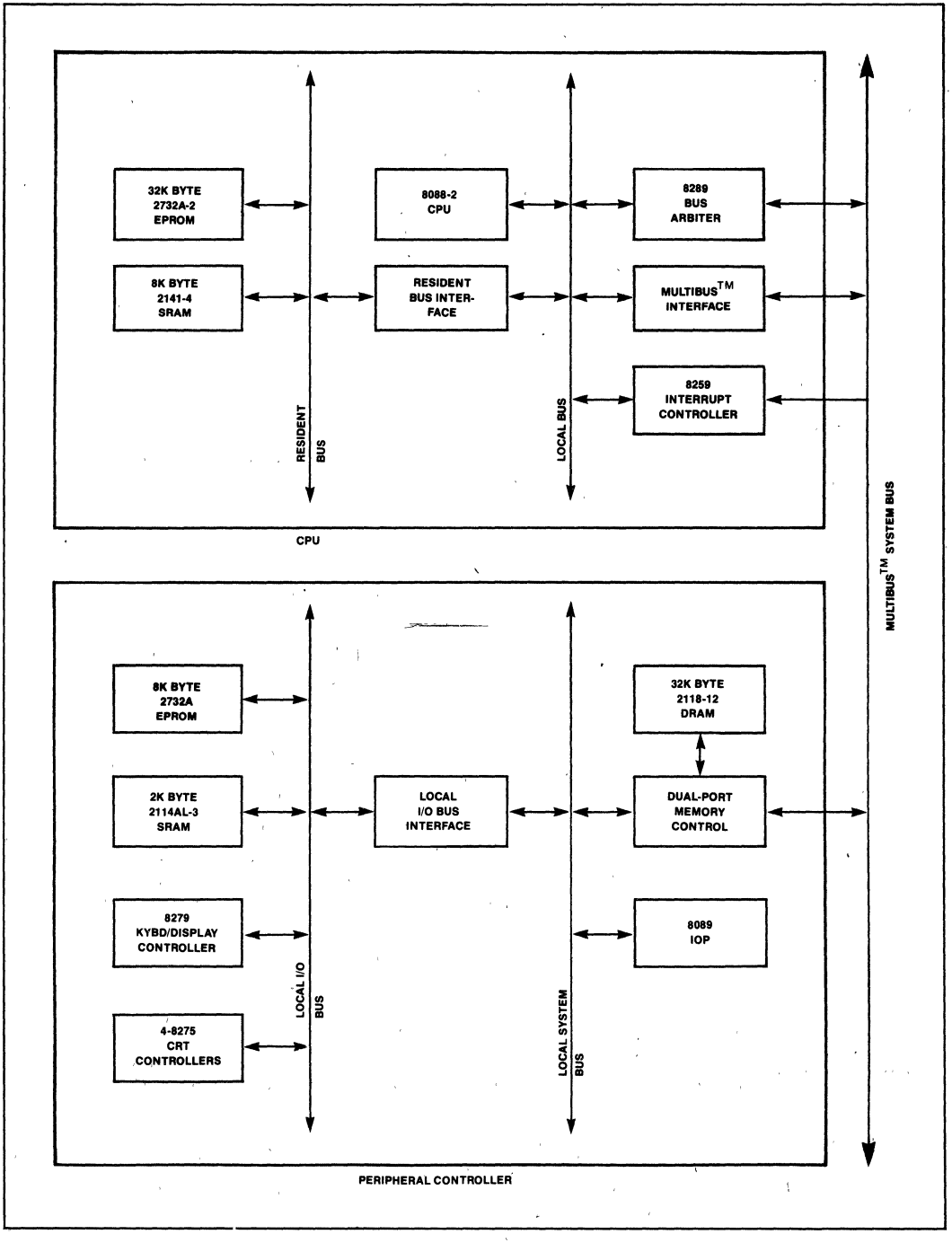


Figure 21. Remote Configuration with Dual-Port RAM

AFN-02172A

the dual-port RAM. In many cases, the 8086 also prepares the task programs and stores them in the dual-port RAM, from which they may be downloaded to a memory on the 8089's I/O bus. The 8089 executes the task programs (from the dual-port RAM or from a local memory on the I/O bus), while the 8086 simultaneously executes other control or application programs. The application programs may encompass a wide variety of operations, but they will always generate the display characters and store them in the dual-port RAM. The 8089 returns status to the 8086 when task program execution has been completed.

Figures 22 and 23 show the manner in which the memories are organized. Figure 22, which shows the memory configuration for the 8086, should be taken as an example, since many different configurations are possible, according to the user's application. Figure 23 shows the memory configuration for the 8089, given the particular choices made for the application discussed in this note. Of the memories shown in Figure 22, the 2141 static RAMs and the 2732A EPROMs are located on the 8086's local bus, while the 2816 EEPROM and the 2118 dual-port RAM are interfaced to the Multibus. The 2816 is a non-volatile read/write memory equivalent in its storage capacity to the 2716 EPROM.

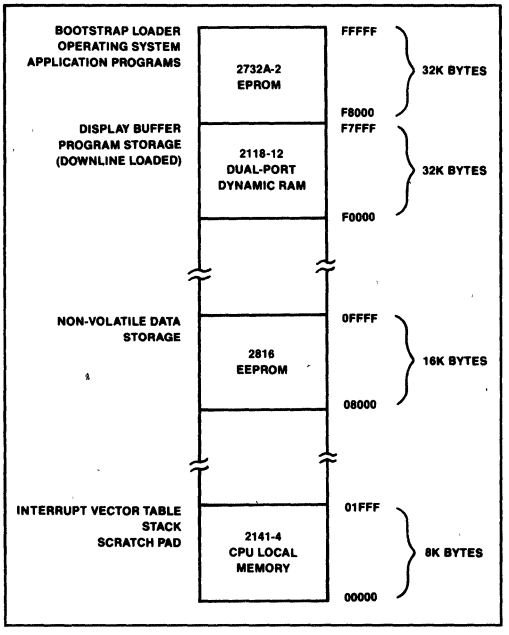


Figure 22. CPU Memory Organization

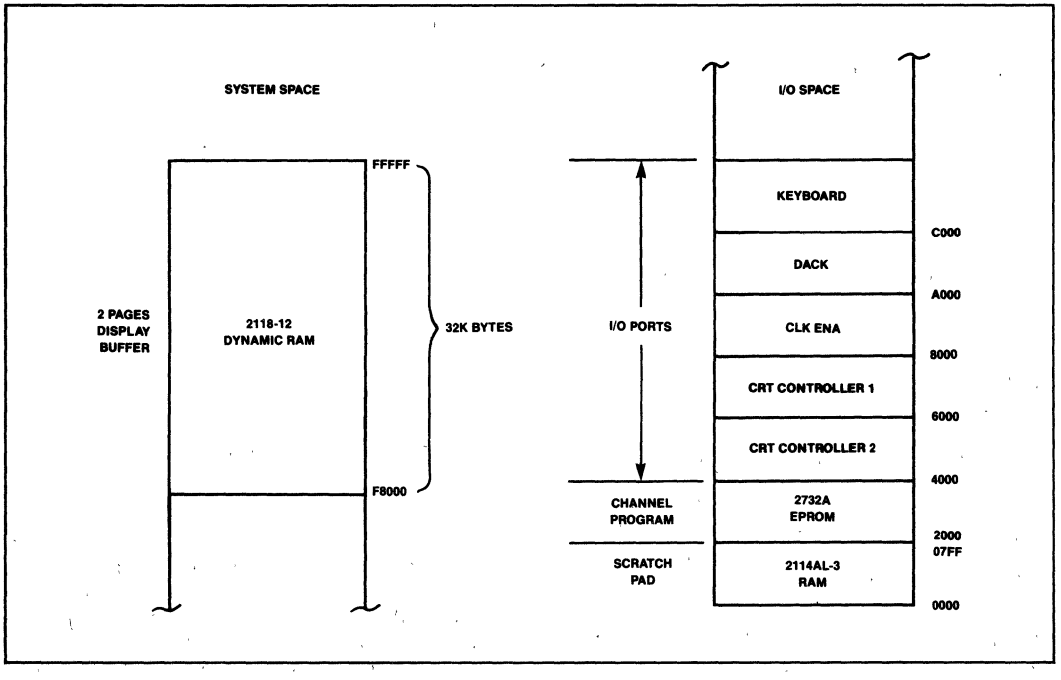


Figure 23. IOP Memory Organization

8086/8089 Software Interface

Comparing Figures 22 and 23, it can be seen that the 2118 dynamic RAM appears in the memory configurations for both the 8086 and the 8089. In the 8086's system space, this memory occupies addresses F0000 through F7FFF, while in the 8089's system space, its address range is F8000 through FFFFF.

Figure 24 shows the organization of the dual-port RAM. The addresses given are those seen by the 8089. The display data (for the CRT refresh function) is contained in the two largest blocks—Display Page 0 and Display Page 1. Each page contains 15K bytes, enough to refresh a color graphic screen containing 48 rows of 80 characters each. In typical operation, the 8086 and the 8089 both access the same page of display data. In special cases, such as animated displays, the 8089 performs repetitive DMA transfers from one of these pages, while the 8086 is generating new display material and storing it in the other page. The display page pointer (DSPLY_PG_PTR) in the parameter block specifies which of these pages is to be displayed at any given time. This pointer may be changed by the 8086, or by a command from the terminal keyboard.

The Command Buffer is a 256-byte area set aside for transferring ASCII characters from the 8086 to the 8089. It is like a second keyboard, scanned by the 8089. It takes precedence over any real keyboard activity. The COM_8086 flag in the parameter block is used to indicate when there are entries in the command block area.

The EEPROM Buffer is a 256-byte area used in connection with the non-volatile EEPROM memory, an optional memory which may be located on the Multibus. One use of such a memory would be to store ASCII strings, which could then be recalled by the 8086 upon recognition of special keyboard control code sequences.

The Keyboard Buffer is a 256-byte area which serves as a storage area for ASCII characters entered from the terminal keyboard. When this buffer becomes full, or when a return is entered at the keyboard, an end-of-file byte is placed after the last entered character, and the keyboard buffer full (KBD_BUF_FULL) flag is set in the parameter block. This prevents the 8089 from processing any more inputs from the keyboard, until the 8086 resets KBD_BUF_FULL.

The Spare blocks total 1K (1024) bytes, and may be used for any purpose, according to the user's application.

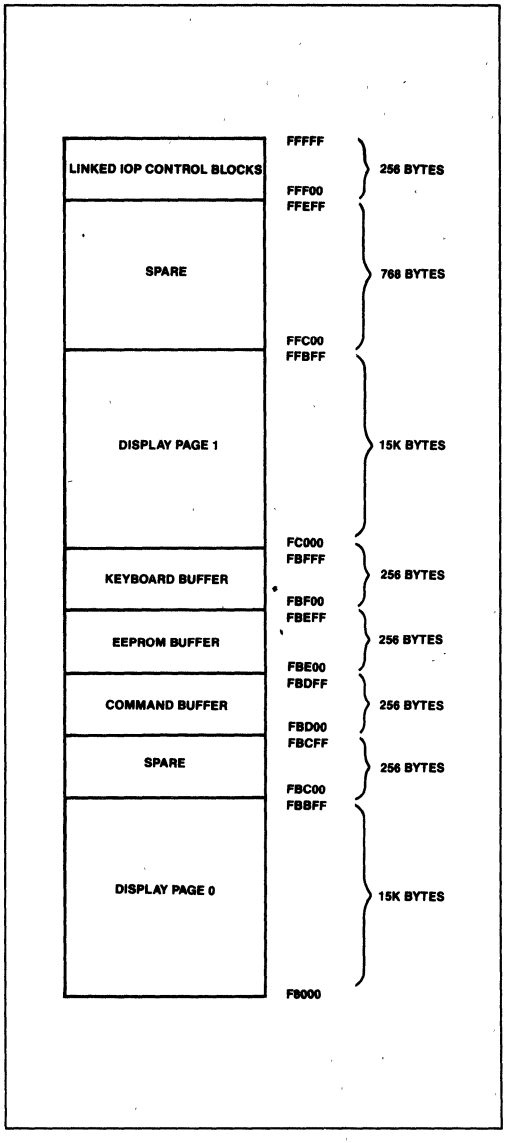


Figure 24. Organization of the Dual-Port RAM

The Linked IOP Control Blocks are those which have been discussed above, as part of the 8089 overview. The specific memory locations are as shown in Figure 25. Note that there is only one parameter block, and no task blocks present. Only one task block is used in this application, and it is stored in the 2732A EPROMs on the 8089's I/O bus.

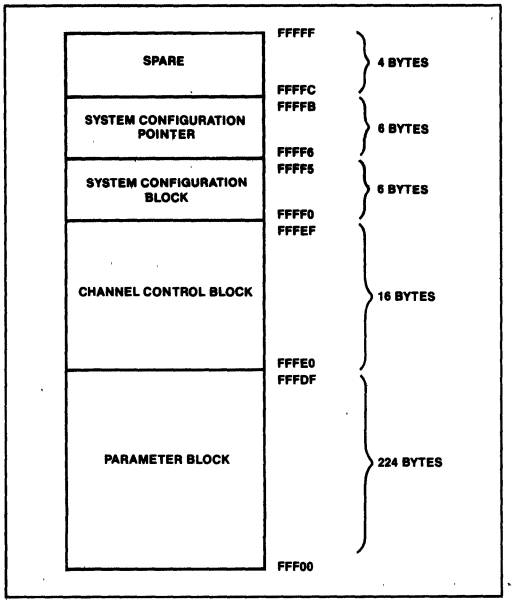


Figure 25. Organization of the Linked IOP Control Blocks Area

As mentioned earlier, the structure of the parameter block is very flexible. Only the first four bytes are fixed (because of the 8089's requirements). These four bytes contain the address of the task block. The remaining space in the parameter block may be defined by the user. The following list shows the parameter block structure that is used in support of the channel program contained in the 2732A EPROMs on the 8089's I/O bus.

TP_LSW	DW
TP_MSD	DW
EEP_INH	DB
EEP_BUF_FULL	DB
EEP_RECALL	DB
COL_CH_INH	DB
KBD_INH	DB
KBD_BUF_FULL	DB
COM_8086	DB
COLOR	DB
STR_PTR_8086	DW
BACK_COL_SW	DB
MON_INH	DB
DSPLY_PG_PTR	DB
SCROLL_REQ	DB
MON_HOM	DW
MON_END	DW
MON_LMARG	DW
MON_RMARG	DW
KBD_BUF_PTR	DW

In the above table, DB represents a one-byte quantity, and DW represents a two-byte quantity.

TP_LSW and TP_MSD are the two words making up the task pointer. However, since in this application the task program is in the I/O space, only the least-significant word (LSW) is fetched.

EEP_INH, when not equal to zero, indicates that the EEPROM buffer is closed to keystrokes or 8086 ASCII commands.

EEP_BUF_FULL, when not equal to zero, indicates that the EEPROM buffer is full.

EEP_RECALL, when not equal to zero, indicates that the 8089 is recalling the contents of an EEPROM buffer area.

COL_CH_INH, when not equal to zero, inhibits the color control keys on the keyboard.

KBD_INH, when not equal to zero, inhibits the processing of keystrokes (entered at the keyboard) by the 8089. Up to 6 keystrokes may be saved in the keyboard controller and may be processed later.

KBD_BUF_FULL, when not equal to zero, indicates that a new line of keyboard data needs to be processed by the 8086. The 8089 sets KBD_BUF_FULL equal to −1 when the return key is pressed. The 8086 resets KBD_BUF_FULL to zero after it has read this data.

COM_8086, when not equal to zero, indicates that there are ASCII commands in the command buffer areas of dual-port RAM that need to be processed by the 8089.

COLOR determines the foreground and background colors to be used in connection with ASCII characters entered at the keyboard, or sent by the 8086 via the command buffer area. In the COLOR byte, bits B0–B2 determine the background color, while B3–B5 determine the foreground color. The following code is used:

000	Black
001	Red
010	Green
011	Yellow
100	Blue
101	Magenta
110	Cyan
111	White

STR_PTR_8086 is a two-byte quantity that serves as an offset address for the ASCII characters in the command buffer.

BACK_COL_SW determines whether the 8089 color control keys will alter the foreground or the background portions of the COLOR byte. If BACK_COL_SW equals zero, the foreground color is altered. If BACK_COL_SW is not equal to zero, the background color is altered.

MON_INH, when not equal to zero, suspends DMA transfers by the 8089 from display memory to the 8275s. When MON_INH is cleared, DMA will resume.

DSPLY_PG_PTR determines which of the two display pages will be used to refresh the CRT. If DSPLY_PG_PTR equals zero, page 0 will be displayed. If DSPLY_PG_PTR does not equal zero, page 1 will be displayed.

SCROLL_REQ is set by the 8089 to indicate to the 8086 that the cursor is at the bottom of the page, and that key entry/command processing has been halted, pending a display memory scroll operation. When the 8086 has performed this operation, it clears SCROLL_REQ.

MON_HOM, MON_END, MON_LMARG, and MON_RMARG specify, respectively, the upper, lower, left, and right boundaries of the region on the screen in which keyboard entries will be displayed.

KBD_BUF_PTR is a two-byte quantity that serves as an address for the ASCII characters in the keyboard buffer.

Note that a number of these parameters support options (e.g., EEPROM buffer) and are not critical to the graphic operation described in this application note.

8089 Display Hardware Interface

This section describes the hardware of the peripheral processing module (PPM), which includes everything between the system bus and the CRT display/keyboard unit. The overall organization of the PPM is as shown in Figure 21. The dual-port RAM can be accessed from either the system bus or the 8089's local bus. The 8089 is said to be operating in the system space when it is accessing the dual-port RAM, and in the I/O space when it is accessing devices on the I/O bus. Included on the I/O bus are four 8275 CRT controllers, an 8279-5 keyboard controller, two 2732A EPROMs, which are used to hold channel programs, and four 2114 static RAMs, which are used as scratch-pad RAM for the 8089.

As explained above (under *OVERVIEW OF CRT GRAPHIC SYSTEMS, Performance Requirements)*, four bytes are used to specify each character in the display. The first byte determines whether the character is a text character or a graphic character, and specifies the colors for foreground and background. If it is a text character, the second byte specifies the character with a seven-bit ASCII code, and the remaining two bytes are not used. If it is a graphics character, the second, third, and fourth bytes contain the color specification for each of the twenty distinct picture elements (pixels) within the character. Use of the foreground color is indicated by a one in the respective bit position, while a zero specifies use of the background color.

The structure of the display characters and the formats of the individual bytes are shown in Figures 4 and 5.

The four 8275 CRT controllers on the 8089's I/O bus are used to process the four bytes comprising each character. Since the 8089 can transfer two bytes at a time in DMA mode, the four bytes are transferred in two stages. In the first stage, the 8089 fetches the first two bytes from the dual-port RAM, and transfers these two bytes into the first pair of CRT controllers. In the second stage, the 8089 fetches the second two bytes from the dual-port RAM, and transfers these two bytes into the second pair of CRT controllers. This process is repeated 80 times to transfer the 80 characters making up each row in the display.

The distinction between text and graphic characters is entirely transparent to the 8089. Four bytes are transferred in every case, even though the text information only requires two bytes per character.

We shall now examine the hardware schematics in detail, to see how the various functions of the PPM are implemented. Figure 26 shows the 8089 IOP and its associated bus controller. At the top left are the inputs through which the 8089 is controlled. The DRQF signal (detailed later) is the DMA request that initiates the transfer of two bytes from the IOP to two of the four CRT controllers. DRQF comes from the 8275s via a one-shot, and is connected to the DRQ 1 input of the 8089.

IRQ is an interrupt request that comes from the 8275s. It is activated after an entire screen's video information has been transferred from the dual-port RAM to the 8275s. IRQ is connected to the EXT 1 input of the 8089. It is necessary to program the 8089 to terminate the DMA transfer on an external event, in order for this signal to be effective.

CA is the channel attention signal. Upon receipt of CA, the 8089 reads the channel control word (CCW) from the dual-port RAM. From the CCW, the 8089 determines the nature of the operation assigned to it by the

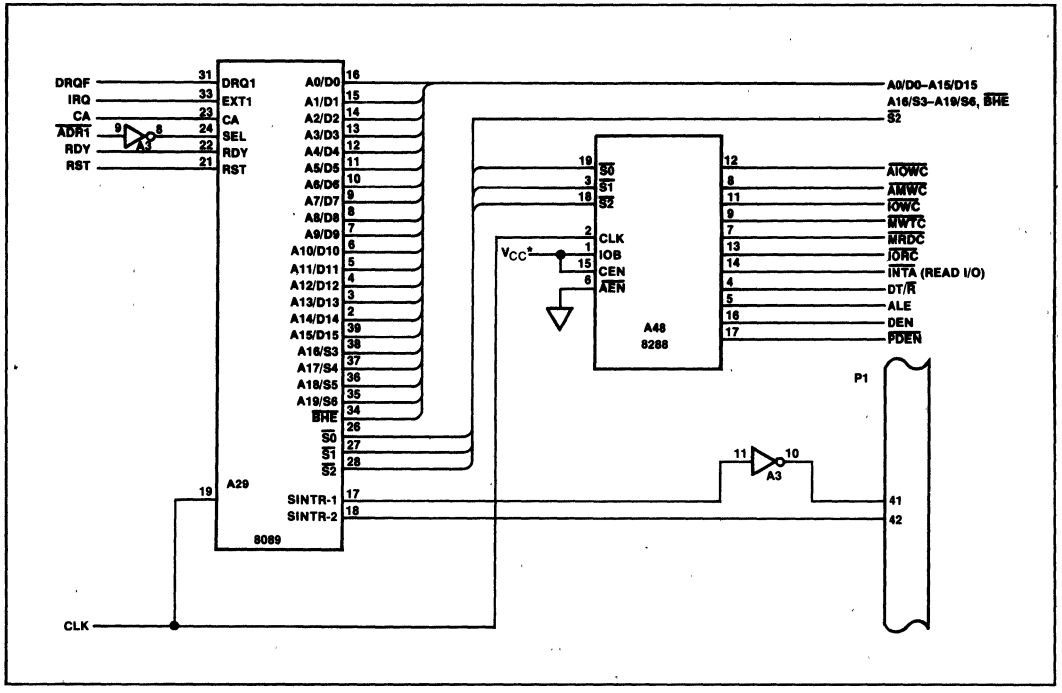


Figure 26. 8089 I/O Processor and 8288 Bus Controller

8086. CA is derived by hardware decoding of an I/O write command made by the 8086 to address 00H or address 01H on the Multibus. The lowest-order bit of this address is used to specify whether channel 1 or channel 2 of the IOP is to be selected, and is connected to the 8089's SEL input. In this application, the DMA transfers are always performed by channel 1.

RDY is the ready signal that comes from the 8202 dynamic RAM controller, and is synchronized by the 8284A clock generator. RDY is low whenever the 8086 is accessing the dual-port RAM. The RDY signal is used to establish a master/slave relationship between the 8086 and the 8089, with the 8086 as the master. As mentioned earlier, the 8089 accesses the dual-port RAM about 50% of the time during DMA transfers. It can be seen, referring to Figure 20, that if no idle clocks occur, the IOP will access the dual-port RAM during the four clock times of the DMA-fetch bus cycle, and will access the I/O bus during the four clock times of the DMA-store bus cycle. While the 8089 is doing the store operation, the 8086 can access the dual-port RAM. Once the 8086 has gained this access, the RDY signal will remain low until the 8086 is finished. The 8089 waits for RDY to go high before making a subsequent fetch.

At 5 MHz, the 8089 requires 3.2 microseconds (16 clock cycles) to transfer the four bytes representing a graphic character from the display memory to the four 8275s, assuming that no wait states have been inserted because of the 8086's access to the dual-port RAM, or because of dynamic RAM refresh functions. A complete row, consisting of 80 characters, requires 80 x 3.2 = 256 microseconds. The time allowed to complete the transfer of one row must be less than the time it takes to display that row on the screen. This latter time is equal to 1/50 of the total screen update time, or 1/3000 of a second (333 microseconds). Comparing the two figures (256 vs 333), it can be seen that there are 77 microseconds available for such wait states. It is the responsibility of the software designer to control the 8086's access to dual-port RAM in such a mannner that the added wait states do not total more than 77 microseconds in any span of 333 microseconds. Otherwise, underruns may occur and the CRT screen will be blanked. See *System Performance* (below) for further discussion on this effect.

RST is the IOP reset signal, which comes from the 8284A clock generator. The first CA after RST causes the IOP to access address FFFF6 in the dual-port RAM, in order to read the system configuration pointer.

Outputs from the IOP are the time-multiplexed address and data lines, BHE/ (bus high enable), status lines S0, S1, and S2, and the system interrupt request lines, SINTR-1 and SINTR-2. The interrupt lines go directly to the Multibus, and from there they become inputs to the 8086's 8259A interrupt controller.

Figure 27 shows the I/O address latches and decoder, and the circuitry used to generate the DACK/ signals for the CRT controllers. The IOP status bit S2 indicates whether the IOP is accessing the I/O space or the system space. Latched by ALE (address latch enable), S2/ generates IO and IO/. IO and IO/ are used to indicate that the 8089 is not accessing dual-port RAM. IO/ goes to the dual-port RAM controller.

The DACK/ signals are generated in the following manner:

1. Both 8275 pairs are accessed by the 8089 (DMA mode) via port A000H.
2. Hardware is used to select one pair of CRT controllers (bytes 0 and 1 or bytes 2 and 3).
3. As the 8089 reads (DMA) the word from the dual-port memory, address bit 1 (SA1) is latched with the memory read command (MRDC/).
4. When SA1 = 0, DACK 1/ is activated.
5. When SA1 = 1, DACK 2/ is activated.
6. In this manner the 8089 performs alternating writes (DMA) to the 8275 pairs.

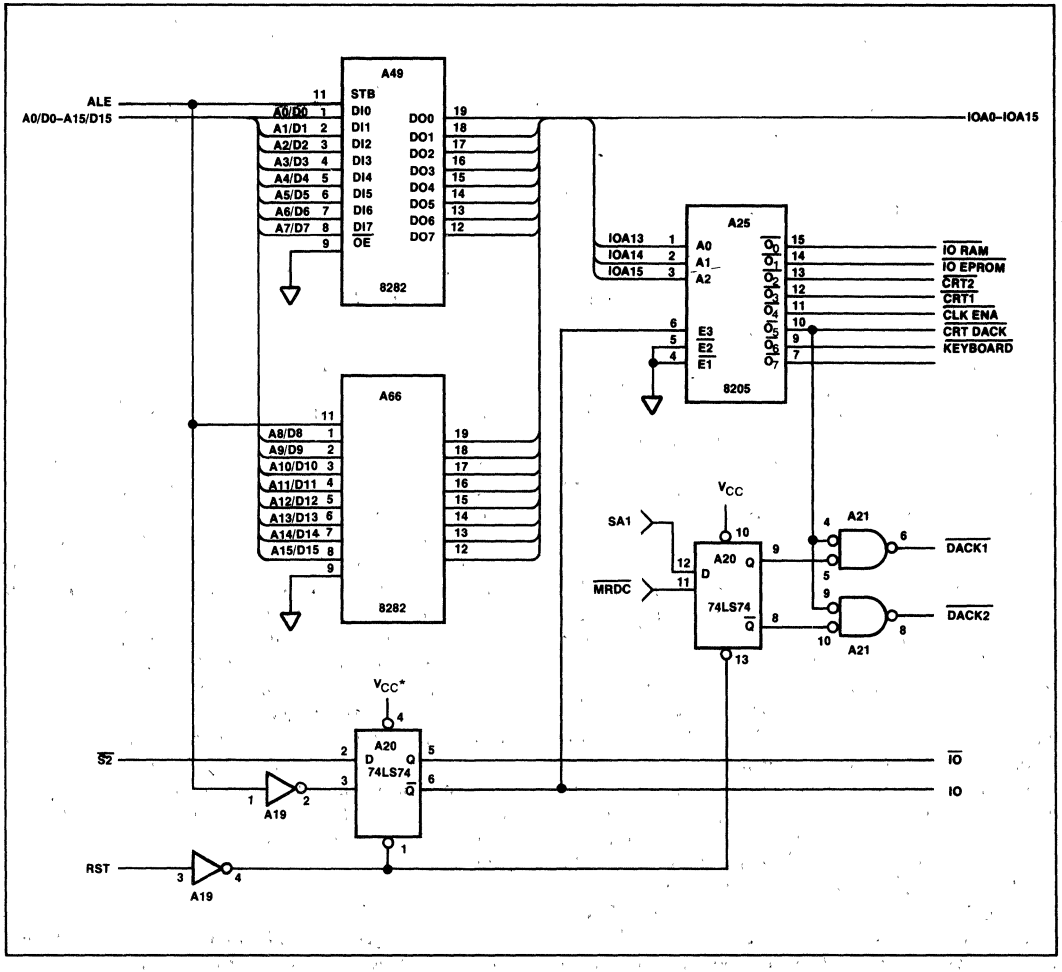


Figure 27. Address Latches, Decoders, and DACK Generator

Figure 28 shows the bus transceivers used between the 8089 and the I/O bus, and also shows the 2732 EPROMs.

Figure 29 shows the 2K bytes of 2114 static RAM on the I/O bus, which are used as scratch-pad RAM for the 8089.

Figure 30 shows the 8279-5 keyboard controller, and also shows the 8284A clock generator that produces the CLK, RDY, and RST signals for the 8089. For more information on interfacing the 8279-5 to the keyboard (Cherry Electrical Products B70–05AB), refer to the 8279/8279-5 data sheet and application note AP-32, *CRT Terminal Design Using the Intel 8275 and 8279*.

Figure 31 shows the clock generator for the character timing and dot timing. The character clock frequency (C CLK) is 1/8 of the dot clock frequency (D CLK), 10.8 MHz. Also shown in Figure 31 is a 9602 one-shot used to generate the video sync pulses.

Figure 32 shows the CRT Controllers #0 and #1. Bit 6 of Byte 0 determines whether the display character is text or graphic. If Bit 6 is low, the character is a text character, and Byte 1 is used to address the 2732A character generator ROM. Bytes 2 and 3 are ignored. The line count outputs LC0–LC3 of an 8275 (any 8275 can be used, since they are all synchronized) are also applied to the character generator to perform the line select function.

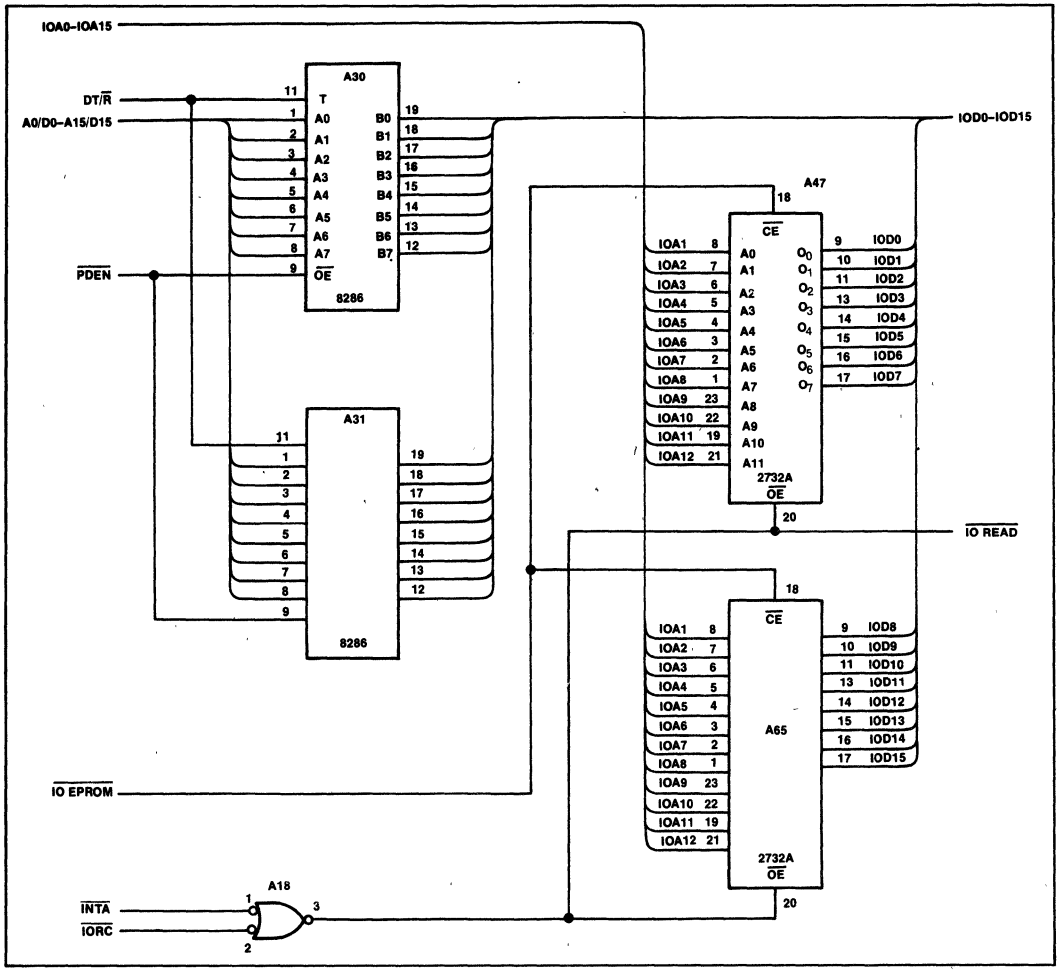


Figure 28. Bus Transceivers and EPROMs on I/O Bus

AFN–02172A

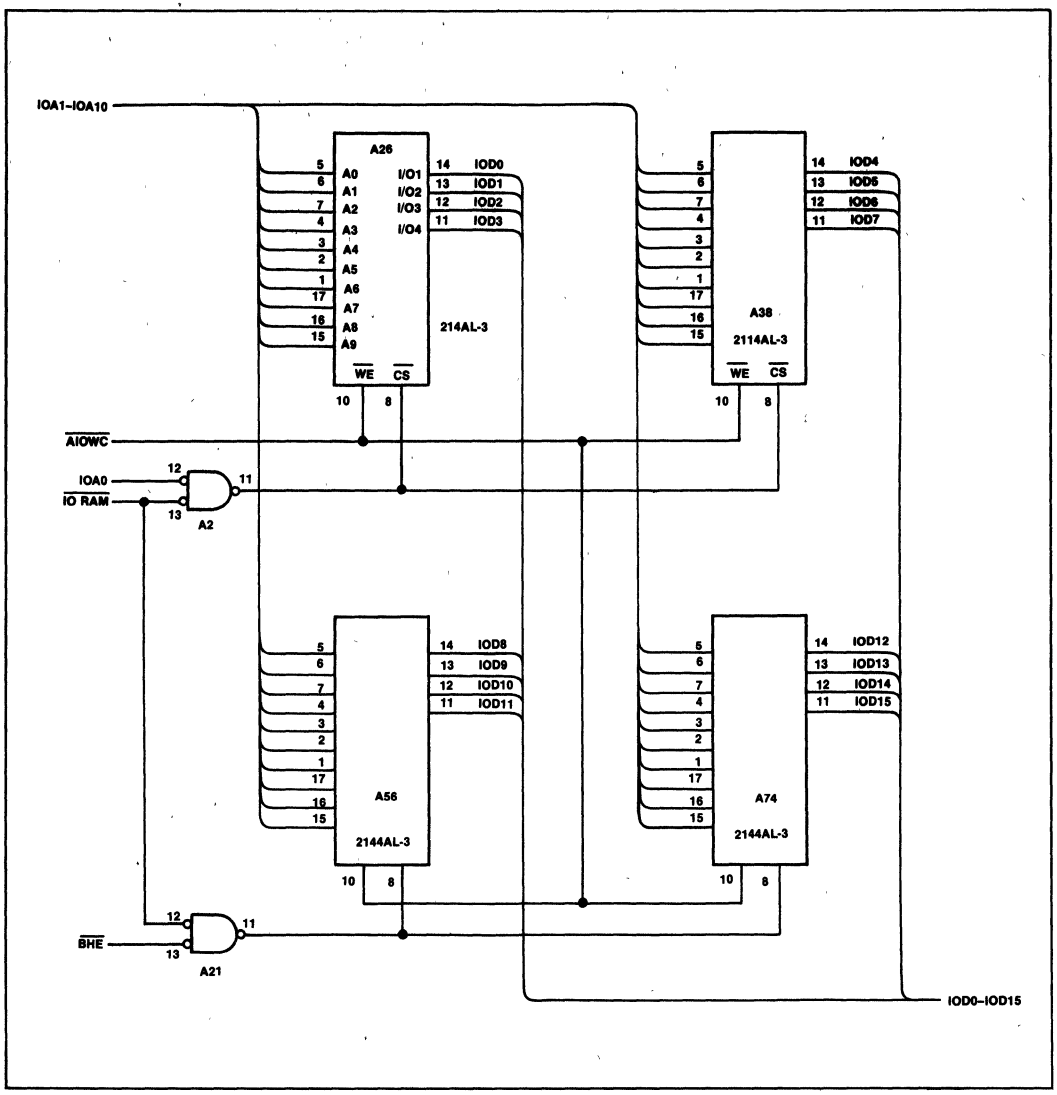


Figure 29. Static RAMs on I/O Bus

AFN–02172A

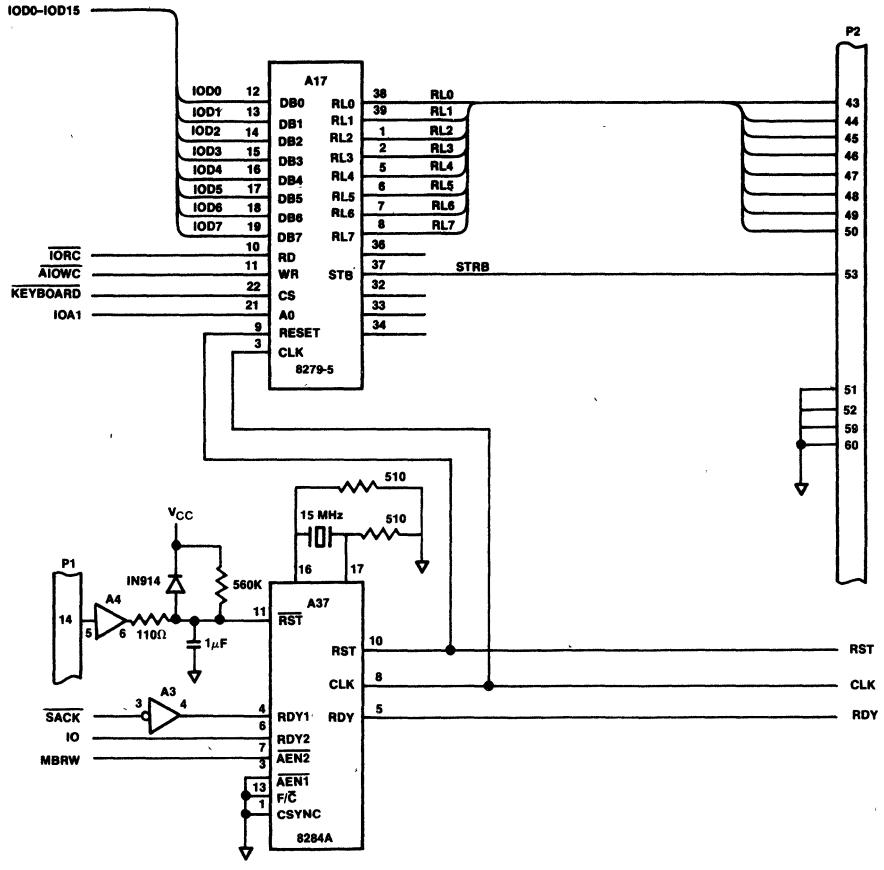


Figure 30. Keyboard Controller and Clock Generator

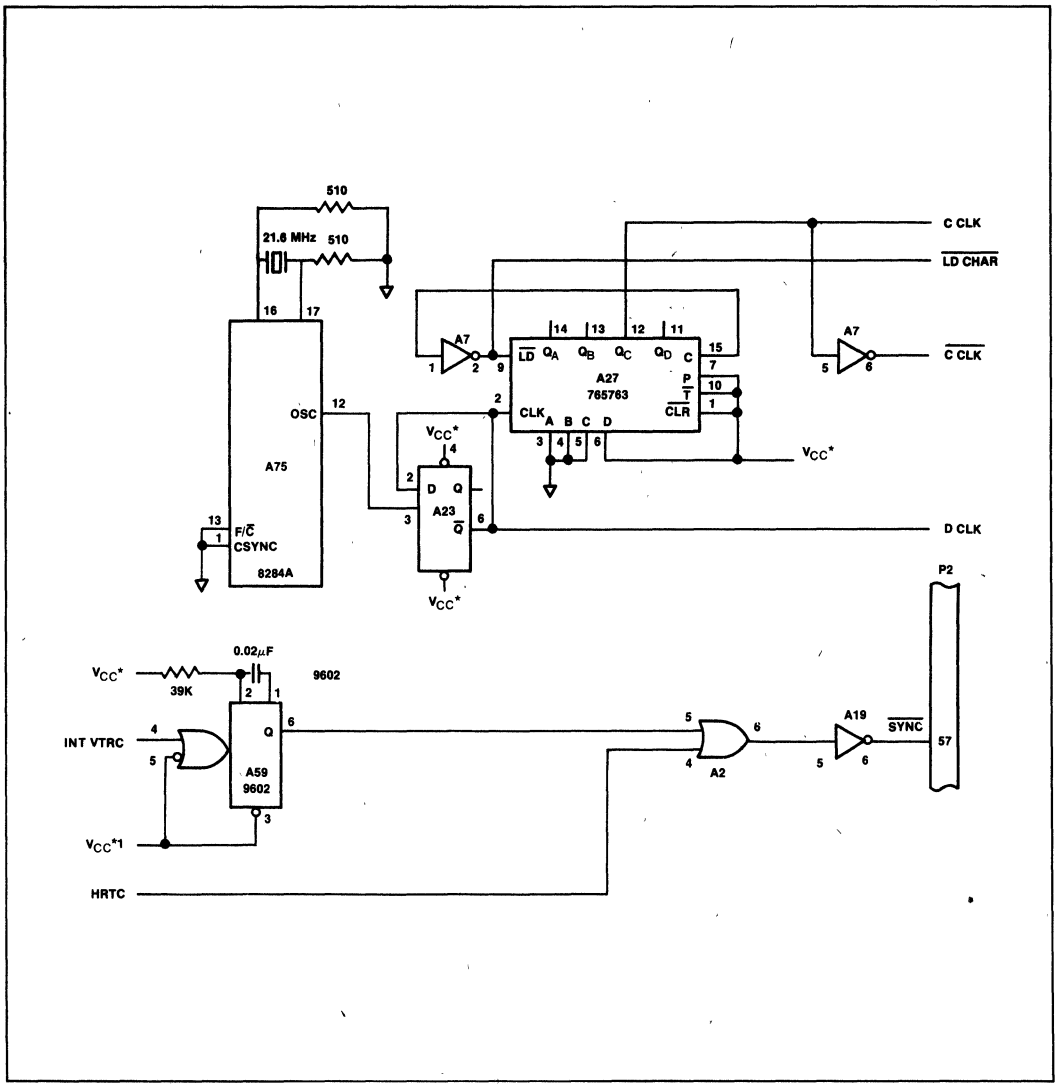


Figure 31. Character Clock Generator and Video Sync Pulse

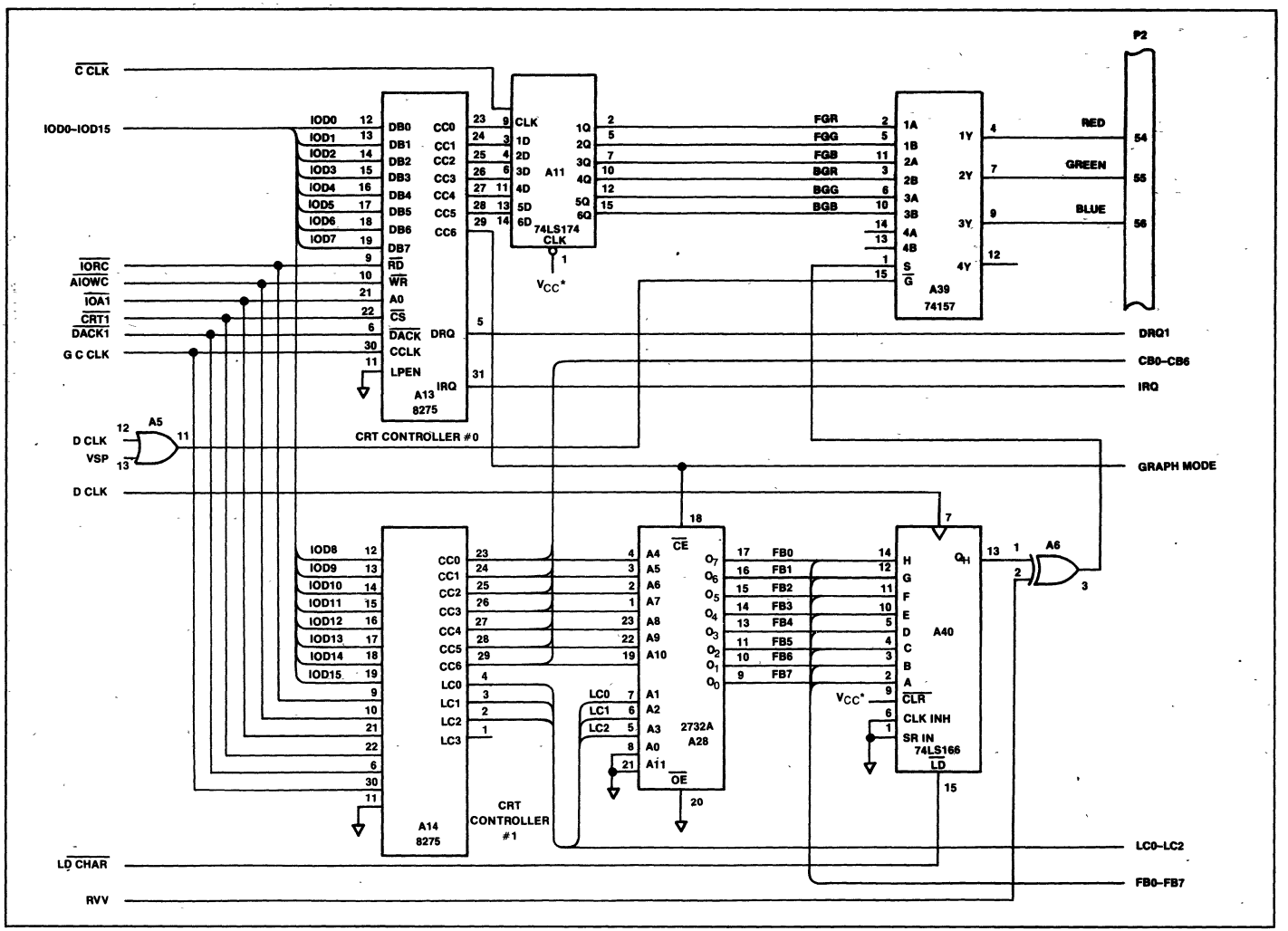


Figure 32. CRT Controllers, Color Multiplexer, and Character Generator

For each character, the foreground and background color bits are output from Byte 0 and latched into the 74LS174, from which they are applied to the input of the 74LS157 multiplexer. Selection between foreground and background is done by the output of the 74LS166 parallel-to-serial converter, which operates from either the text or graphic character generator, as appropriate. The roles of foreground and background color may be reversed by the RVV (reverse video) signal from the 8275, which is exclusive-ORed with this color select output.

Since the RBG (red-blue-green) inputs of the color monitor (Aydin Controls 8039D) are AC coupled, return-to-zero type outputs are needed to pass these signals through the input stages. This is provided by strobing the gate input of the 74LS157 multiplexer with the D CLK (dot clock) signal. By varying the duty cycle of the D CLK, the user can produce many different shades of color. The D CLK signal is ORed with the VSP (video suppress) signal from the 8275, to produce complete video blanking when desired.

Figure 33 shows the CRT Controllers #2 and #3, the decoder for the line select function, and latches for the video control signals. CRT controllers #2 and #3 are operational in graphics mode only. Synchronization of the two pairs of CRT controllers is discussed in the *8089 Display Functions Software* section.

Figure 34 shows the tri-state buffers used to handle the color information within a graphic character. The decoded line count outputs (ROW 0/–ROW 4/) are used to select which buffer is enabled onto the bus. The buffer A36, enabled by the GRAPH MODE signal, is used to "double up" the four graphic cells to produce eight (horizontal) dot inputs to the shift register (Figure 32).

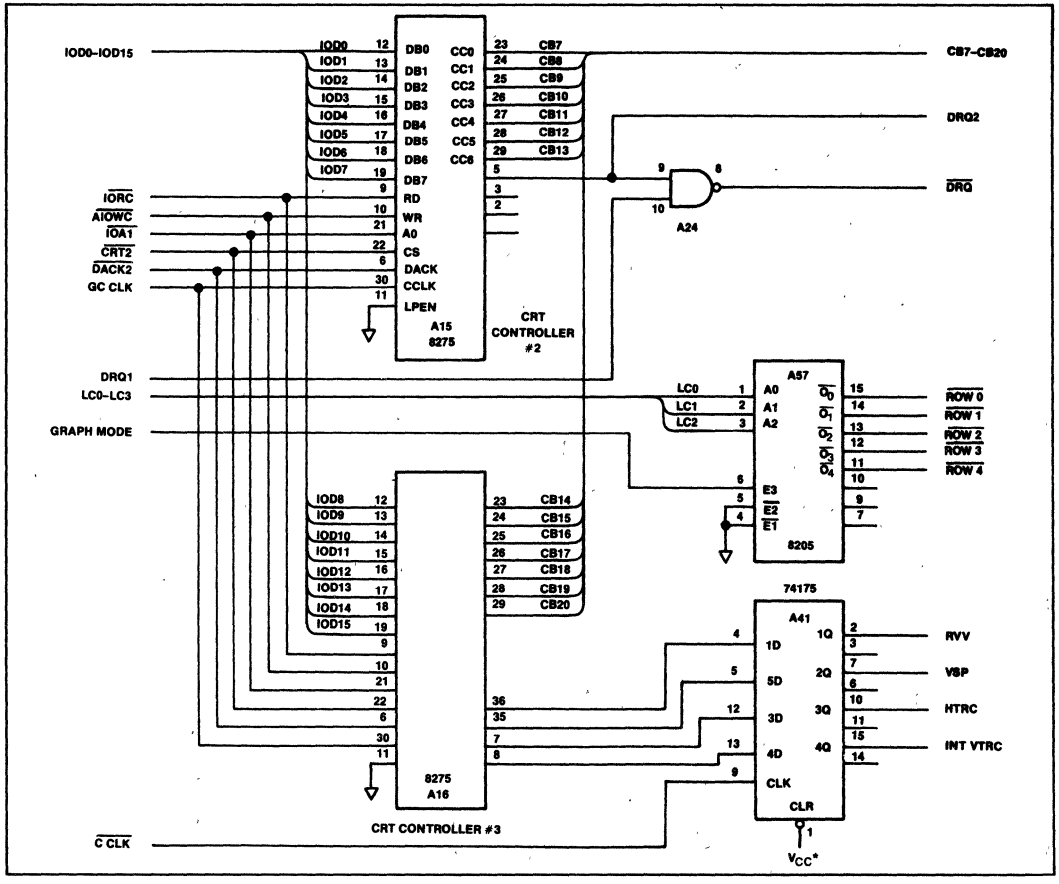


Figure 33. CRT Controllers, Line Decoder, and Video Control Signal Latch

AFN-02172A

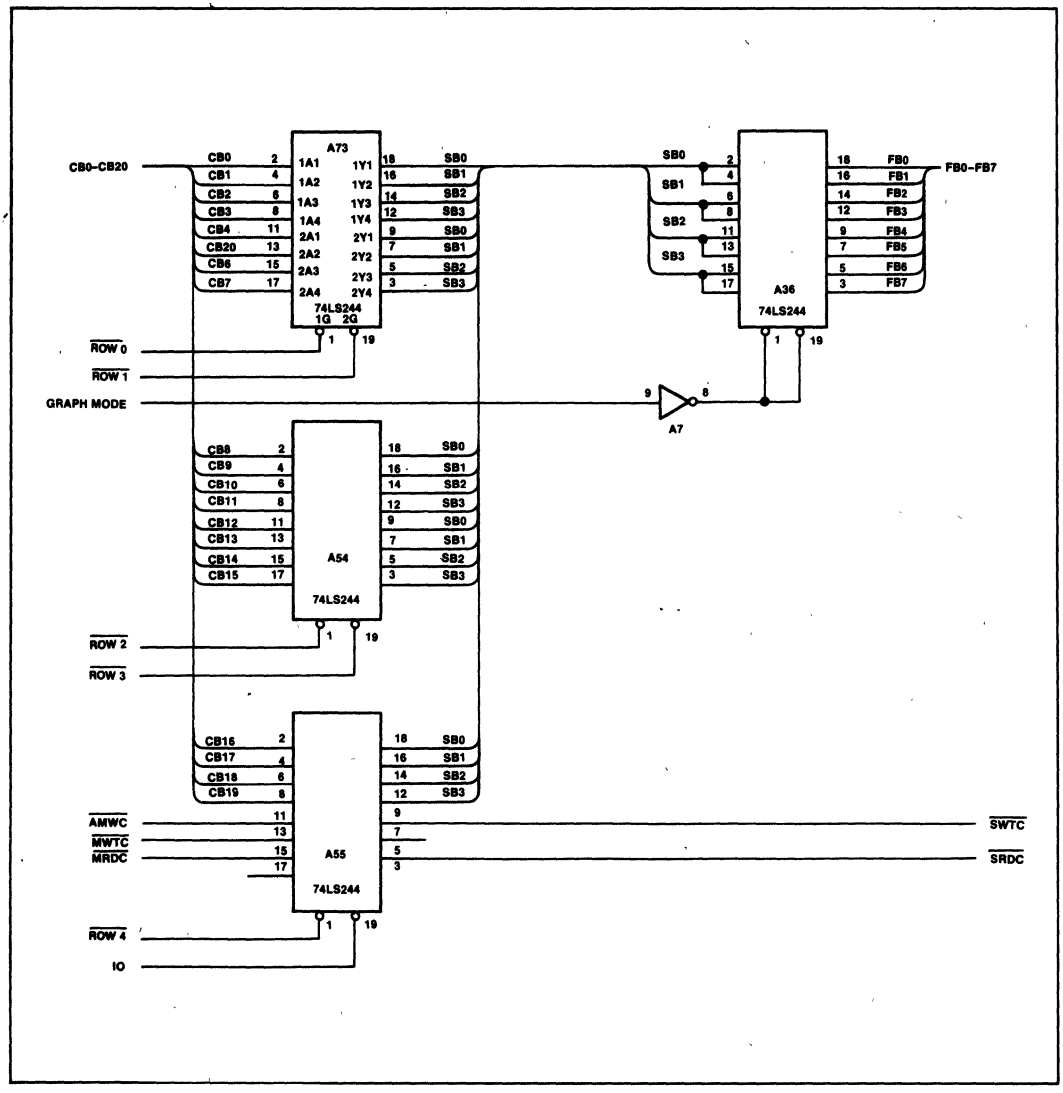


Figure 34. Tri-State Buffers for Graphic Color Information

AFN-02172A

The block diagram in Figure 35 shows how the text characters are processed. The following statements apply to Figure 35:

1. Byte 0, Bit 6 = 0 indicates text mode.
2. The six color signals from CRT Controller #0 (three foreground and three background) are latched and transmitted to the multiplexer.
3. The seven character output signals and the three line count signals from CRT Controller #1 are transmitted to the text character generator.
4. The eight output signals from the text character generator are transmitted to the parallel-to-serial converter.
5. The serial, horizontal dot data is transmitted to the multiplexer and selects foreground (dot data bit = 0) or background (dot data bit = 1) color signals.
6. The red, blue, and green color signals are transmitted to the color monitor.
7. CRT Controllers #2 and #3 are not operational in text mode.

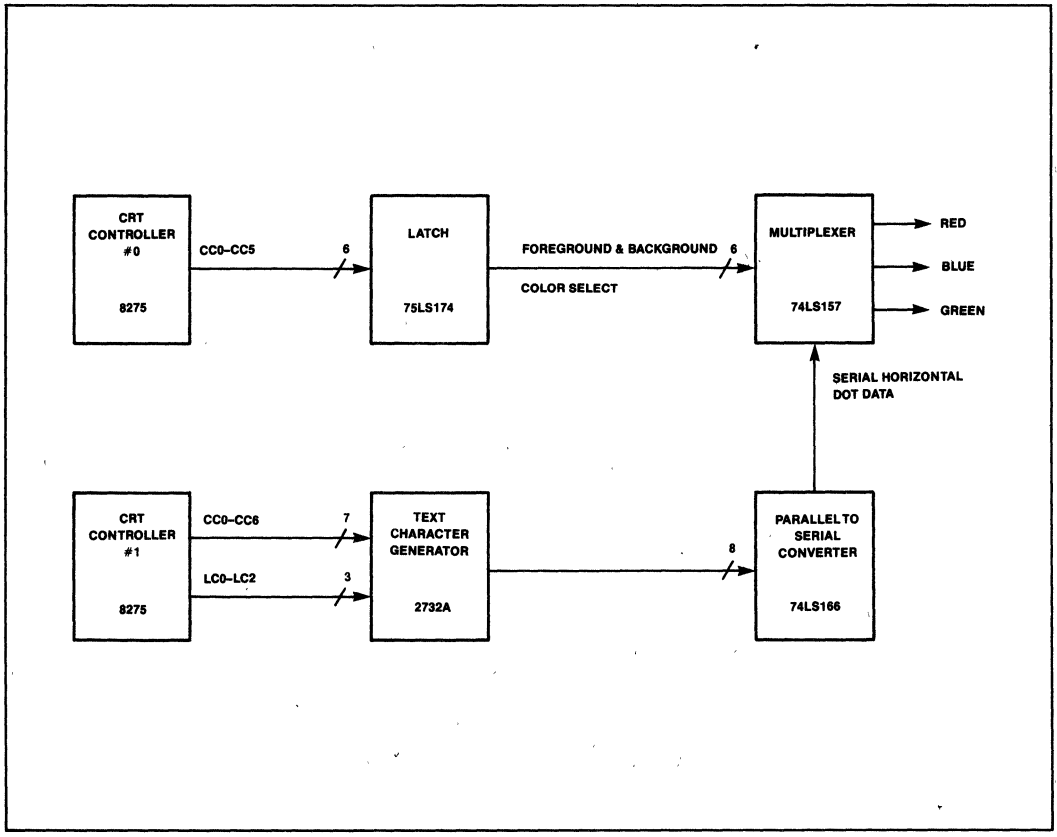


Figure 35. Processing of Text Characters

The block diagram in Figure 36 shows how graphic characters are processed. The following statements apply to Figure 36:

1. Byte 0, Bit 6 = 1 indicates graphic mode.
2. The six color signals from CRT Controller #0 (three foreground and three background) are latched and transmitted to the multiplexer.
3. The three line count signals from CRT Controller #1 are transmitted to a one-of-eight decoder which generates five row select signals (ROW 0–ROW 4).
4. The twenty pixel signals from CRT Controllers #1, #2, and #3 are transmitted to three octal buffers.
5. The four pixel signals of the selected row (based on the row select signals) are transmitted to another octal buffer.
6. The octal buffer converts these four bits to eight bits by duplicating each signal. Thus, output bits 0 and 1 are equal, 2 and 3 are equal, etc.
7. The eight output signals of the octal buffer are transmitted to the parallel-to-serial converter.
8. The serial, horizontal dot data is transmitted to the multiplexer and selects foreground (dot data bit = 0) or background (dot data bit = 1) color signals.
9. The red, blue, and green color signals are transmitted to the color monitor.

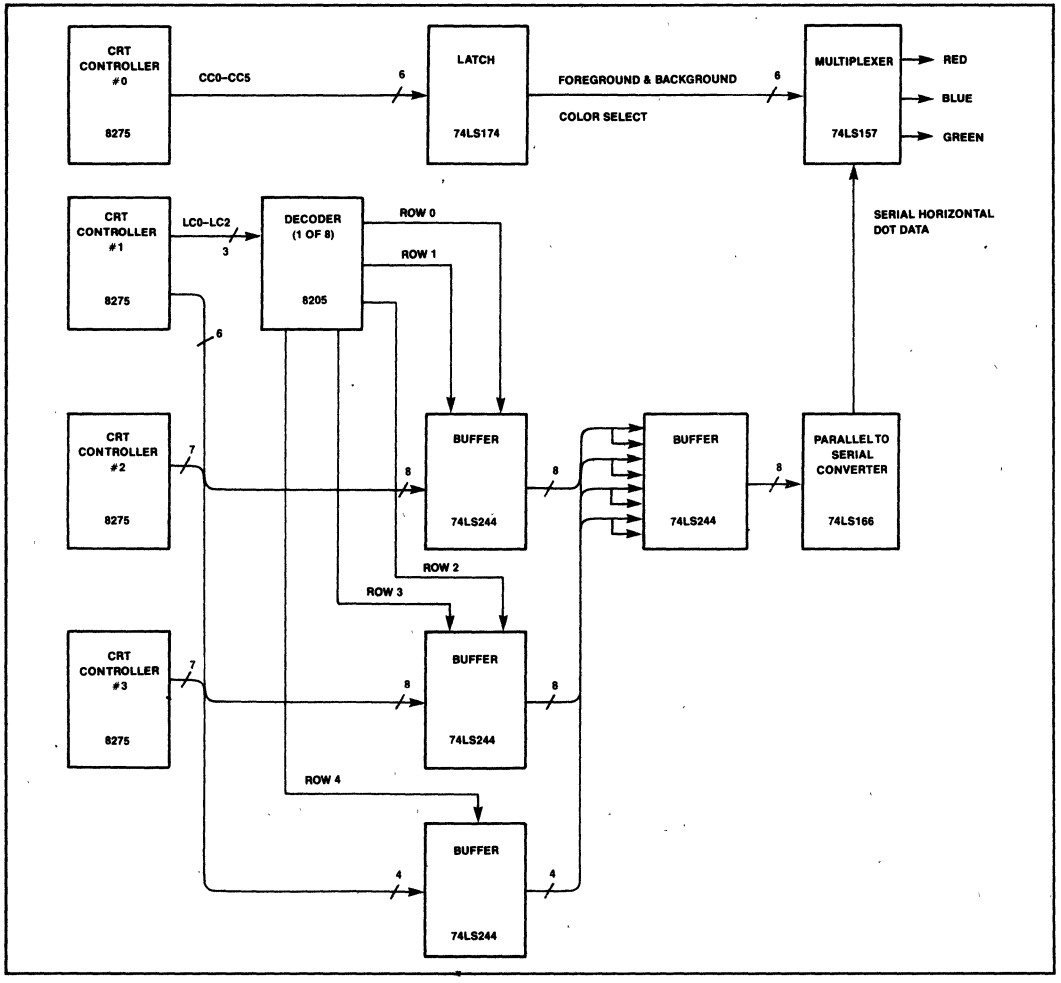


Figure 36. Processing of Graphic Characters

Figure 37 shows the circuit used to synchronize the 8275s, and also the circuit used to generate the DRQF signal. As mentioned earlier (see Figure 20), if the 8089 were to wait for a subsequent DRQ signal from the 8275s, some clock cycles would be allocated to idle clocks, and the DMA transfer would become less efficient. To preclude this, the circuit shown in Figure 37 generates a surrogate (early) DRQ signal, DRQF, using a one-shot triggered by the trailing edge of DRQ (DRQ 1 AND DRQ 2). The one-shot times out prior to the rising edge of CLK in T4 of the DMA's store bus cycle.

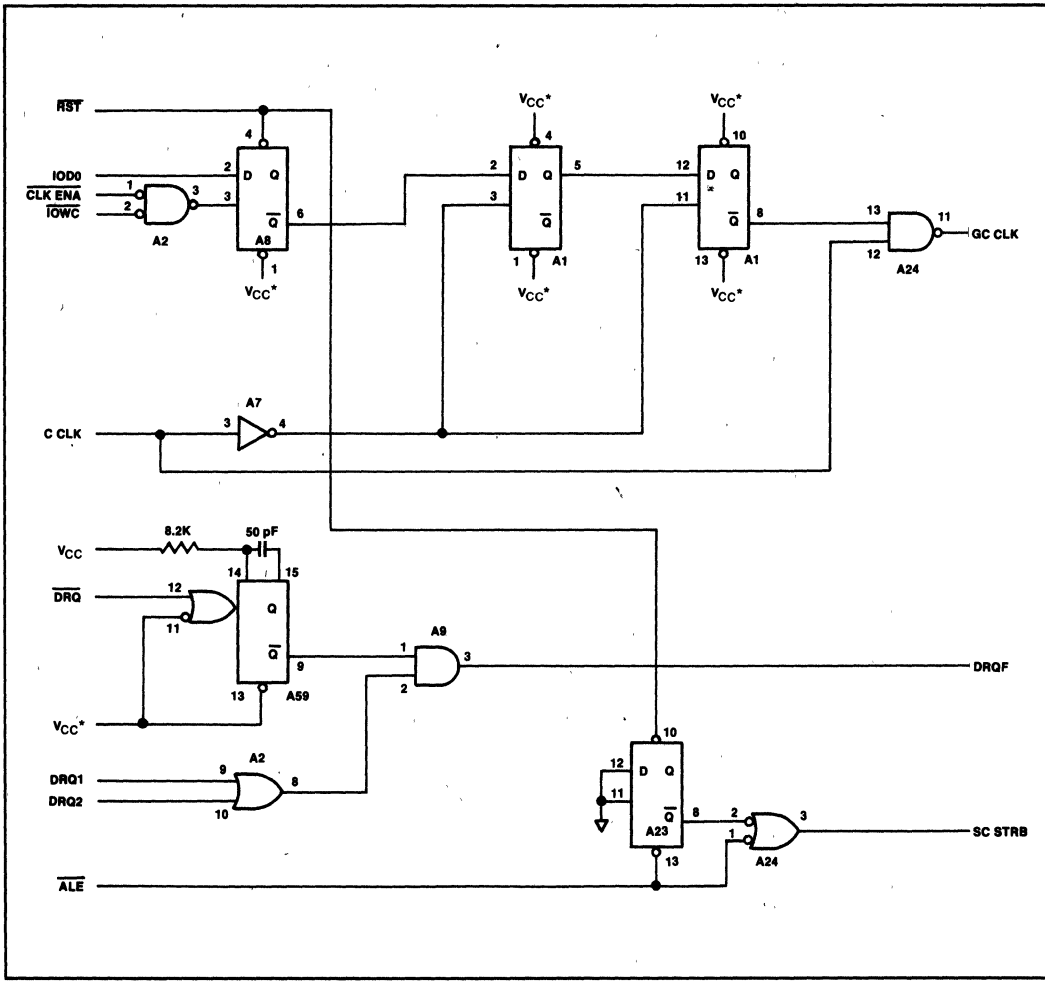


Figure 37. Circuits to Synchronize CRT Controllers and Generate DRQF

Figure 38 shows the relationship between the individual DRQ signals from the 8275s and the DRQF signal that is sent to the 8089. DRQ 1 is the data request representing the 8275s #0 and #1, while DRQ 2 similarly represents the 8275s #2 and #3. The DACK 1/ and DACK 2/ signals (along with AIOWC/) are used to deactivate DRQ 1 and DRQ 2, respectively.

Figure 39 shows the multiplexer used to control writing of data to the dual-port RAM. When IO and SWTC/ are both low, the 8089 data is gated to the dual-port RAM. When BDSEL/ and SWTC/ are both low, the 8086 data is gated to the dual-port RAM. BDSEL/ may be active only when the 8089 is in the I/O space. Note that the address range for the dual-port RAM is F8000–FFFFF as seen by the 8089, and F0000–F7FFF as seen by the 8086.

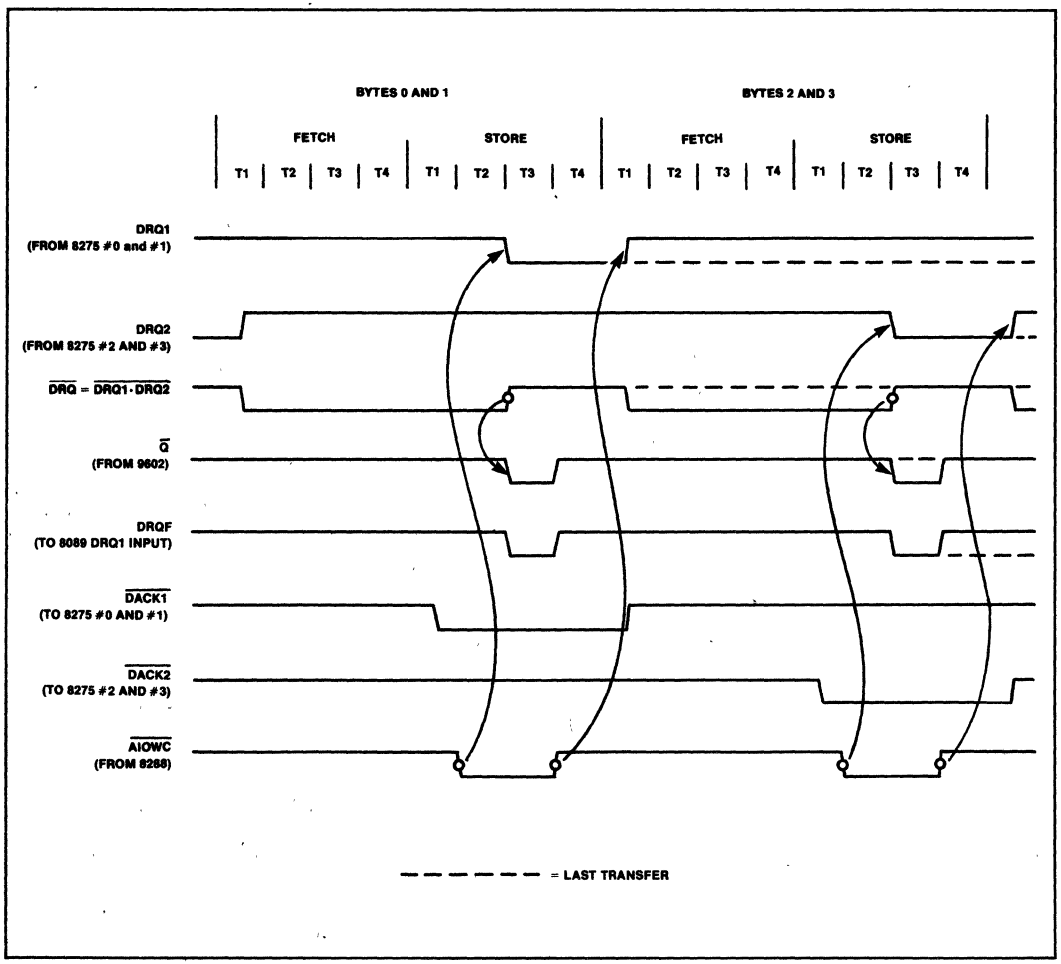


Figure 38. Derivation of DRQF Signal

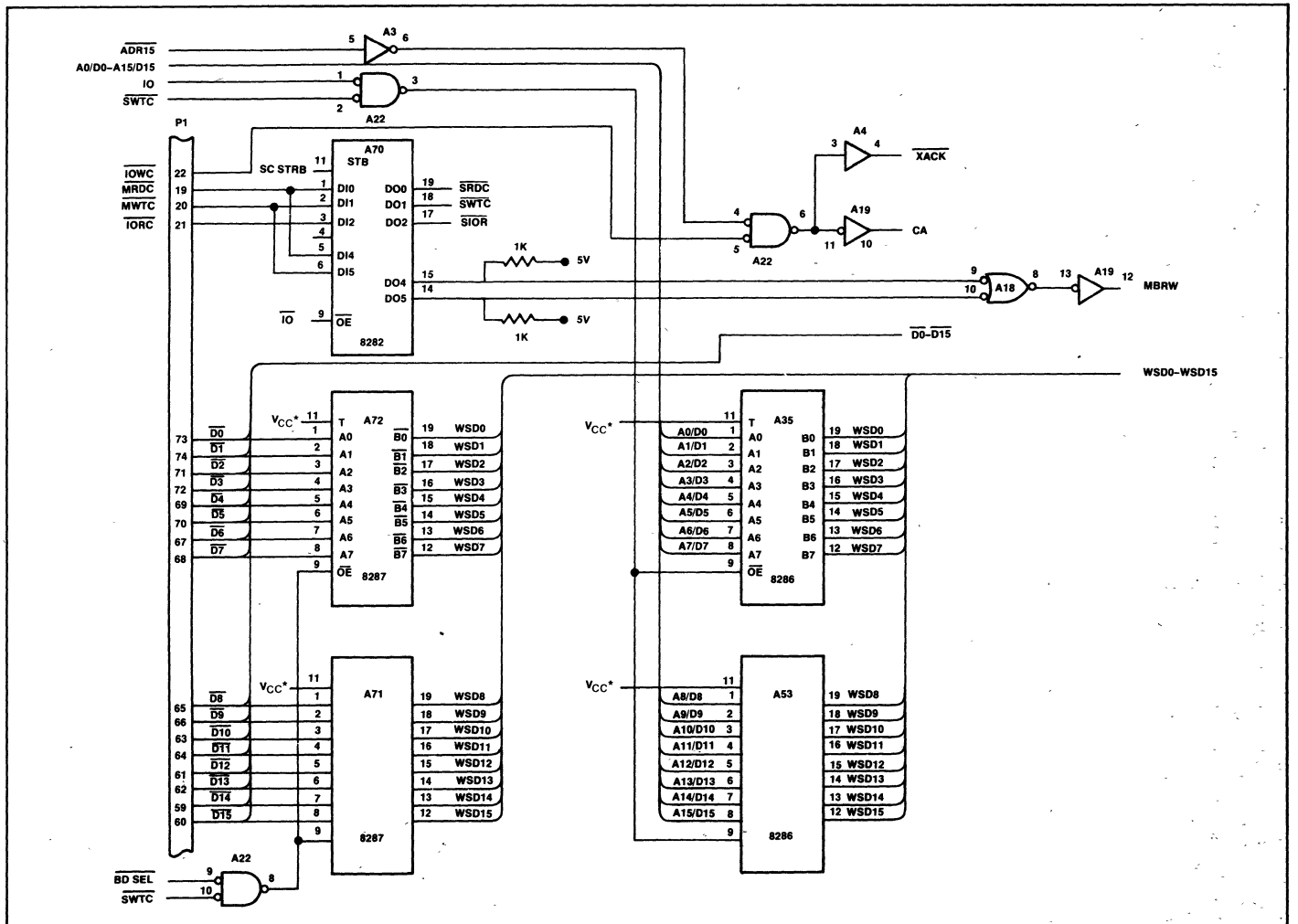


Figure 39. Multiplexer for Writing to Dual-Port RAM

Figure 40 shows the demultiplexer used to control reading of data from the dual-port RAM. The internal transfer acknowledge (SACK/) signal from the dynamic RAM controller latches this data. If MRDC/ is active, the data is then gated to the 8089. If BD ENA/ is active, the data is gated to the Multibus for transmission to the 8086.

Figure 41 shows the multiplexer for the address inputs to the dual-port RAM. If the IO signal is high, the address on the Multibus is gated into the dual-port RAM. If IO is low, the address from the 8089 is gated into the dual-port RAM.

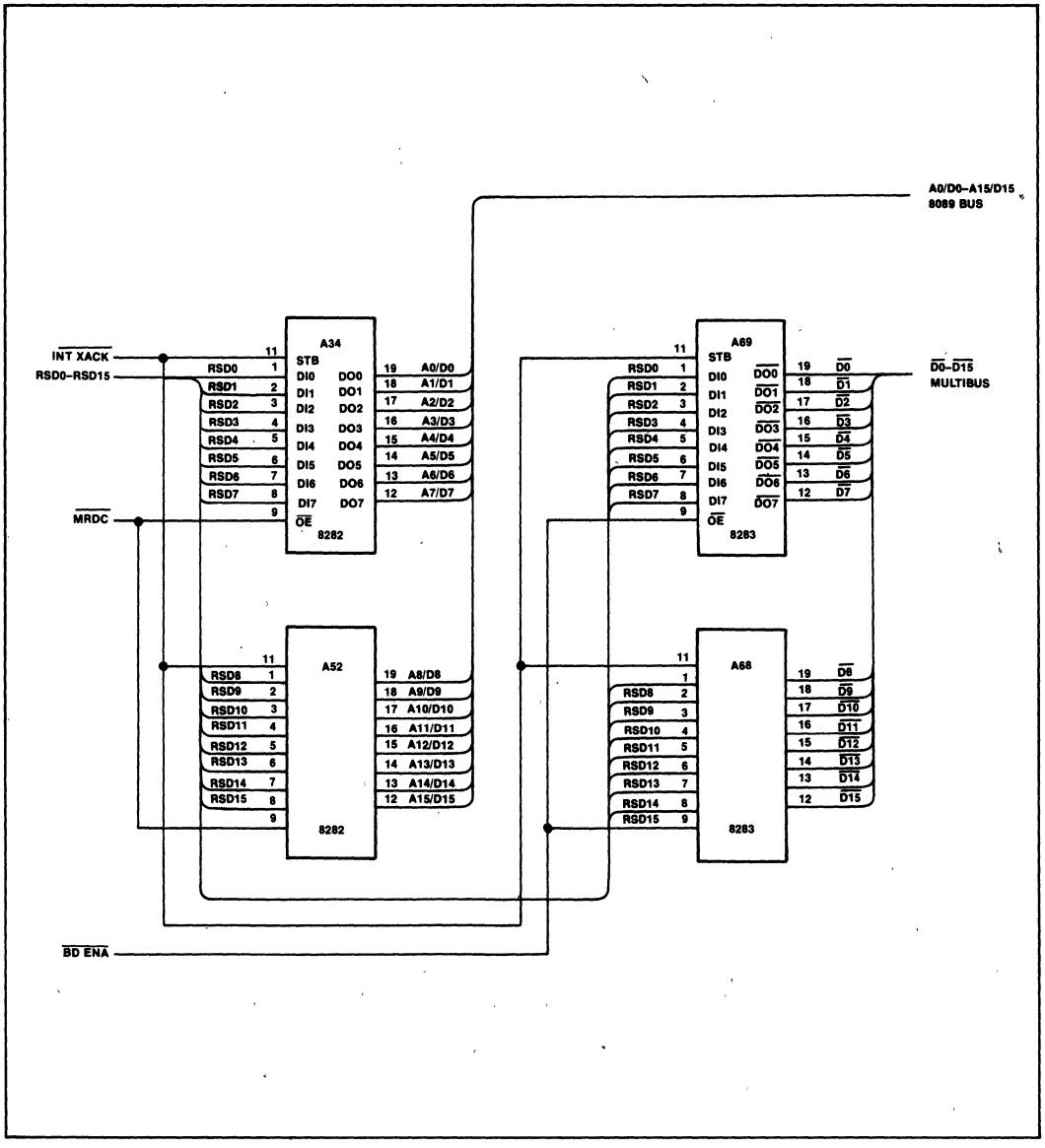


Figure 40. Demultiplexer for Reading from Dual-Port RAM

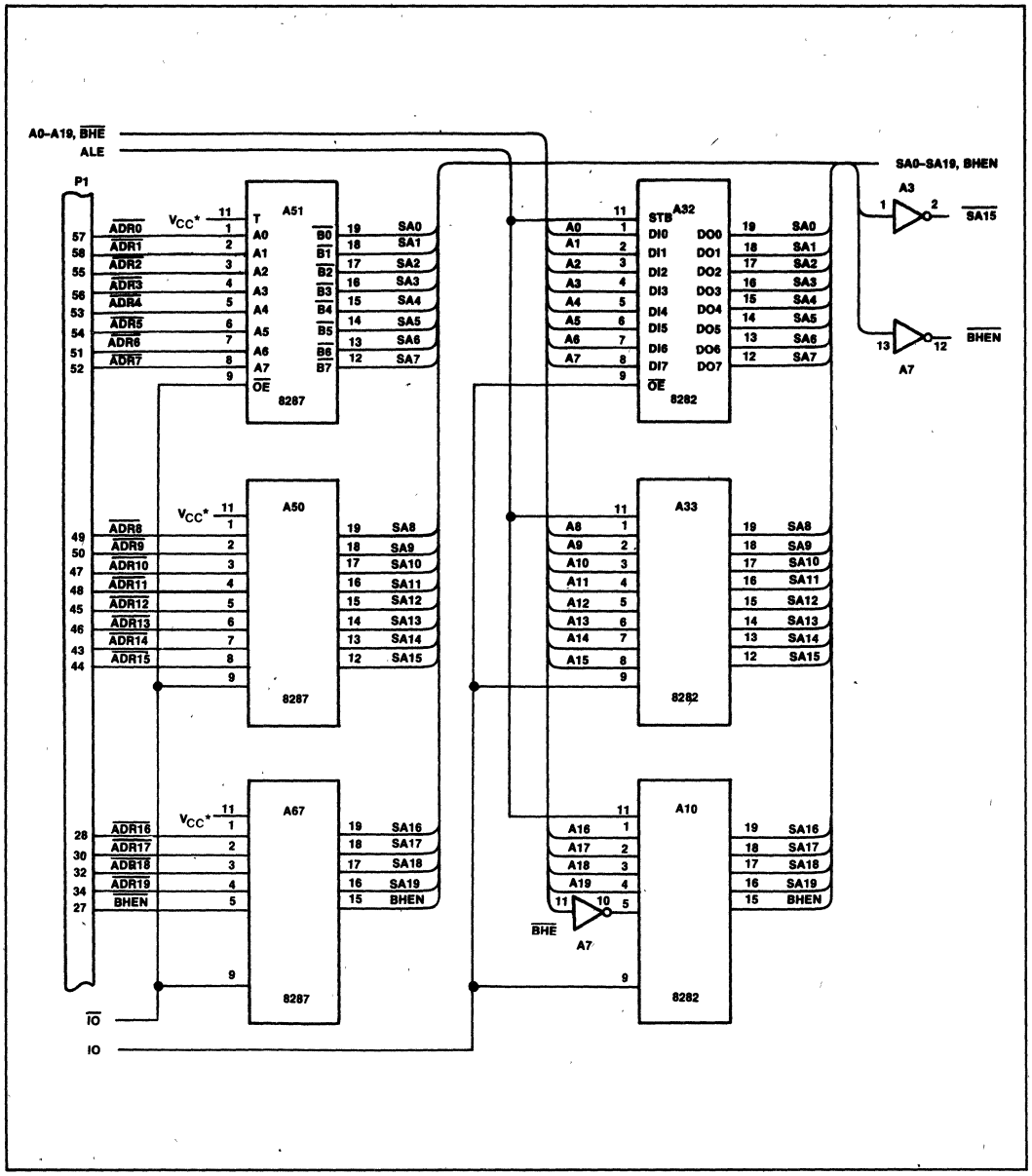


Figure 41. Multiplexer for Address Inputs to Dual-Port RAM

Figure 42 shows the 8202 dynamic RAM controller. The inputs SA0–SA19 come from the multiplexer shown in Figure 41. The dynamic RAM controller generates the control signals (shown at the right of the page) for operating the dynamic RAM.

Figures 43 and 44 show the dynamic RAM itself.

8089 Display Functions Software

The 8089 display functions software consists of a single program which is executed by the 8089 on a continuous basis. This program performs the following functions:

Initialization for the 8089 itself and for the CRT controllers and the keyboard controller.

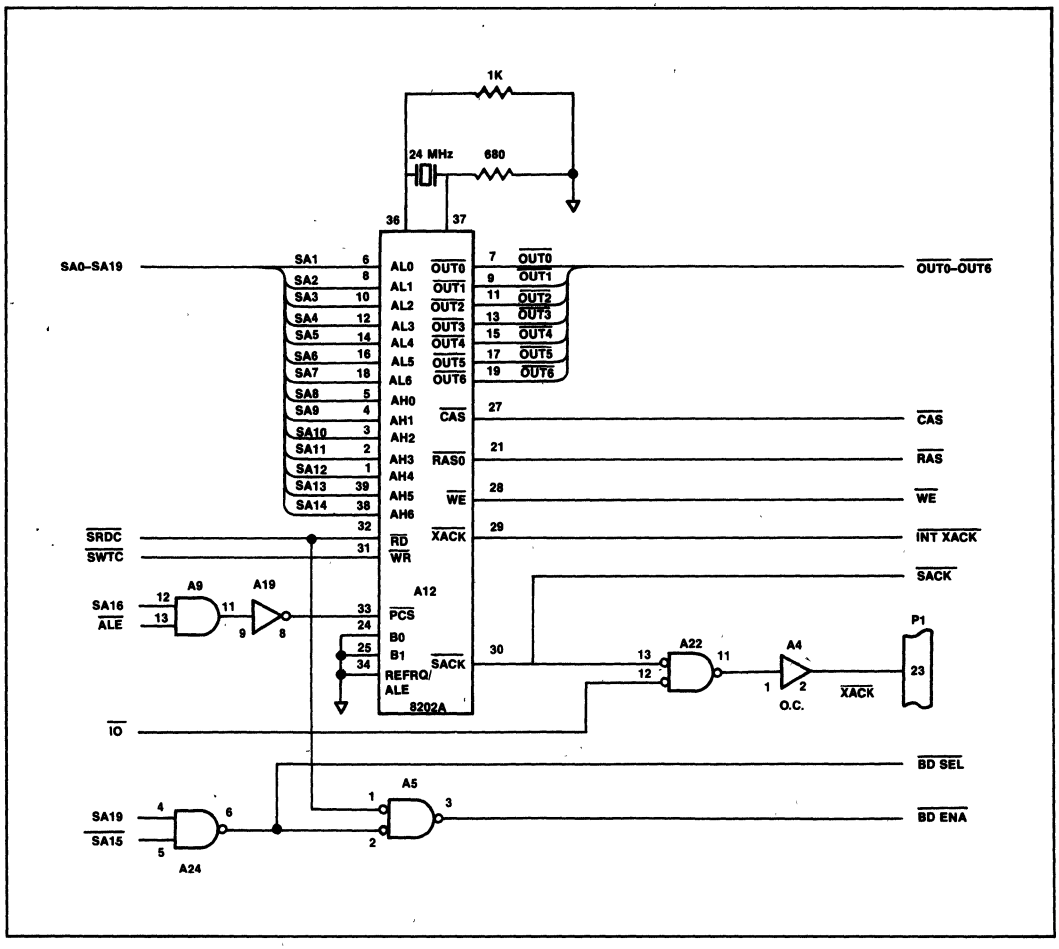


Figure 42. Dynamic RAM Controller

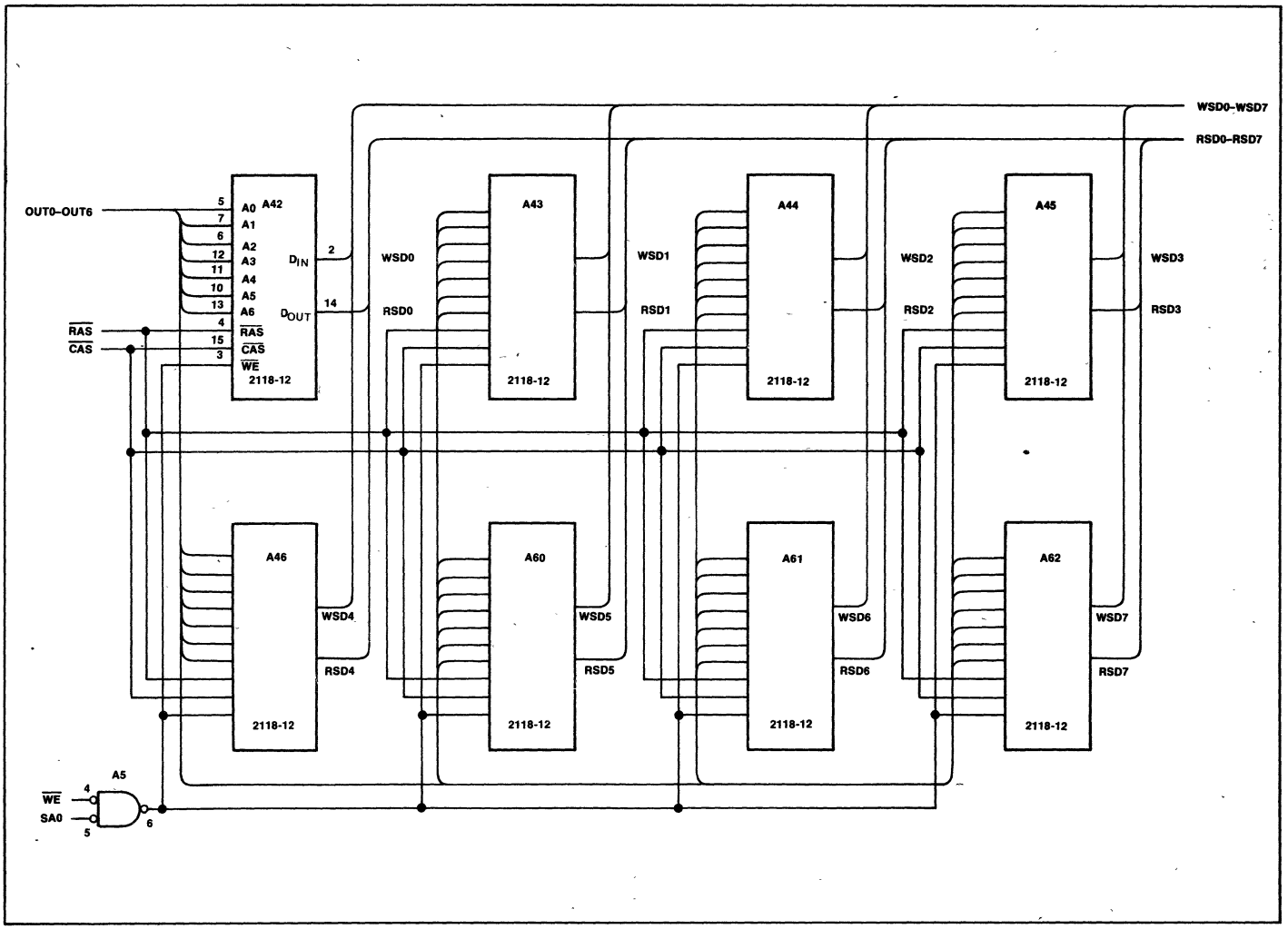


Figure 43. Dynamic RAM (Low Data Byte)

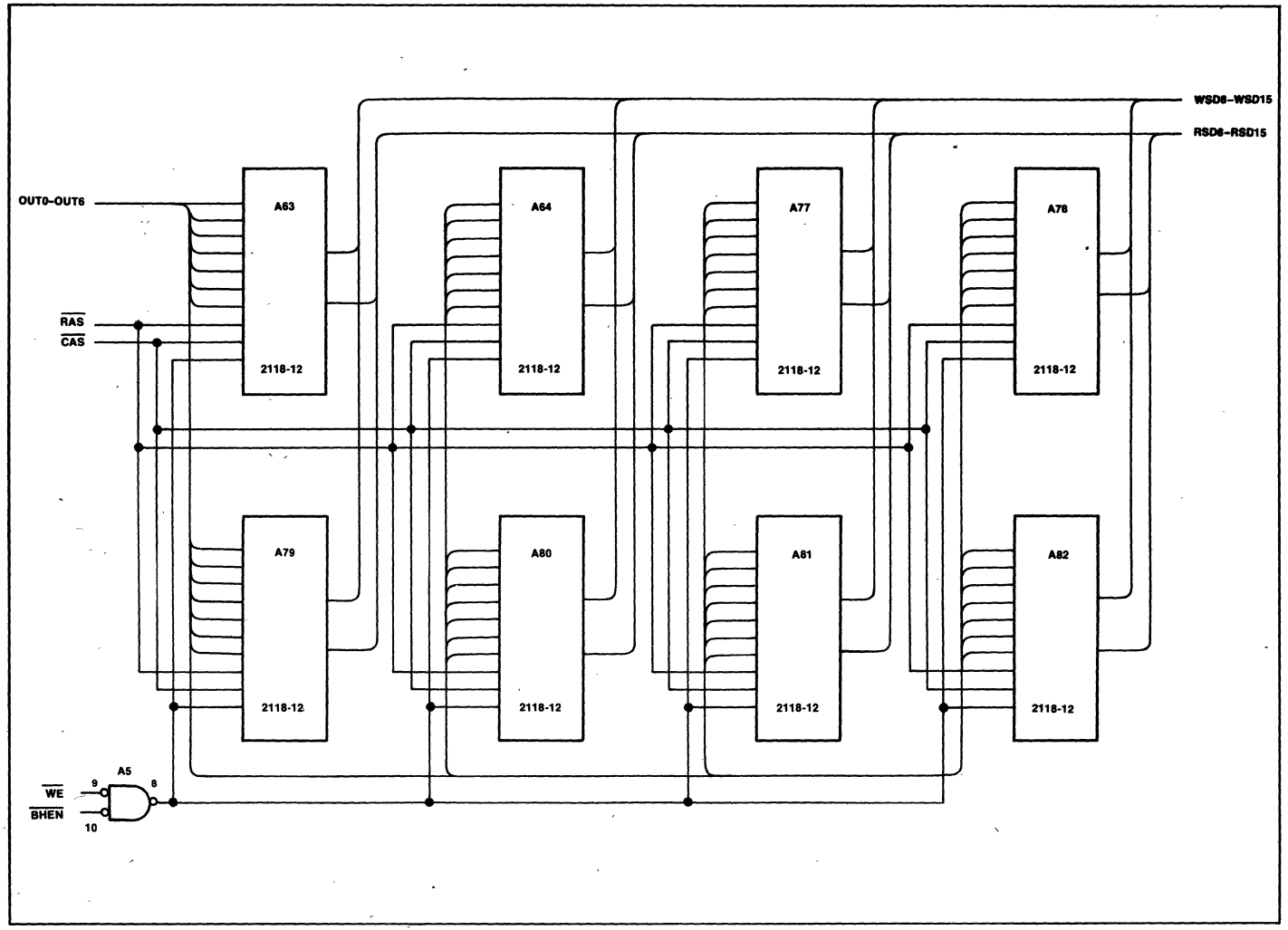


Figure 44. Dynamic RAM (High Data Byte)

The transfer instruction which causes the DMA transfer of the CRT refresh data to begin.

Polling routines for the keyboard and the command buffer.

Figure 45 is a simplified flowchart showing the relationships among these three main functions. The program begins upon receipt of the second CA (channel attention) following an IOP reset. After the initialization processes have been completed, the program loops continuously, alternating between DMA transfer and polling processes. There are 48 rows of characters on the screen. The polling processes are carried out during the vertical retrace time, which is the equivalent of 2 rows. Thus, it is easy to see that the DMA process uses up 96% of the 8089's time, leaving 4% for the polling processes.

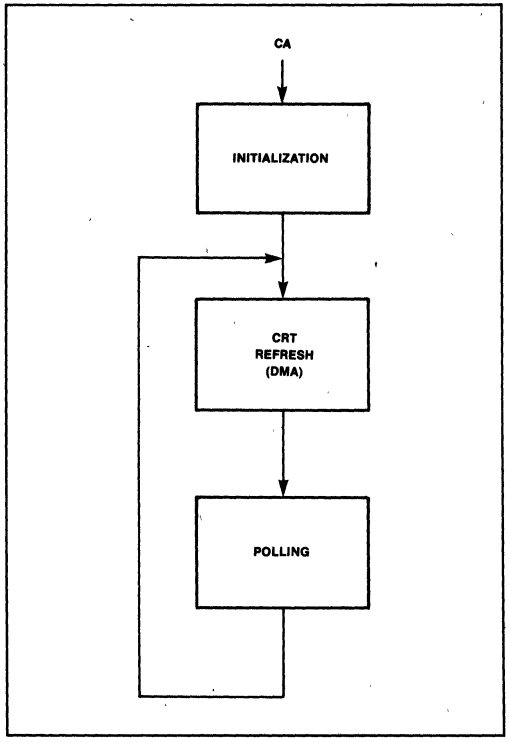


Figure 45. Channel Program Simplified Flowchart

As mentioned earlier, the channel program is stored in the 2732A EPROMs on the I/O bus. Figure 23 (above) shows the address assignments for devices on the I/O bus. The 2732As occupy addresses 2000–3FFF. The 8089 also uses a scratch-pad static RAM (2K bytes at addresses 0000–07FF). The CRT controllers are accessed by using addresses 4000 and 6000 on the I/O bus. Address 6000 is "CRT Controller 1" and actually refers to the first pair of 8275s. Address 4000 is "CRT Controller 2," the second pair of 8275s. Address 8000 is a clock enable address. Write commands to this address enable or disable the GC clock, which is the character clock for the 8275s. Address A000 is decoded to produce the DACK signal for the 8275s. Address C000 is the address of the keyboard controller.

The exact manner in which the channel program executes depends on the flag settings and parameter values in the parameter block.

Appendix A is a flowchart for the complete channel program. Appendix B is the corresponding ASM-89 assembly language listing. In the paragraphs to follow, a general overview of the channel program is given. The reader may refer to the flowchart and listing if a more detailed description is desired.

The first CA after IOP reset causes the 8089 to fetch the system configuration pointer (SCP) and system configuration block (SCB) from dual-port memory. These blocks contain certain very basic system-level information for the 8089, as explained above under *Overview of the 8089*.

The next CA causes the channel program to begin execution (at the point marked START on the flowchart). The initialization portion of the channel program consists of the following operations:

Start and initialize the 8275 CRT controllers.

Initialize the 8279 keyboard controller.

Initialize the dual-port variables (parameter block).

Synchronize the 8275 CRT controllers.

To initialize and synchronize the 8275s, the channel program performs the following operations:

Enable the GC CLK to the 8275s by writing 01H to I/O port address 8000H.

Send the Reset command to the 8275s, followed by the four screen format parameters (all commands sent to the 8275s are sent first to the pair of 8275s at address 6000H and then repeated for the second pair of 8275s at address 4000H).

Send the Preset Counters command to the 8275s.

Disable the GC CLK by writing 00H to address 8000H.

Send the Start Display command to the 8275s.

Enable the GC CLK again by writing 01H to address 8000H. The 8275s are now initialized and synchronized.

After the initializations have been completed, the channel program enters its main loop. The 8089 channel control register is loaded to specify the following DMA conditions:

Data transfer from memory to I/O port.

Destination-synchronized transfer.

GA register pointing to data source.

Termination on external event.

Termination offset = 0.

The source for the DMA transfer (display page 0 or 1) is then selected according to the value of DSPLY_-PG_PTR (the display page pointer initialized by the host CPU) in the parameter block. The CRT character clock is then started and the DMA transfer begins. When the entire screen has been refreshed, the 8275s activate the 8089's EXT input.

The 8089 then executes the SINTR instruction, which causes an interrupt to be sent to the 8086 (SINTR-1 line on the Multibus), to notify the 8086 that the page transfer has been completed. The 8089 then reads the CRT controller status registers which causes the IRQ signal (from the 8275s to the 8089) to be reset.

The channel program then begins the polling process which checks for ASCII commands from the 8086 (in the command buffer) and also for key depressions at the keyboard. In addition to the alphanumeric characters, the channel program recognizes the following control characters:

Character	Code	Description
CNTRL-A	01	Monitor Inhibit
CNTRL-B	02	Monitor Uninhibit
CNTRL-C	03	EEPROM Inhibit
CNTRL-D	04	EEPROM Uninhibit
CNTRL-E	05	Turn on EEPROM Buffer
CNTRL-F	06	Display Page 0
CNTRL-G	07	Display Page 1
CNTRL-H	08	Backspace
CNTRL-I	09	TAB (Every 8 Characters)
CNTRL-J	0A	Linefeed
CNTRL-K	0B	EEPROM Buffer Off
CNTRL-L	0C	Erase Page
CNTRL-M	0D	Carriage Return
CNTRL-N	0E	Set Background Color
CNTRL-O	0F	Set Foreground Color
CNTRL-P	10	Set Color to Black
CNTRL-Q	11	Set Color to Red
CNTRL-R	12	Set Color to Green
CNTRL-S	13	Set Color to Yellow
CNTRL-T	14	Set Color to Blue
CNTRL-U	15	Set Color to Magenta
CNTRL-V	16	Set Color to Cyan
CNTRL-W	17	Set Color to White
CNTRL-X	18	Abort Line
CNTRL-Y	19	Cursor Right
CNTRL-Z	1A	Cursor Down and Left
CNTRL-^	1E	Cursor Up
CNTRL-/	1C	Cursor Home
CNTRL-DEL	1F	Recall EEPROM Buffer

The first four commands listed above are not recognized if they originate from the physical keyboard, but are recognized if they appear as ASCII commands in the command buffer (that is, if they come from the 8086). Refer to the flowchart (Appendix A) for more details on how the channel program responds to the control characters.

System Performance

The 8089 performs DMA transfers on 921,600 bytes of display data per second. In addition, the 8089 executes a polling routine (described above) during the vertical retrace time (the equivalent of two display rows). The DMA transfer (for a single frame) takes 16.000 milliseconds. This leaves .667 milliseconds for the polling routine to execute, out of a total of 1/60-second CRT refresh period. The program listed in Appendix B takes about 300 microseconds to execute, approximately half the available time. When the polling process is finished, the channel program goes back to DMA mode, and waits for the first DRQ signal from the 8275s.

While the polling routine is executing, the 8089 makes most of its memory accesses in the I/O space, and the dual-port RAM is available to the 8086. When the 8089 returns to the DMA routine, however, it hangs the dual-port RAM while waiting for DRQ. This occurs because the fetch from the dual-port RAM deactivates the IO signal which locks out the 8086 from the dual-port RAM. The IO signal is then not activated until DRQ is received and the data is written to the CRT controllers. This can adversely affect system throughput. Therefore, if it is desired to increase the 8086's access to the dual-port RAM during this period, the user should insert NOPs into the channel program so that it spends more time in the I/O space before returning to DMA.

The 8086 may also access dual-port RAM during the DMA transfer. The dual-port RAM is available to the 8086 on approximately a 50% duty cycle (during the store portion of the DMA transfer cycle). The 8089's store cycle is 800 nanoseconds long (assuming a 5 MHz clock). The 8086's access to dual-port RAM (assuming an 8 MHz clock) takes 500 nanoseconds. However, since the two processors operate asynchronously, the 8086 may begin its access at any point during the 8089's

DMA store cycle. Since the 8086 is the master relative to the dual-port RAM, the ready signal for the 8089's next fetch operation will not be generated until the 8086 is through. Thus, on occasion, the 8089 will have to wait.

Each row of characters requires 256 microseconds of DMA transfer time if no such wait states occur. The repetition rate for rows of characters is 333 microseconds (1/3000 second). Thus, the accumulated wait states due to the 8086's access to dual-port RAM may total 77 microseconds before any underrun occurs. The 8086 programs should be written in such a manner that the added wait states do not total 77 microseconds during any one period of 333 microseconds. The most important single factor in assuring this is to avoid making long burst transfers to or from the dual-port RAM. If an underrun does occur, the entire screen will be blanked until the beginning of the next frame.

Aside from the shared access to dual-port RAM, the two processors may operate concurrently with no coordination necessary. Operations performed by the 8086 (such as numeric processing of display data) may be programmed without regard to the overhead associated with IOP operations.

Conclusions

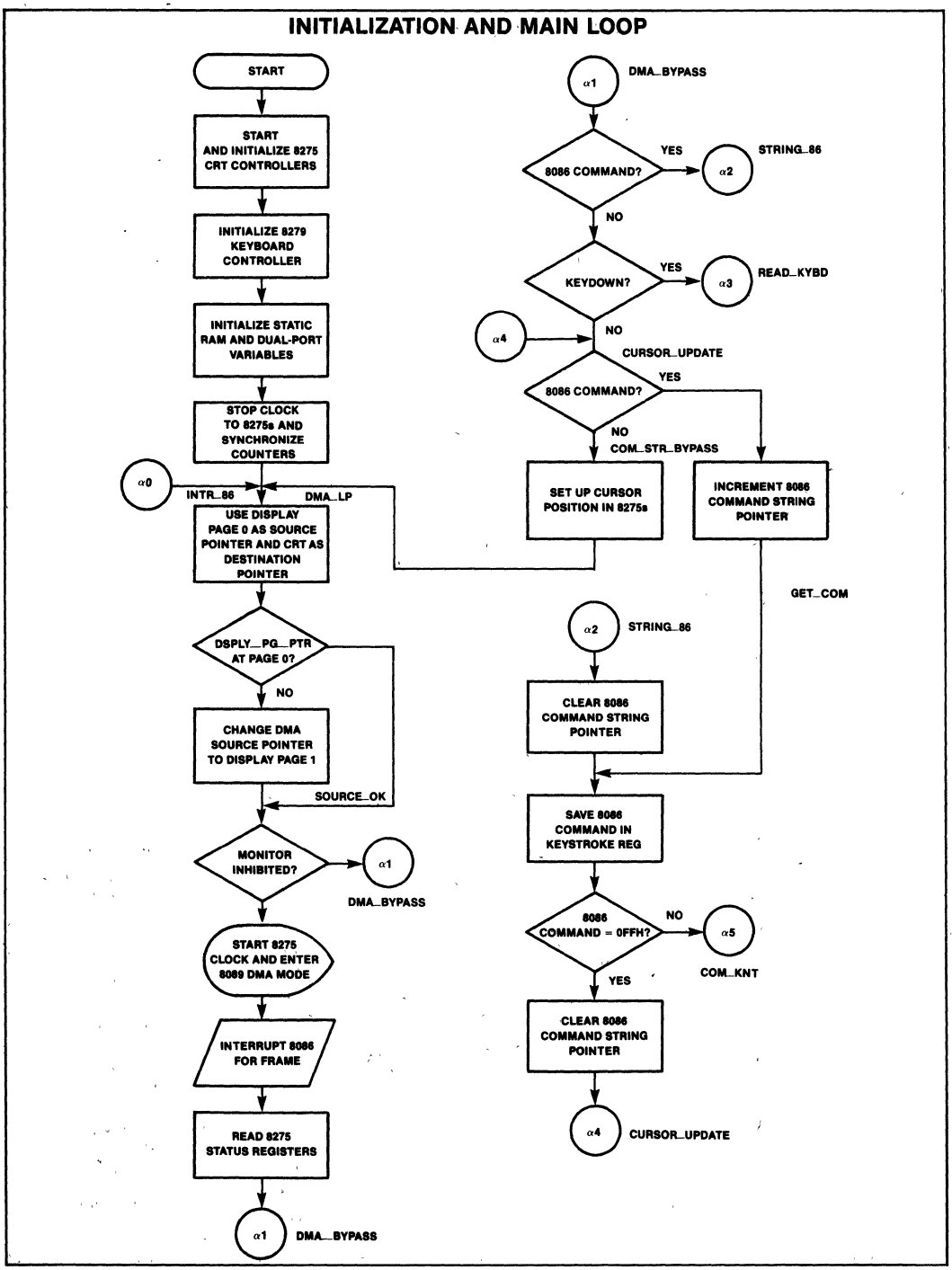
This application note has demonstrated that a high-performance, color-graphic CRT terminal can be conveniently built using the Intel iAPX 86/11 microprocessor system. This system utilizes a high-performance 8086 CPU operating at 8 MHz and an 8089 I/O processor operating at 5 MHz.

In particular, the unique abilities of the 8089 lend themselves to the graphic CRT application by enabling a true multiprocessing approach to be used. The following list summarizes the capabilities used in this specific design:

- High-speed DMA transfers (up to 1.25 megabytes/second) without wait states.
- Capabilities of a CPU and a DMA controller in a single 40-pin package.
- Support of concurrent operation for the system CPU and the I/O processor. Ability to access memory and address devices on both a system bus and a separate I/O bus.
- Flexible, memory-based communications between the I/O processor and the system CPU.
- Capability for 1-megabyte addressing in the system space.
- Capability for 16-bit DMA transfer, with external event termination.
- Support of modular, subsystem development effort due to the simple software interface (memory-based communications, plus channel attention and interrupt signals) and the simple hardware interface (CA, SEL, and SINTR lines).

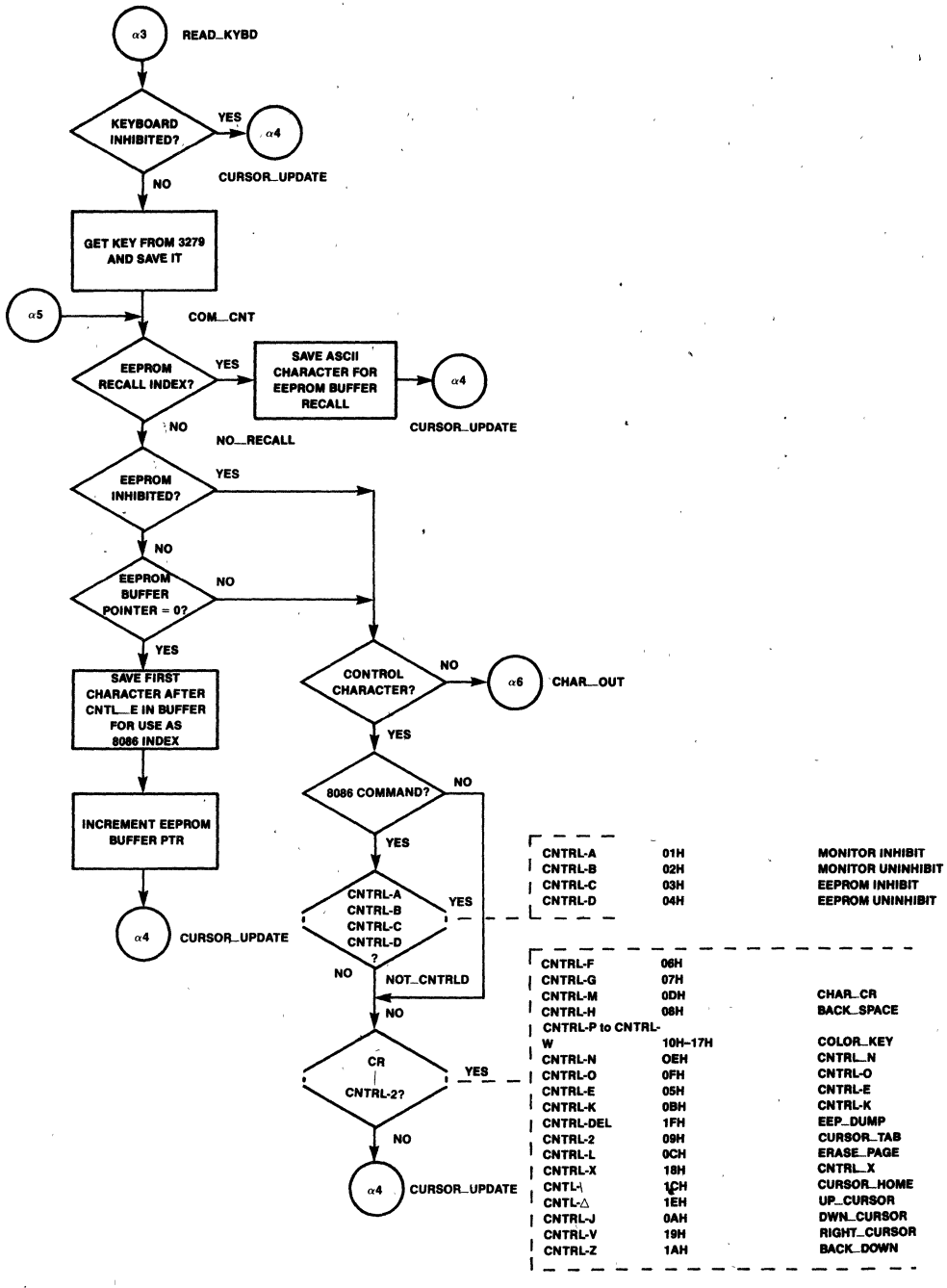
The following 8089 capabilities were not used in the design described in this note, but may be useful in other graphic CRT systems or I/O processing systems:

- Two channels, each of which may execute instructions and perform DMA transfers.
- Bit manipulation instructions.
- Support of both 8-bit and 16-bit bus width in the system space and in the I/O space.
- Enhanced DMA capabilities, including:
  - Translation (e.g., ASCII to EBCDIC code).
  - Termination on masked compare.
  - Word assembly/disassembly (8-bit word to/from 16-bit word).
  - Memory-to-memory or I/O-to-I/O transfer.
  - Synchronization on source, destination, or neither.


INITIALIZATION AND MAIN LOOP
START
START AND INITIALIZE 8275 CRT CONTROLLERS
INITIALIZE 8279 KEYBOARD CONTROLLER
INITIALIZE STATIC RAM AND DUAL-PORT VARIABLES
STOP CLOCK TO 8275s AND SYNCHRONIZE COUNTERS
α0
INTR_86
DMA_LP
USE DISPLAY PAGE 0 AS SOURCE POINTER AND CRT AS DESTINATION POINTER
DSPLY_PG_PTR AT PAGE 0?
NO
CHANGE DMA SOURCE POINTER TO DISPLAY PAGE 1
SOURCE_OK
MONITOR INHIBITED?
α1
DMA_BYPASS
START 8275 CLOCK AND ENTER 8089 DMA MODE
INTERRUPT 8086 FOR FRAME
READ 8275 STATUS REGISTERS
α1
DMA_BYPASS
α1
DMA_BYPASS
8086 COMMAND?
YES
α2
STRING_86
NO
KEYDOWN?
YES
α3
READ_KYBD
α4
NO
CURSOR_UPDATE
8086 COMMAND?
YES
NO
COM_STR_BYPASS
SET UP CURSOR POSITION IN 8275s
INCREMENT 8086 COMMAND STRING POINTER
GET_COM
α2
STRING_86
CLEAR 8086 COMMAND STRING POINTER
SAVE 8086 COMMAND IN KEYSTROKE REG
8086 COMMAND = 0FFH?
NO
α5
COM_KNT
YES
CLEAR 8086 COMMAND STRING POINTER
α4
CURSOR_UPDATE

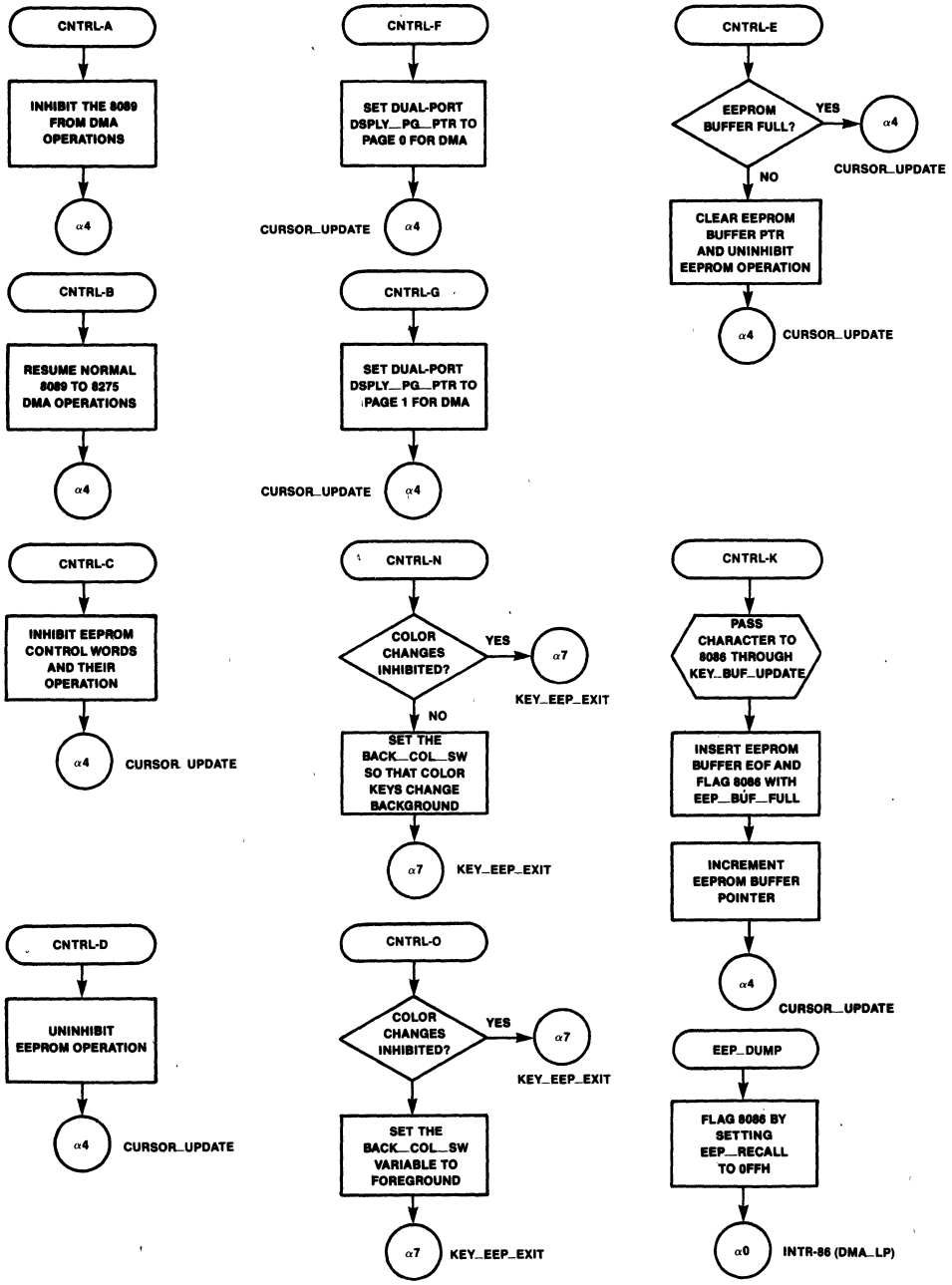
AFN-02172A

KEY AND COMMAND DECODE



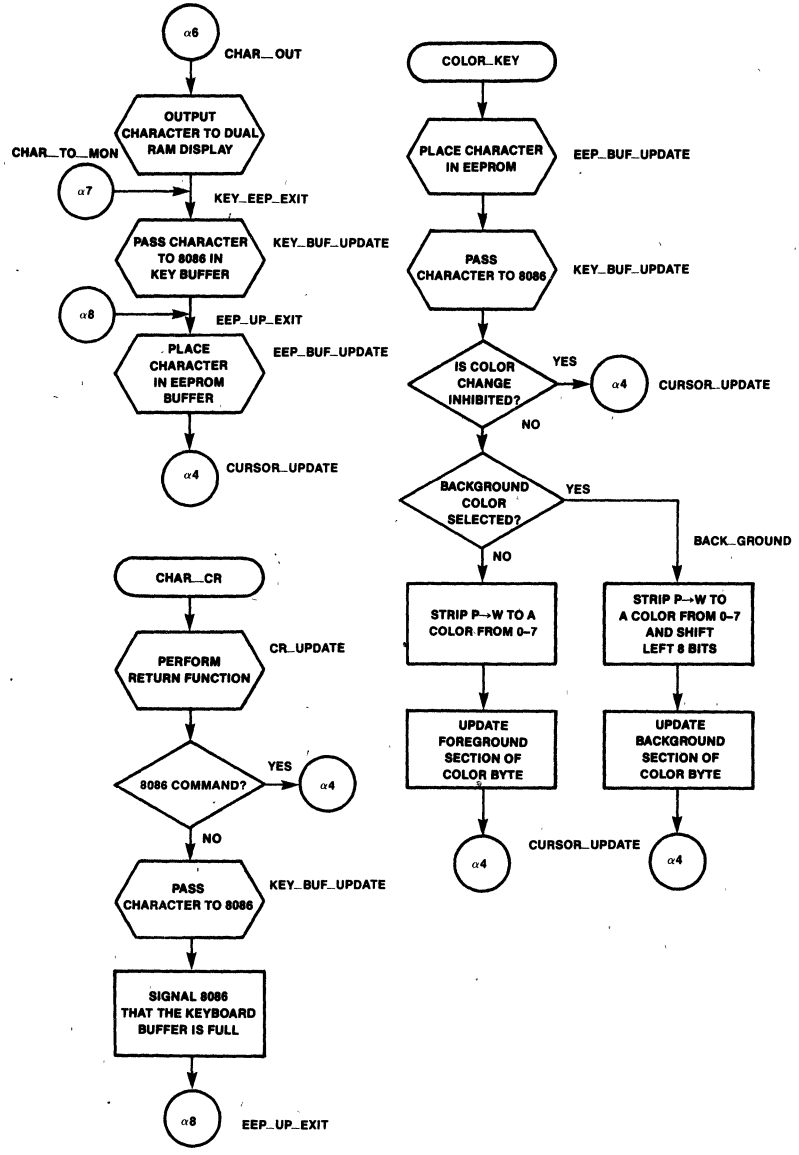
AFN–02172A

CONTROL KEY OPERATIONS



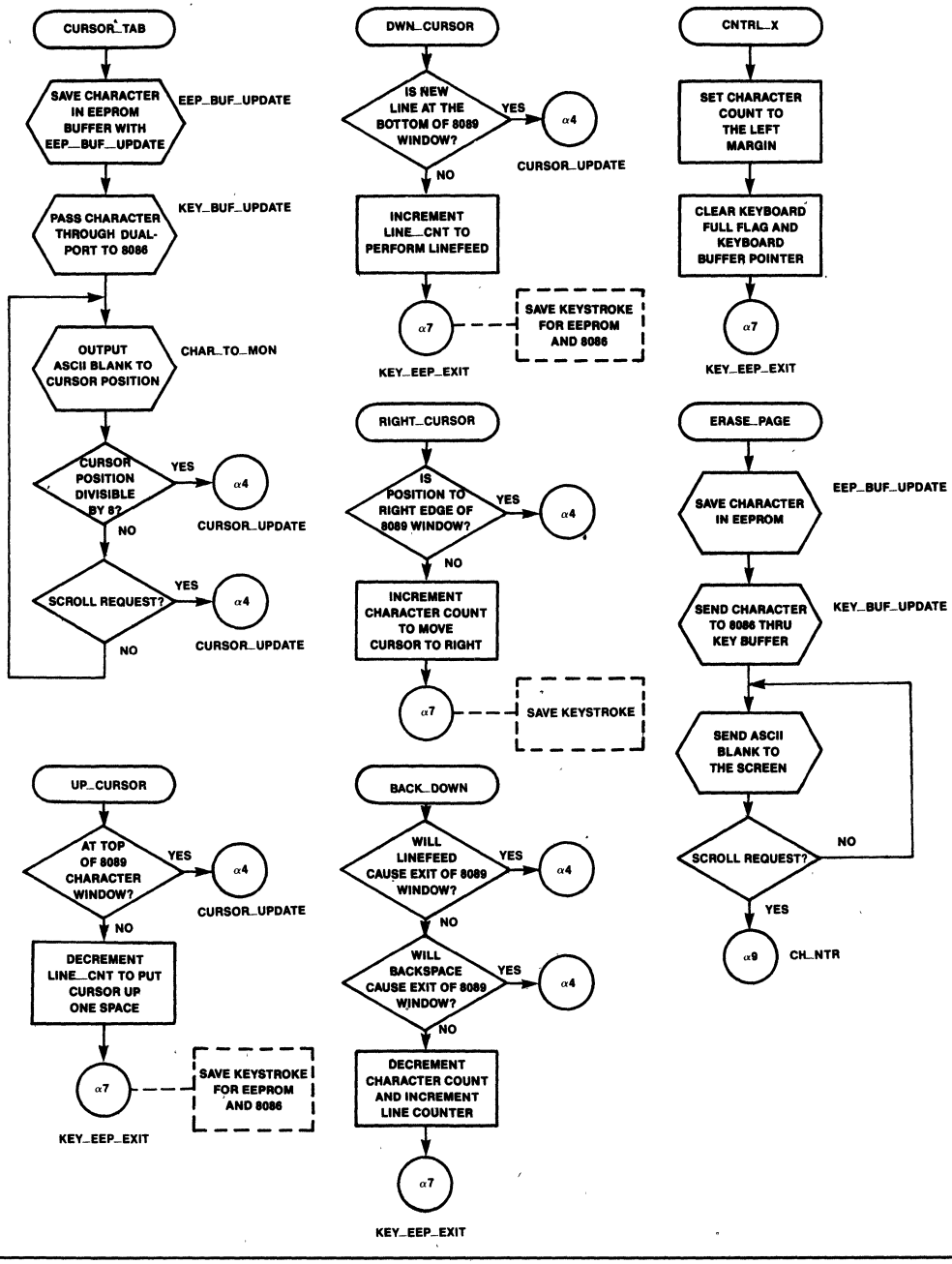
AFN-02172A

CONTROL KEY OPERATIONS



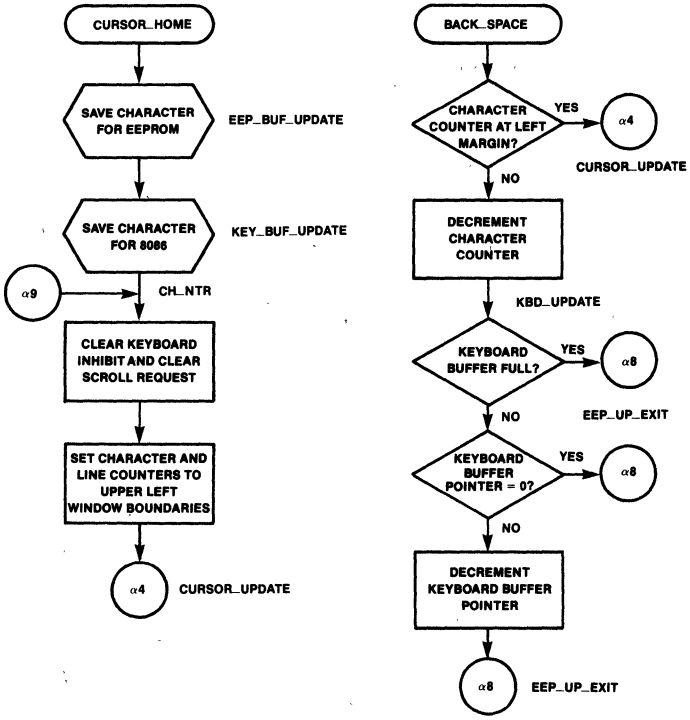
AFN–02172A

CONTROL KEY OPERATIONS

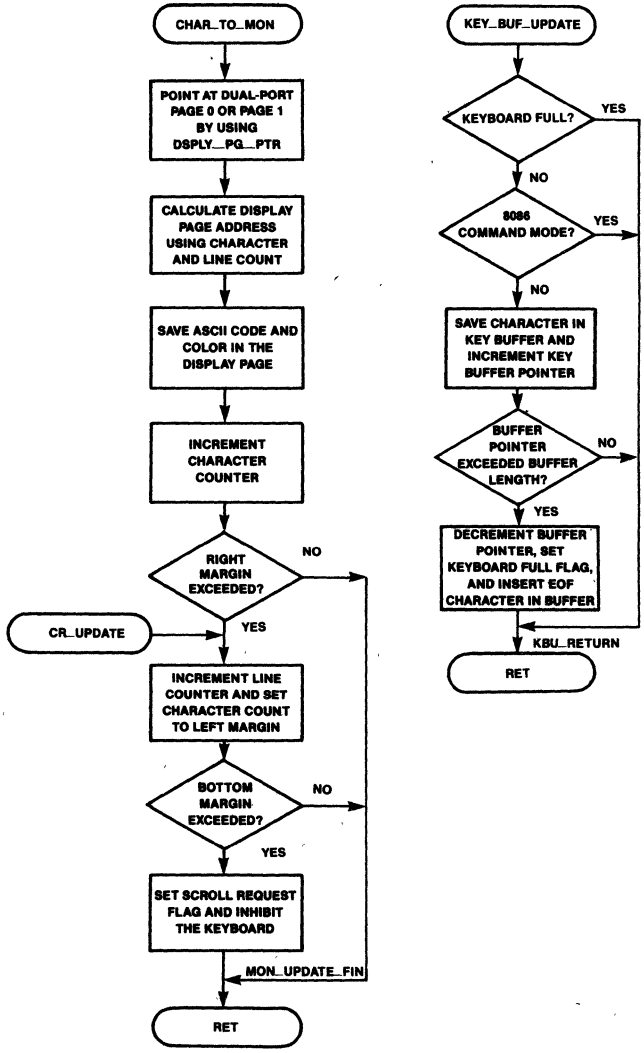


AFN-02172A

CONTROL KEY OPERATIONS

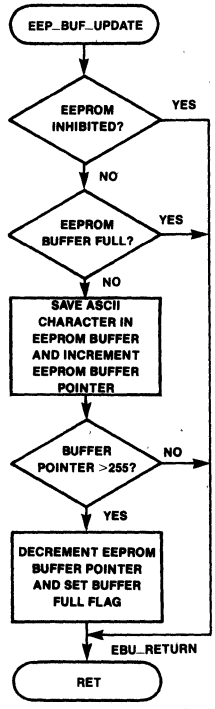


SUPPORT SUBROUTINES



AFN-02172A

SUPPORT SUBROUTINES



```
8089 MACRO ASSEMBLER


ISIS-II 8089 MACRO ASSEMBLER X202 ASSEMBLY OF MODULE N89
OBJECT MODULE PLACED IN :F1:N89.OBJ
ASSEMBLER INVOKED BY:   :F2:ASM89   :F1:N89.SRC
 1 ;
 2 ;   8089 DUMB TERMINAL PROGRAM
 3 ;
 4 ;   B.K.NELSON
 5 ;
 6 ;   STARTED:   4/30/80
 7 ;   LAST CHANGE:  8/12/80
 8 ;
 9 ;   THIS PROGRAM INITIALIZES FOUR 8275 CRT CONTROLLERS AND A
10 ;   8279 KEYBOARD CONTROLLER.   ASCII INFORMATION FLOW MAY FOLLOW
11 ;   THESE PATHS:
12 ;                   KEYBOARD   TO   8086 COMMAND INTERPRETER
13 ;                   KEYBOARD   TO   8086 EEPROM ROUTINE
14 ;                   KEYBOARD   TO   MONITOR
15 ;                   8086       TO   MONITOR
16 ;                   EEPROM     TO   8086 COMMAND INTERPRETER
17 ;                   EEPROM     TO   8086 EEPROM ROUTINE
18 ;                   EEPROM     TO   MONITOR
19 ;
20 ;   COMMAND CODES ARE:
21 ;K E
22 ;- -                CNTRL-A          MONITOR INHIBIT
23 ;- -                CNTRL-B          MONITOR UNINHIBIT
24 ;- -                CNTRL-C          EEPROM INHIBIT
25 ;- -                CNTRL-D          EEPROM UNINHIBIT
26 ;- -                CNTRL-E          TURN ON EEPROM BUFFER
27 ;- -                CNTRL-F          DISPLAY PAGE 0 SELECTED
28 ;- -                CNTRL-G          DISPLAY PAGE 1 SELECTED
29 ;O X                CNTRL-H          BACKSPACE (CURSOR LEFT)
30 ;X X                CNTRL-I          TAB (EVERY 8 CHARACTERS)
31 ;X X                CNTRL-J          LINEFEED (CURSOR DOWN)
32 ;X -                CNTRL-K          TURN EEPROM BUFFER OFF
33 ;X X                CNTRL-L          ERASE PAGE
34 ;X X                CNTRL-M          CARRIAGE RETURN
35 ;X X                CNTRL-N          TURN OFF BACKGROUND/FOREGROUND*
36 ;X X                CNTRL-O          TURN ON BACKGROUND/FOREGROUND*
37 ;X X                CNTRL-P          SET COLOR TO BLACK
38 ;X X                CNTRL-Q          SET COLOR TO RED
39 ;X X                CNTRL-R          GREEN
40 ;X X                CNTRL-S          YELLOW
41 ;X X                CNTRL-T          BLUE
42 ;X X                CNTRL-U          MAGENTA
43 ;X X                CNTRL-V          CYAN
44 ;X X                CNTRL-W          WHITE
45 ;O X                CNTRL-X          ABORT LINE
46 ;X X                CNTRL-Y          MOVE CURSOR RIGHT
47 ;X X                CNTRL-Z          MOVE CURSOR DOWN AND LEFT
48 ;X X                CNTRL-^          MOVE CURSOR UP
49 ;X X                CNTRL-\          HOME CURSOR
50 ;- -                CNTRL-DEL        RECALL EEPROM BUFFER
51 ;
```

```
LINE SOURCE
  52 ;
  53 ;   THE TWO COLUMNS ASSOCIATED WITH EACH CONTROL KEY REPRESENT TH/
          -E
  54 ;   APPROPRIATE KEYBOARD AND EEPROM BUFFER ACTION CONNECTED WITH /
          -THAT
  55 ;   KEY.
  56 ;                  -  KEYSTROKE NOT STORED IN BUFFER
  57 ;                  X  KEYSTROKE STORED IN BUFFER
  58 ;                  O  OPERATION PERFORMED ON BUFFER
  59 ;
  60 ;   A CHARACTER IS STORED IN THE EEPROM BUFFER ONLY IF THE OPERAT/
          -ION
  61 ;   WAS PERFORMED ON THE MONITOR.
  62 DUMBTERM           SEGMENT
  63 ;
  64 ;   8275 REGISTERS
  65 ;
  66 CRT_REGS           STRUC
  67    CRT_PARAM:      DW        1
  68    CRT_COM_STAT:   DW        1
  69 CRT_REGS           ENDS
  70 ;
  71 ;   8279 REGISTERS
  72 ;
  73 KYBD_REGS          STRUC
  74    KBD_DATA:       DW        1
  75    KBD_COM_STAT:   DW        1
  76 KYBD_REGS          ENDS
  77 ;
  78 ;   8086/8089 COMMON FLAGS
  79 ;
  80 DP_RAM_FLAGS       STRUC
  81    TP_LSW:         DW        1
  82    TP_MSD:         DW        1
  83    EEP_INH:        DB        1
  84    EEP_BUF_FULL:   DB        1
  85    EEP_RECALL:     DB        1
  86    COL_CH_INH:     DB        1
  87    KBD_INH:        DB        1
  88    KBD_BUF_FULL:   DB        1
  89 ;
  90 ;
  91    COM_8086:       DB        1
  92    COLOR:          DB        1
  93    STR_PTR_8086:   DW        1
  94    BACK_COL_SW:    DB        1
  95    MON_INH:        DB        1
  96    DSPLY_PG_PTR:   DB        1
  97    SCROLL_REQ:     DB        1
  98    NEW_CHAR_FLAG:            DB        1
  99    NEW_CHAR:                 DB        1
 100 ;
 101 ;
 102    MON_HOM:        DW        1
 103    MON_END:        DW        1
 104    MON_LMARG:      DW        1
 105    MON_RMARG:      DW        1
```

```
LINE SOURCE
 106    KBD_BUF_PTR:    DW        1
 107    E2_MON_INH:     DB        1
 108 ;
 109 DP_RAM_FLAGS       ENDS
 110 ;
 111 ;   DISPLAY CHARACTER STRUCTURE
 112 ;
 113 CHAR_DEF           STRUC
 114    COLOR_MODE:     DB        1
 115    ASCII_GRAPH1:   DB        1
 116    GRAPH_2AND3:    DB        1
 117    GRAPH_4AND5:    DB        1
 118 CHAR_DEF           ENDS
 119 ;
 120 ;   PRIVATE 8089 FLAGS
 121 ;
 122 STAT_RAM_FLAGS     STRUC
 123    STACK:          DW        1
 124    STACK_MSD:      DW        1
 125                    DW        1
 126                    DW        1
 127    EEP_BUF_PTR:    DW        1
 128 ;
 129 ;
 130    LINE_CNT:       DW        1
 131    CHAR_CNT:       DW        1
 132 ;
 133 ;
 134    ASCII:          DB        1
 135    ASCII_TEMP:     DB        1
 136    CURSOR_X1:      DB        1
 137    CURSOR_X2:      DB        1
 138    CURSOR_Y1:      DB        1
 139    CURSOR_Y2:      DB        1
 140 ;
 141 ;
 142    LINE_TEMP:      DW        1
 143    CHAR_TEMP:      DW        1
 144    PAGE_INDEX:     DW        1
 145 STAT_RAM_FLAGS     ENDS
 146 ;
 147 ;   ADDRESS EQUATES
 148 ;
 149 STAT_RAM           EQU       00000H
 150 CRT1               EQU       06000H
 151 CRT2               EQU       04000H
 152 CLK_EN             EQU       08000H
 153 CRT_DATA           EQU       0A000H
 154 KYBD               EQU       0C000H
 155 ;
 156 ;
 157 DSPLY_PAGE0        EQU       0F8000H
 158 DSPLY_PAGE1        EQU       0FC000H
 159 COM_BUF            EQU       0FBD00H
 160 EEP_BUF            EQU       0FBE00H
 161 KEY_BUF            EQU       0FBF00H
 162 DP_PB              EQU       0FFF00H
```

```
LINE SOURCE
 163 ;
 164 ;   DATA/COMMAND EQUATES
 165 ;
 166 EOF             EQU     0FFH
 167 CRT_RST         EQU     000H
 168 CRT_PARAM1      EQU     04F4FH
 169 CRT_PARAM2      EQU     06F6FH
 170 CRT_PARAM3      EQU     04444H
 171 CRT_PARAM4      EQU     00606H
 172 CRT_CURSOR      EQU     08080H
 173 CRT_CNTR        EQU     0E0E0H
 174 START_DISP      EQU     02020H
 175 END_DISP_PG     EQU     15360
 176 KBD_STR_SET     EQU     006H
 177 KBD_PRG_CLK     EQU     034H
 178 KBD_FIFO_RD     EQU     050H
 179 ;************************************************
 180 ;***********  INITIALIZATION  *******************
 181 ;************************************************
 182 ;
 183 ;   TURN ON THE CRT CHARACTER CLOCK AND RESET THE
 184 ;   CRT CONTROLLERS
 185 ;
 186 START:
 187         MOVI    GB,CLK_EN
 188         MOVI    [GB],001H
 189         MOVI    GB,CRT1
 190         MOVI    GC,CRT2
 191         MOVI    [GC].CRT_COM_STAT,CRT_RST
 192         MOVI    [GB].CRT_COM_STAT,CRT_RST
 193 ;
 194 ;   SUPPLY THE FOUR PARAMETER BYTES THAT SPECIFY
 195 ;   80X48 CHARACTERS, TRANSPARENT ATTRIBUTES, AND
 196 ;   A BLINKING UNDERLINE CURSOR
 197 ;
 198         MOVI    [GB],CRT_PARAM1
 199         MOVI    [GB],CRT_PARAM2
 200         MOVI    [GB],CRT_PARAM3
 201         MOVI    [GB],CRT_PARAM4
 202         MOVI    [GC],CRT_PARAM1
 203         MOVI    [GC],CRT_PARAM2
 204         MOVI    [GC],CRT_PARAM3
 205         MOVI    [GC],CRT_PARAM4
 206 ;
 207 ;   SET CURSOR TO UPPER LEFT CORNER OF MONITOR
 208 ;
 209         MOVI    [GC].CRT_COM_STAT,CRT_CURSOR
 210         MOVI    [GC],000H
 211         MOVI    [GC],000H
 212         MOVI    [GB].CRT_COM_STAT,CRT_CURSOR
 213         MOVI    [GB],000H
 214         MOVI    [GB],000H
 215 ;
 216 ;   SYNCHRONIZE 8275 CLUSTER BY RESETTING COUNTERS
 217 ;
 218         MOVI    [GC].CRT_COM_STAT,CRT_CNTR
 219         MOVI    [GB].CRT_COM_STAT,CRT_CNTR
```

```
LINE SOURCE
 220          MOVI     GC, STAT_RAM
 221          MOVBI    [GC]. CURSOR_X1, 000H
 222          MOVBI    [GC]. CURSOR_X2, 000H
 223          MOVBI    [GC]. CURSOR_Y1, 000H
 224          MOVBI    [GC]. CURSOR_Y2, 000H
 225 ;
 226 ;   INITIALIZE 8279 KEYBOARD CONTROLLER
 227 ;
 228          MOVI     GB, KYBD
 229          MOVI     [GB]. KBD_COM_STAT, KBD_STR_SET
 230          MOVI     [GB]. KBD_COM_STAT, KBD_PRG_CLK
 231          MOVI     [GB]. KBD_COM_STAT, KBD_FIFO_RD
 232 ;
 233 ;   INITIALIZE 8089 FLAGS
 234 ;
 235          MOVI     GC, STAT_RAM
 236          LPDI     GA, DP_PB
 237          MOVI     [GC]. LINE_CNT, 000H
 238          MOVI     [GC]. CHAR_CNT, 000H
 239 ;
 240 ;
 241          MOVBI    [GA]. EEP_INH, 0FFH
 242          MOVBI    [GA]. EEP_BUF_FULL, 00H
 243          MOVBI    [GA]. EEP_RECALL, 00H
 244          MOVBI    [GA]. KBD_INH, 00H
 245          MOVBI    [GA]. KBD_BUF_FULL, 00H
 246          MOVBI    [GA]. COM_8086, 00H
 247          MOVBI    [GA]. COLOR, 038H
 248          MOVBI    [GA]. BACK_COL_SW, 00H
 249          MOVBI    [GA]. COL_CH_INH, 00H
 250          MOVBI    [GA]. SCROLL_REQ, 00H
 251          MOVBI    [GA]. DSPLY_PG_PTR, 00H
 252          MOVBI    [GA]. MON_INH, 00H
 253          MOVBI    [GA]. E2_MON_INH, 0
 254          MOVI     [GA]. MON_HOM, 00H
 255          MOVI     [GA]. MON_END, 048
 256          MOVI     [GA]. MON_RMARG, 080
 257          MOVI     [GA]. MON_LMARG, 00H
 258 ;
 259 ;   INITIALIZE 8089 POINTER
 260 ;
 261          MOVI     [GC]. EEP_BUF_PTR, 00H
 262          MOVI     [GA]. STR_PTR_8086, 00H
 263          MOVI     [GA]. KBD_BUF_PTR, 000H
 264 ;************************************************
 265 ;************  EXECUTIVE  ***********************
 266 ;************************************************
 267 ;
 268 ;   DMA SET-UP
 269 ;
 270 ;   LOAD CHANNEL CONTROL REGISTER TO SPECIFY:
 271 ;                MEMORY TO PORT
 272 ;                SYNCHRO ON DEST
 273 ;                GA POINTS TO SOURCE
 274 ;                TERMINATE ON EXT
 275 ;                TERMINATION OFFSET=0
 276 ;
```

```
LINE SOURCE
 277          MOVI     GC,CLK_EN
 278          MOVI     [GC],00H          ;INHIBIT CHAR CLOCK
 279                                     ;ON 8275 TO SYNCHRONIZE
 280          MOVI     GC,CRT1
 281          MOVI     [GC] CRT_COM_STAT,START_DISP
 282          MOVI     GC,CRT2
 283          MOVI     [GC].CRT_COM_STAT,START_DISP
 284 DMA_LP:
 285          MOVI     CC,0512OH
 286 ;
 287 ;   SETUP DESTINATION AND THEN
 288 ;   SOURCE ACCORDING TO DISPLAY PAGE
 289 ;   POINTER
 290 ;
 291          MOVI     GB,CRT_DATA
 292          LPDI     GA,DSPLY_PAGE0
 293          LPDI     GC,DP_PB
 294          JZB      [GC].DSPLY_PG_PTR,SOURCE_OK
 295          LPDI     GA,DSPLY_PAGE1
 296 SOURCE_OK:
 297          JNZB     [GC].MON_INH,DMA_BYPASS ;IF THE MONITOR IS INHIB/
       -ITED
 298                                           ;BYPASS THE DMA
 299          JNZB     [GC].E2_MON_INH,DMA_BYPASS_1
 300          MOVI     GC,CLK_EN
 301 ;
 302 ;   START CRT CHARACTER CLOCK AND BEGIN DMA
 303 ;
 304          XFER
 305          MOVI     [GC],01H
 306          SINTR
 307 ;
 308 ;   SIGNAL THE 8086 THAT END OF FRAME HAS OCCURED AND THE UPDATIN/
       -G OF THE
 309 ;   INTERRUPT DRIVEN SECONDS COUNTER MAY BEGIN
 310 ;
 311 ;
 312 ;   READ CRT STATUS REGISTERS IN ORDER TO RESET IRQ
 313 ;
 314          MOVI     GC,CRT1
 315          MOV      GA,[GC].CRT_COM_STAT
 316          MOVI     GC,CRT2
 317          MOV      GB,[GC].CRT_COM_STAT
 318          JMP      DMA_BYPASS
 319 DMA_BYPASS_1:
 320          MOVI     GC,120
 321 E2_WAIT_LOOP:
 322          MOVI     GB,300
 323 E2_INNER_LOOP:
 324          DEC      GB
 325          JNZ      GB,E2_INNER_LOOP
 326          DEC      GC
 327          JNZ      GC,E2_WAIT_LOOP
 328 DMA_BYPASS:
 329 ;
 330 ;   CHECK FOR STRING FROM 8086
 331 ;   IT HAS PRIORITY OVER KEYBOARD
```

```
LINE SOURCE
 332 ;
 333          LPDI     GC,DP_PB
 334          JNZB     [GC].COM_8086,STRING_86
 335 ;
 336 ;   CHECK 8279 KYBD STATUS
 337 ;
 338          MOVI     GB,KYBD
 339          MOVB     GA,[GB].KBD_COM_STAT
 340          ANDI     GA,0FH
 341          LJNZ     GA,READ_KYBD     ;KEY DOWN
 342 ;
 343 ;   UPDATE THE CURSOR POSITION
 344 ;
 345 CURSOR_UPDATE:
 346          LPDI     GC,DP_PB
 347 ;
 348 ;   CHECK FOR 86 COMMAND CHARACTER MODE AND PROCESS
 349 ;   THE NEXT BYTE
 350          JZB      [GC].COM_8086,COM_STR_BYPASS
 351          INC      [GC].STR_PTR_8086
 352          JMP      GET_COM
 353 COM_STR_BYPASS:
 354          MOVI     GB,CRT1
 355          MOVI     GC,STAT_RAM
 356          MOVI     [GB].CRT_COM_STAT,CRT_CURSOR
 357          MOVB     GA,[GC].CHAR_CNT          ;SET UP FOR X POSITION
 358          MOVB     [GC].CURSOR_X1,GA         ;CURSOR OUTPUT
 359          MOVB     [GC].CURSOR_X2,GA         ;BY DOUBLING UP
 360          MOVB     GA,[GC].LINE_CNT
 361          MOVB     [GC].CURSOR_Y1,GA         ;SAME FOR Y POSITION
 362          MOVB     [GC].CURSOR_Y2,GA
 363          MOV      [GB],[GC].CURSOR_X1
 364          MOV      [GB],[GC].CURSOR_Y1
 365          MOVI     GB,CRT2                   ;DO IT FOR ALL
 366          MOVI     [GB].CRT_COM_STAT,CRT_CURSOR
 367          MOV      [GB],[GC].CURSOR_X1       ;CONTROLLERS
 368          MOV      [GB],[GC].CURSOR_Y1
 369 INTR_86:
 370          JMP      DMA_LP
 371 STRING_86:
 372          MOVI     [GC].STR_PTR_8086,00H
 373 GET_COM:
 374          MOV      IX,[GC].STR_PTR_8086
 375          LPDI     GB,COM_BUF
 376 ;
 377 ;   GET NEXT COMMAND CHARACTER FROM THE 8086
 378 ;   AND SAVE IT AS A KEYSTROKE
 379 ;
 380          MOVB     GA,[GB+IX]
 381          LPDI     GC,COM_BUF        ;***TEST CODE****
 382          MOVB     GA,[GB + IX]      ;***
 383          LPDI     GC,DP_PB          ;***
 384          MOVI     GB,STAT_RAM
 385          MOVB     [GB].ASCII,GA
 386 ;
 387 ;   CHECK FOR END OF COMMAND STRING
 388 ;
```

```
LINE SOURCE
 389          MOVI     MC,OFFFFH
 390          JMCNE    [GB].ASCII,COM_CNT
 391 ;
 392 ;   END OF COMMAND STRING-RESET COMMAND FLAG
 393 ;
 394          MOVBI    [GC].COM_8086,00H
 395          JMP      CURSOR_UPDATE
 396 READ_KYBD:
 397 ;
 398 ;   TEMPORARY GET CHAR ROUTINE
 399 ;
 400          JNZB     [GC].KBD_INH,CURSOR_UPDATE
 401          JNZB     [GC].KBD_BUF_FULL,CURSOR_UPDATE
 402 ;
 403 ;   IF THE KEYBOARD IS INHIBITED OR THE BUFFER FULL,
 404 ;   DONT READ THE 8279
 405 ;
 406          MOVB     GA,[GB].KBD_DATA
 407          NOT      GA
 408          ANDI     GA,007FH
 409          MOVB     [GC].NEW_CHAR,GA
 410          MOVBI    [GC].NEW_CHAR_FLAG,1
 411          MOVI     GB,STAT_RAM
 412          MOVB     [GB].ASCII,GA              ;SAVE KEYSTROKE
 413 COM_CNT:
 414          LPDI     GB,DP_PB
 415          MOVI     GC,STAT_RAM
 416 ;
 417 ;   CHECK FOR FIRST CHARACTER AFTER CNTRL-DEL, THIS CHARACTER WILL
 418 ;   BE PLACED IN EEP_RECALL AND USED FOR SELECTING WHICH EEP BUFF/
       -ER
 419 ;   IS TO BE RECALLED
 420 ;
 421          MOVB     GA,[GB].EEP_RECALL         ;IF MSB OF EEP_RECALL IS/
       - SET
 422          ANDI     GA,007FH                   ;USE PRESENT ASCII CHARA/
       -CTER
 423          JZ       GA,NO_RECALL               ;AS INDEX FOR EEPROM REC/
       -ALL
 424          MOVB     GA,[GC].ASCII
 425          MOVB     [GB].EEP_RECALL,GA
 426          JMP      CURSOR_UPDATE
 427 NO_RECALL:
 428 ;
 429 ;   CHECK FOR FIRST CHARACTER AFTER CNTRL_E
 430 ;   THIS CHARACTER WILL BE PLACED IN THE
 431 ;   EEPROM BUFFER AND NOT PROCESSED
 432 ;
 433          JNZB     [GB].EEP_INH,EEP_BYPASS
 434          JNZ      [GC].EEP_BUF_PTR,EEP_BYPASS
 435 ;
 436 ;   INSERT ASCII CHARACTER
 437 ;
 438          MOV      IX,[GC].EEP_BUF_PTR
 439          MOVB     GA,[GC].ASCII
 440          LPDI     GB,EEP_BUF
 441          MOVB     [GB+IX],GA
```

```
LINE SOURCE
 442            INC       [GC].EEP_BUF_PTR
 443            JMP       CURSOR_UPDATE
 444 EEP_BYPASS:
 445 ;
 446 ;   CHECK FOR NON CONTROL CHARACTER
 447 ;
 448            MOVI      MC,06000H
 449            LJMCNE    [GC].ASCII,CHAR_OUT
 450 ;
 451 ;*************************************************************
 452 ;*************** CONTROL KEY DECODE **************************
 453 ;*************************************************************
 454 ;
 455 ;   LOOK FOR 8086 COMMAND STRING SO CERTAIN
 456 ;   COMMANDS WILL NOT BE AVAILABLE FROM
 457 ;   KEYBOARD
 458 ;
 459 ;          JZB       [GB].COM_8086,NOT_CNTRLG
 460 ;
 461 ;   CHECK FOR MONITOR INHIBIT
 462 ;          (CNTRL-A)
 463 ;
 464            MOVI      MC,07F01H
 465            JMCNE     [GC].ASCII,NOT_CNTRLA
 466            MOVBI     [GB].MON_INH,0FFH
 467            JMP       CURSOR_UPDATE
 468 NOT_CNTRLA:
 469 ;
 470 ;   CHECK FOR MONITOR UNINHIBIT
 471 ;          (CNTRL-B)
 472 ;
 473            MOVI      MC,07F02H
 474            JMCNE     [GC].ASCII,NOT_CNTRLB
 475            MOVBI     [GB].MON_INH,00H
 476            JMP       CURSOR_UPDATE
 477 NOT_CNTRLB:
 478 NOT_CNTRLC:
 479 NOT_CNTRLD:
 480 ;
 481 ;   CHECK FOR SET DISPLAY PAGE 0
 482 ;          (CNTRL-F)
 483 ;
 484            MOVI      MC,07F06H
 485            JMCNE     [GC].ASCII,NOT_CNTRLF
 486            MOVBI     [GB].DSPLY_PG_PTR,00H
 487            JMP       CURSOR_UPDATE
 488 NOT_CNTRLF:
 489 ;
 490 ;   CHECK FOR SET DISPLAY PAGE 1
 491 ;          (CNTRL-G)
 492 ;
 493            MOVI      MC,07F07H
 494            JMCNE     [GC].ASCII,NOT_CNTRLG
 495            MOVBI     [GB].DSPLY_PG_PTR,0FFH
 496            JMP       CURSOR_UPDATE
 497 NOT_CNTRLG:
 498 ;
```

```
LINE SOURCE
 499 ;   THE FOLLOWING CONTROL COMMANDS ARE
 500 ;   AVAILABLE THROUGH THE 8089 KEYBOARD
 501 ;
 502 ;
 503 ;   LOOK FOR CARRIAGE RETURN
 504 ;
 505           MOVI     MC,07F0DH
 506           LJMCE    [GC].ASCII,CHAR_CR
 507 ;
 508 ;   LOOK FOR BACKSPACE
 509 ;
 510           MOVI     MC,07F08H
 511           LJMCE    [GC].ASCII,BACK_SPACE
 512 ;
 513 ;   LOOK FOR COLOR CONTROL KEYS
 514 ;   CNTRL-P THRU CNTRL-W
 515 ;
 516           MOVI     MC,07810H
 517           LJMCE    [GC].ASCII,COLOR_KEY
 518 ;
 519 ;   CHECK FOR SET BACKGROUND COLOR FLAG
 520 ;         (CNTRL-N)
 521 ;
 522           MOVI     MC,07F0EH
 523           LJMCE    [GC].ASCII,CNTRL_N
 524 ;
 525 ;   CHECK FOR SET FOREGROUND COLOR
 526 ;         (CNTRL-O)
 527           MOVI     MC,07F0FH
 528           LJMCE    [GC].ASCII,CNTRL_O
 529 ;
 530 ;
 531 ;
 532 ;   CHECK FOR EEPROM BUFFER RECALL
 533 ;         (CNTRL-DEL)
 534           MOVI     MC,07F1FH
 535           LJMCE    [GC].ASCII,EEP_DUMP
 536 ;
 537 ;   LOOK FOR TAB
 538 ;         (CNTRL-I)
 539 ;
 540           MOVI     MC,07F09H
 541           LJMCE    [GC].ASCII,CURSOR_TAB
 542 ;
 543 ;   LOOK FOR ERASE PAGE
 544 ;         (CNTRL-L)
 545 ;
 546           MOVI     MC,07F0CH
 547           LJMCE    [GC].ASCII,ERASE_PAGE
 548 ;
 549 ;   LOOK FOR CANCEL LINE
 550 ;         (CNTRL-X)
 551 ;
 552           MOVI     MC,07F18H
 553           LJMCE    [GC].ASCII,CNTRL_X
 554 ;
 555 ;   LOOK FOR HOME THE CURSOR
```

```
LINE SOURCE
 556 ;          (CNTRL \)
 557 ;
 558            MOVI     MC, 07F1CH
 559            LJMCE    [GC].ASCII, CURSOR_HOME
 560
 561 ;    LOOK FOR UP CURSOR
 562 ;          (CNTRL ^)
 563            MOVI     MC, 07F1EH
 564            LJMCE    [GC].ASCII, UP_CURSOR
 565 ;
 566 ;    LOOK FOR DOWN CURSOR
 567 ;          (CNTRL J)
 568 ;
 569            MOVI     MC, 07F0AH
 570            LJMCE    [GC].ASCII, DWN_CURSOR
 571 ;
 572 ;    LOOK FOR RIGHT CURSOR
 573 ;          (CNTRL-Y)
 574 ;
 575            MOVI     MC, 07F19H
 576            LJMCE    [GC].ASCII, RIGHT_CURSOR
 577 ;
 578 ;    LOOK FOR DOWN AND LEFT CURSOR
 579 ;          (CNTRL-Z)
 580 ;
 581            MOVI     MC, 07F1AH
 582            LJMCE    [GC].ASCII, BACK_DOWN
 583 ;
 584 ;    ALL OTHER KEY INPUTS ARE IGNORED
 585 ;
 586            JMP      CURSOR_UPDATE
 587 ;*************************************************
 588 ;*************  CONTROL SEGMENTS  ****************
 589 ;*************************************************
 590 ;
 591 ;
 592 ;    SET THE COLOR BACKGROUND/FOREGROUND* FLAG TO
 593 ;    BACKGROUND (0)
 594 ;
 595 CNTRL_N:
 596            MOVI     GB, STAT_RAM
 597            LPDI     GC, DP_PB
 598 ;
 599 ;    CHECK FOR MONITOR OR COLOR CHANGE INHIBITED
 600 ;
 601            JNZB     [GC].COL_CH_INH, KEEP_BF
 602            MOVBI    [GC].BACK_COL_SW, 00H
 603 KEEP_BF:
 604            LJMP     KEY_EEP_EXIT
 605 ;
 606 ;    SET THE COLOR BACKGROUND/FOREGROUND* FLAG
 607 ;    TO FOREGROUND
 608 ;
 609 CNTRL_O:
 610            MOVI     GB, STAT_RAM
 611            LPDI     GC, DP_PB
 612 ;
```

```
LINE SOURCE
 613 ;   CHECK FOR MONITOR OR COLOR CHANGE INHIBITED
 614 ;
 615           JNZB      [GC].COL_CH_INH,KEEP_BF2
 616           MOVBI     [GC].BACK_COL_SW,0FFH
 617 KEEP_BF2:
 618           LJMP      KEY_EEP_EXIT
 619 ;
 620 ;   TURN ON THE EEPROM BUFFER
 621 ;         (CNTRL_E)
 622 ;
 623 ;   THIS ROUTINE INITIALIZES THE EEPROM BUFFER
 624 ;   POINTER
 625 ;
 626 CNTRL_E:
 627           MOVI      GB,STAT_RAM
 628           LPDI      GC,DP_PB
 629           LJNZB     [GC].EEP_BUF_FULL,CURSOR_UPDATE
 630           MOVBI     [GC].EEP_BUF_FULL,00H     ;********
 631           MOVI      [GB].EEP_BUF_PTR,00H
 632           MOVBI     [GC].EEP_INH,00H
 633           JMP       CURSOR_UPDATE
 634 ;
 635 ;   TURN THE EEPROM BUFFER OFF
 636 ;
 637 CNTRL_K:
 638           MOVI      GB,STAT_RAM
 639           LPDI      GC,DP_PB
 640           LCALL     [GB],KEY_BUF_UPDATE
 641           MOVBI     [GC].EEP_BUF_FULL,0FFH
 642           MOVBI     [GC].EEP_INH,0FFH
 643           MOV       IX,[GB].EEP_BUF_PTR
 644           LPDI      GA,EEP_BUF
 645 ;
 646 ;   INSERT END OF FILE MARKER
 647 ;
 648           MOVBI     [GA+IX],0FFH
 649           INC       [GB].EEP_BUF_PTR
 650           JMP       CURSOR_UPDATE
 651 ;
 652 ;   DUMP EEPROM BUFFER 0-9
 653 ;
 654 EEP_DUMP:
 655           MOVI      GB,STAT_RAM
 656           LPDI      GC,DP_PB
 657           LPDI      GC,DP_PB
 658           MOVBI     [GC].EEP_RECALL,0FFH      ;SET FLAG TO ALL ONES, B/
       -UT IT
 659                                               ;WILL BE REPLACED BY THE/
       - NEXT
 660                                               ;ASCII CHARACTER
 661 ED_XIT:
 662           JMP       INTR_86
 663 CHAR_OUT:
 664           MOVI      GB,STAT_RAM
 665           LCALL     [GB],CHAR_TO_MON
 666 ;
 667 ;   PASS KEYSTROKES TO 8086
```

```
LINE SOURCE
 668 ;
 669 KEY_EEP_EXIT:
 670           MOVI      GB, STAT_RAM
 671           LCALL     [GB], KEY_BUF_UPDATE
 672 EEP_UP_EXIT:
 673           MOVI      GB, STAT_RAM
 674           LCALL     [GB], EEP_BUF_UPDATE
 675           JMP       CURSOR_UPDATE
 676 CHAR_CR:
 677           MOVI      GB, STAT_RAM
 678           LCALL     [GB], CR_UPDATE
 679 ;
 680 ;   SET KEYBOARD AND EEPROM BUFFER FULL
 681 ;   FLAGS IF NOT INHIBITED
 682 ;
 683           MOVI      GB, STAT_RAM
 684           LPDI      GC, DP_PB
 685           JNZB      [GC].COM_8086, CURSOR_UPDATE      ; IF IN 8086 COMM/
       -AND
 686                                                       ; MODE, DONT ALTER
 687                                                       ; KEYBOARD STATUS
 688           MOVI      GB, STAT_RAM
 689           LCALL     [GB], KEY_BUF_UPDATE
 690           MOVBI     [GC].KBD_BUF_FULL, OFFH   ; ******
 691 EEP_CHK:
 692           JMP       EEP_UP_EXIT
 693 ;
 694 ;   ALTER BACKGROUND OR FOREGROUND COLOR ACCORDING
 695 ;   TO THE 3 LEAST SIGNIFICANT BITS OF THE INPUT
 696 ;   KEY AND THE STATUS OF THE BACKGROUND/FOREGROUND*
 697 ;   FLAG.
 698 ;
 699 COLOR_KEY:
 700           MOVI      GB, STAT_RAM
 701           LPDI      GC, DP_PB
 702           LCALL     [GB], EEP_BUF_UPDATE
 703           LCALL     [GB], KEY_BUF_UPDATE
 704           LJNZB     [GC].COL_CH_INH, CURSOR_UPDATE
 705           MOVB      GA, [GC].BACK_COL_SW
 706 ;
 707 ;   CHECK B/F* FLAG
 708 ;
 709           JNZ       GA, BACK_GROUND
 710           MOVB      GA, [GB].ASCII
 711           ANDI      GA, 07H
 712           MOV       [GB].ASCII, GA
 713           MOVB      GA, [GC].COLOR
 714           ANDI      GA, 038H
 715 ;
 716 ;   OR INPUT COLOR INTO FOREGROUND SECTION OF COLOR BYTE
 717 ;
 718           ORB       GA, [GB].ASCII
 719           MOVB      [GC].COLOR, GA
 720           JMP       CURSOR_UPDATE
 721 BACK_GROUND:
 722           MOVB      GA, [GB].ASCII
 723           ADD       GA, [GB].ASCII
```

```
LINE SOURCE
 724            ADD       GA,[GB].ASCII
 725            ADD       GA,[GB].ASCII
 726            MOVB      [GB].ASCII_TEMP,GA
 727            ADD       GA,[GB].ASCII_TEMP
 728 ;
 729 ;   SHIFT INPUT COLOR OVER AND OR IT INTO THE BACKGROUND
 730 ;   SECTION OF THE COLOR BYTE
 731 ;
 732            ANDI      GA,038H
 733            MOV       [GB].ASCII,GA
 734            MOVB      GA,[GC].COLOR
 735            ANDI      GA,047H
 736            ORB       GA,[GB].ASCII
 737            MOVB      [GC].COLOR,GA
 738            JMP       CURSOR_UPDATE
 739 ;
 740 ;   TAB ROUTINE
 741 ;
 742 ;   THIS ROUTINE MOVES THE CURSOR TO THE NEXT
 743 ;   COLUMN WHOSE NUMBER IS A MULTIPLE OF 8.
 744 ;
 745 CURSOR_TAB:
 746            MOVI      GB,STAT_RAM
 747            LCALL     [GB],EEP_BUF_UPDATE
 748            LCALL     [GB],KEY_BUF_UPDATE
 749            LPDI      GC,DP_PB
 750 ;
 751 ;   CHECK FOR CHARACTER COUNT BEING A
 752 ;   MULTIPLE OF EIGHT (3 LSB = 0)
 753 ;
 754 TAB_CNT:
 755 ;
 756 ;   PLACE BLANK ON THE SCREEN
 757 ;
 758            MOVBI     [GB].ASCII,020H
 759            LCALL     [GB],CHAR_TO_MON
 760            MOV       GA,[GB].CHAR_CNT
 761            ANDI      GA,07H
 762            LJZ       GA,CURSOR_UPDATE
 763            JZB       [GC].SCROLL_REQ,TAB_CNT
 764            JMP       CURSOR_UPDATE
 765 ;
 766 ;   ERASE PAGE ROUTINE
 767 ;
 768 ;   THIS ROUTINE ERASES THE PAGE FROM THE CURRENT
 769 ;   CURSOR POSITION.   IT ENDS WITH THE CURSOR AT
 770 ;   THE HOME POSITION.
 771 ;
 772 ;
 773 ;   UP CURSOR ROUTINE
 774 ;
 775 UP_CURSOR:
 776            MOVI      GB,STAT_RAM
 777            LPDI      GC,DP_PB
 778            MOV       IX,[GC].MON_HOM
 779            NOT       IX                    ;CHECK FOR UPPER BOUNDARY
 780            AND       IX,[GB].LINE_CNT
```

```
LINE SOURCE
 781            LJZ       IX,CURSOR_UPDATE
 782            DEC       [GB].LINE_CNT
 783            JMP       KEY_EEP_EXIT
 784 ;
 785 ;   LINE FEED (DOWN CURSOR)
 786 ;
 787 DWN_CURSOR:
 788            MOVI      GB,STAT_RAM
 789            LPDI      GC,DP_PB
 790            MOV       IX,[GB].LINE_CNT          ;COMPARE PRESENT LINE
 791            INC       IX                        ;COUNT + 1 TO BOTTOM
 792            NOT       IX                        ;MARGIN
 793            AND       IX,[GC].MON_END           ;IF EQUAL ABORT CURSOR M/
        -OVE
 794            LJZ       IX,CURSOR_UPDATE
 795            INC       [GB].LINE_CNT             ;MOVE OK
 796            JMP       KEY_EEP_EXIT
 797 ;
 798 ;   MOVE CURSOR RIGHT
 799 ;
 800 RIGHT_CURSOR:
 801            MOVI      GB,STAT_RAM
 802            LPDI      GC,DP_PB
 803            MOV       IX,[GB].CHAR_CNT          ;COMPARE PRESENT CHARACT/
        -ER
 804            INC       IX                        ;COUNT + 1 TO RIGHT
 805            NOT       IX                        ;MARGIN
 806            AND       IX,[GC].MON_RMARG         ;IF EQUAL ABORT
 807            LJZ       IX,CURSOR_UPDATE          ;CURSOR MOVE
 808            INC       [GB].CHAR_CNT             ;MOV OK
 809            JMP       KEY_EEP_EXIT
 810 BACK_DOWN:
 811            MOVI      GB,STAT_RAM
 812            LPDI      GC,DP_PB
 813            MOV       IX,[GB].LINE_CNT          ;COMPARE PRESENT LINE
 814            INC       IX                        ;COUNT + 1 TO BOTTOM
 815            NOT       IX                        ;MARGIN
 816            AND       IX,[GC].MON_END           ;IF EQUAL ABORT CURSOR M/
        -OVE
 817            LJZ       IX,CURSOR_UPDATE
 818            MOV       IX,[GC].MON_LMARG         ;IF CURSOR IS AT LEFT MA/
        -RGIN
 819            NOT       IX                        ;ABORT CURSOR MOVE
 820            AND       IX,[GB].CHAR_CNT
 821            LJZ       IX,CURSOR_UPDATE
 822            INC       [GB].LINE_CNT
 823            DEC       [GB].CHAR_CNT
 824            JMP       KEY_EEP_EXIT
 825 ;
 826 ;   CANCEL THE PRESENT LINE
 827 ;
 828 CNTRL_X:
 829            MOVI      GB,STAT_RAM
 830            LPDI      GC,DP_PB
 831            MOV       [GB].CHAR_CNT,[GC].MON_LMARG
 832 ;
 833 ;   RESET THE KEYBOARD BUFFER POINTER
```

```
LINE SOURCE
 834 ;
 835          MOVBI    [GC].KBD_BUF_FULL,00H
 836          MOVI     [GC].KBD_BUF_PTR,00H
 837          JMP      KEY_EEP_EXIT
 838 ERASE_PAGE:
 839          MOVI     GB,STAT_RAM
 840          LCALL    [GB],EEP_BUF_UPDATE
 841          LCALL    [GB],KEY_BUF_UPDATE
 842          LPDI     GC,DP_PB
 843 ;
 844 ;  STORE BLANKS ON THE SCREEN
 845 ;
 846          MOVBI    [GB].ASCII,020H
 847 ERASE_CNT:
 848          LCALL    [GB],CHAR_TO_MON
 849          JZB      [GC].SCROLL_REQ,ERASE_CNT
 850          JMP      CH_NTR
 851 ;
 852 ;  HOME THE CURSOR
 853 ;
 854 CURSOR_HOME:
 855          MOVI     GB,STAT_RAM
 856          LCALL    [GB],EEP_BUF_UPDATE
 857          LCALL    [GB],KEY_BUF_UPDATE
 858 CH_NTR:
 859          LPDI     GC,DP_PB
 860          MOVBI    [GC].KBD_INH,00H
 861          MOVBI    [GC].SCROLL_REQ,00H
 862          MOV      [GB].CHAR_CNT,[GC].MON_LMARG
 863          MOV      [GB].LINE_CNT,[GC].MON_HOM
 864          JMP      CURSOR_UPDATE
 865 ;
 866 ;  PERFORM BACK-SPACE BY DECREMENTING THE DISPLAY
 867 ;  PAGE POINTER, KEYBOARD POINTER, EEPROM POINTER,
 868 ;  AND CURSOR POSITION
 869 ;
 870 BACK_SPACE:
 871          MOVI     GB,STAT_RAM
 872          LPDI     GC,DP_PB
 873          MOV      IX,[GC].MON_LMARG        ;IF CURSOR IS AT LEFT
 874          NOT      IX                       ;MARGIN ABORT BACKSPACE
 875          AND      IX,[GB].CHAR_CNT
 876          LJZ      IX,CURSOR_UPDATE
 877          DEC      [GB].CHAR_CNT
 878 ;
 879 ;  DO BACKSPACE IF MONITOR NOT INHIBITED AND CURSOR IS
 880 ;  NOT AT THE BEGINNING OF A LINE
 881 ;
 882 KYBD_UPDATE:
 883          LJNZB    [GC].KBD_BUF_FULL,EEP_EXIT
 884 ;
 885 ;  IF KEY BUFFER POINTER IS ZERO, DONT BACKSPACE IT
 886 ;
 887          JZ       [GC].KBD_BUF_PTR,EEP_EXIT
 888          DEC      [GC].KBD_BUF_PTR
 889 EEP_EXIT:
 890          MOVI     GB,STAT_RAM
```

```
LINE SOURCE
 891          JMP      EEP_UP_EXIT
 892 ;*************************************************
 893 ;***************  SUBROUTINES  *********************
 894 ;*************************************************
 895 CHAR_TO_MON.
 896 ;
 897 ;   SET UP DISPLAY PAGE POINTER AND INDEX
 898 ;
 899          LPDI     GB,DSPLY_PAGE0
 900          LPDI     GC,DP_PB
 901          JZ       [GC].DSPLY_PG_PTR,PTR_OK
 902          LPDI     GB,DSPLY_PAGE1
 903 ;
 904 ;   COMPUTE 80XLINE_CNT
 905 ;
 906 PTR_OK:
 907          MOVI     GC,STAT_RAM
 908          MOV      GA,[GC].LINE_CNT
 909          ADD      GA,[GC].LINE_CNT
 910          ADD      GA,[GC].LINE_CNT
 911          ADD      GA,[GC].LINE_CNT
 912          ADD      GA,[GC].LINE_CNT
 913          MOV      [GC].LINE_TEMP,GA
 914          ADD      GA,[GC].LINE_TEMP        ; 2 X 5
 915          MOV      [GC].LINE_TEMP,GA
 916          ADD      GA,[GC].LINE_TEMP        ; 4 X 5
 917          MOV      [GC].LINE_TEMP,GA
 918          ADD      GA,[GC].LINE_TEMP        ; 8 X 5
 919          MOV      [GC].LINE_TEMP,GA
 920          ADD      GA,[GC].LINE_TEMP        ; 16 X 5
 921 ;
 922 ;   MEMORY POINTER = DISPLAY PAGE POINTER +
 923 ;                    4X(80XLINE_CNT + CHAR_CNT)
 924 ;
 925          ADD      GA,[GC].CHAR_CNT
 926          MOV      [GC].LINE_TEMP,GA
 927          ADD      GA,[GC].LINE_TEMP
 928          ADD      GA,[GC].LINE_TEMP
 929          ADD      GA,[GC].LINE_TEMP
 930          MOV      [GC].PAGE_INDEX,GA
 931          ADD      GB,[GC].PAGE_INDEX
 932 ;
 933 ;   SAVE ASCII CODE IN DISPLAY PAGE
 934 ;
 935          MOVB     [GB].ASCII_GRAPH1,[GC].ASCII
 936 ;
 937 ;   SAVE BACKGROUND AND FOREGROUND COLOR IN
 938 ;   DISPLAY PAGE
 939 ;
 940          LPDI     GC,DP_PB
 941          MOVB     [GB].COLOR_MODE,[GC].COLOR
 942 ;
 943 ;   CLEAR OTHER 2 DISPLAY PAGE BYTES
 944 ;
 945          MOVBI    [GB].GRAPH_2AND3,00H
 946          MOVBI    [GB].GRAPH_4AND5,00H
 947 ;
```

```
LINE SOURCE
 948 ;   INCREMENT X CURSOR POSITION AND CHARACTER POINTER.
 949 ;   CHECK FOR RIGHT MARGIN OVERRUN
 950 ;
 951           MOVI     GB,STAT_RAM
 952           INC      [GB].CHAR_CNT
 953           MOV      [GB].CHAR_TEMP,[GB].CHAR_CNT
 954           NOT      [GB].CHAR_TEMP
 955           MOV      GA,[GC].MON_RMARG
 956           AND      GA,[GB].CHAR_TEMP
 957           JNZ      GA,MON_UPDATE_FIN
 958 CR_UPDATE:
 959 ;   IF RIGHT MARGIN WAS EXCEEDED, MOVE CHARACTER COUNT
 960 ;   TO LEFT MARGIN AND INCREMENT LINE COUNT AND Y CURSOR
 961 ;   POSITION
 962           LPDI     GC,DP_PB
 963           MOVI     GB,STAT_RAM
 964           INC      [GB].LINE_CNT
 965           MOV      [GB].CHAR_CNT,[GC].MON_LMARG
 966 ;
 967 ;   CHECK IF LINE COUNT WENT PAST BOTTOM OF SCREEN
 968 ;
 969           MOV      [GB].LINE_TEMP,[GB].LINE_CNT
 970           NOT      [GB].LINE_TEMP
 971           MOV      GA,[GB].LINE_TEMP
 972           AND      GA,[GC].MON_END
 973           JNZ      GA,MON_UPDATE_FIN
 974 ;
 975 ;   LINE COUNT EXCEEDED BOTTOM MARGIN--
 976 ;   SET SCROLL FLAG
 977 ;   AND KEYBOARD INHIBIT AND DECREMENT LINE COUNT
 978 ;
 979           MOVBI    [GC].SCROLL_REQ,0FFH
 980           MOVBI    [GC].KBD_INH,0FFH ;****
 981           DEC      [GB].LINE_CNT
 982 MON_UPDATE_FIN:
 983 ;
 984 ;   RETURN TO CALLING ROUTINE
 985 ;
 986           MOVI     GB,STAT_RAM
 987           LPDI     GC,DP_PB
 988           MOVP     TP,[GB]
 989 ;
 990 ;   KEYBOARD BUFFER SUBROUTINE
 991 ;
 992 ;   TRANSFER THE ASCII CHARACTERS OBTAINED FROM THE
 993 ;   8279 CONTROLLER INTO A BUFFER FOR LATER
 994 ;   PROCESSING BY THE 8086.
 995 ;
 996 KEY_BUF_UPDATE:
 997           LPDI     GC,DP_PB
 998           MOVI     GB,STAT_RAM
 999 ;
1000 ;   BYPASS IF BUFFER FULL
1001 ;
1002           JNZB     [GC].KBD_BUF_FULL,KBU_RETURN
1003 ;
1004 ;   BYPASS IF 8086 COMMAND MODE
```

```
LINE SOURCE
1005 ;
1006          JNZB      [GC] COM_8086,KBU_RETURN
1007 ;
1008 ;   XFER THE CHARACTER
1009 ;
1010          MOV       IX,[GC].KBD_BUF_PTR
1011          LPDI      GA,KEY_BUF
1012          MOVB      [GA+IX],[GB].ASCII
1013          INC       [GC].KBD_BUF_PTR
1014          MOV       GA,[GC].KBD_BUF_PTR
1015          ANDI      GA,OFFOOH
1016          JZ        GA,KBU_RETURN
1017 ;
1018 ;   POINTER OVERRUN-SET BUFFER FULL FLAG
1019 ;
1020          DEC       [GC].KBD_BUF_PTR
1021          MOVBI     [GC].KBD_BUF_FULL,OFFH
1022          MOVBI     [GA+IX],OFFH              ;SET END OF BUFFER MARKER
1023 KBU_RETURN:
1024          MOVP      TP,[GB]
1025 ;
1026 ;   EEPROM BUFFER SUBROUTINE
1027 ;
1028 ;   THIS ROUTINE TRANSFERS THE ASCII CHARACTERS OBTAINED
1029 ;   FROM THE 8279 CONTROLLER INTO THE DUAL PORT EEPROM BUFFER
1030 ;
1031 EEP_BUF_UPDATE:
1032          MOVI      GB,STAT_RAM
1033          LPDI      GC,DP_PB
1034 ;
1035 ;   CHECK FOR BUFFER FULL FLAG OR EEPROM INHIBITED
1036 ;
1037          JNZB      [GC].EEP_INH,EBU_RETURN
1038          JNZB      [GC].EEP_BUF_FULL,EBU_RETURN
1039 ;
1040 ;   XFER THE CHARACTER
1041 ;
1042          MOV       IX,[GB].EEP_BUF_PTR
1043          LPDI      GA,EEP_BUF
1044          MOVB      [GA+IX],[GB].ASCII
1045          INC       [GB].EEP_BUF_PTR
1046          MOV       GA,[GB].EEP_BUF_PTR
1047          ANDI      GA,OFFOOH
1048          JZ        GA,EBU_RETURN
1049 ;
1050 ;   POINTER OVERRUN-SET BUFFER FULL FLAG
1051 ;
1052          DEC       [GB].EEP_BUF_PTR
1053          MOVBI     [GC].EEP_BUF_FULL,OFFH
1054 EBU_RETURN:
1055          MOVP      TP,[GB]
1056 DUMBTERM          ENDS
1057                   END
```

intel

APPLICATION NOTE

AP-143

March 1982

Using the iAPX 86/20 Numeric Data Processor in a Small Business Computer

Ken Shoemaker
Microcomputer Applications

Order Number: 210383-001

INTRODUCTION

As the performance of microcomputers has improved, the types of functions performed by these microcomputers have grown. One application filled by these machines has been to perform typical "adding machine" type calculations, balancing ledgers, etc. This type of machine has come to be called a "small business computer." To be a true business computer, however, the types of operations performed by these machines need to be expanded beyond simple "balance the books" types of operations. There are many algorithms that have been impractical for these small business computers because the number of calculations required by the algorithms and the performance available from these machines did not make them feasible. Such operations were available only on large mainframe or minicomputers. With the introduction of the iAPX 86/20, a microcomputer can finally perform these types of calculations at a cost level appropriate to small business computers.

The iAPX 86/20 features the Intel 8086 with the 8087 numerics co-processor. This combination allows for high-performance, high-precision numeric calculations. Many types of operations require this performance to provide accurate results in a reasonable amount of time. This increased performance will also be particularly welcome in the interactive user environment, typically found in small business computers. It is very frustrating to wait many seconds or even minutes after hitting "return" for the computer to generate results.

In general, if there are many methods to solving a business computer problem, the method requiring the largest number or calculations will provide the best results. In many applications, approximate methods have been used because the speed of the hardware (or the cost of the computer time) did not allow a more exact method to be used. Because of the high performance of the iAPX 86/20, these numeric intensive methods may now be used in small business computer software.

The types of calculations demonstrated in this note are:

- Interest and Annuities. These calculations require the use of floating point multiplication, division, exponentiation and logarithms. These calculations are used to determine the present or future value of certain types of funds.
- Restocking. These iterative calculations require extensive use of floating point multiplication and division. They are used to determine the optimum restocking times for a given item when the set-up charges, holding costs and demand for the item are known or can be estimated.
- Linear Programming. These calculations require extensive use of floating point multiplication and division. One of many applications for linear programming is the determination of optimum production quantities of diverse products when the quantities of their various constituents are both overlapping and limited.

iAPX 86/20 HARDWARE OVERVIEW

The iAPX 86/20 is a 16-bit microprocessor based on the Intel 8086 CPU. The 8086 CPU features eight internal general-purpose 16-bit registers, memory segmentation, and many other features allowing for efficient code generation from high-level language compilers. When augmented with the 8087, it becomes a vehicle for high-speed numerics processing. The 8087 adds eight 80-bit internal floating point registers, and a floating point arithmetic logic unit (ALU) which can speed floating point operations up to 100 times over other software floating point simulators or emulators.

The 8086 and 8087 execute a single instruction stream. The 8087 monitors this stream for numeric instructions. When a numeric instruction is decoded, the 8086 generates any needed memory addresses for the 8087. The 8087 then begins instruction execution automatically. No other software interface is required, unlike other floating point processors currently available where, for example, the main processor must explicitly write the floating point numbers and commands into the floating point unit. The 8086 then continues to execute non-numeric instructions until another 8087 instruction is encountered, whereupon it must wait for the 8087 to complete the previous numeric instruction. The overlapped 8086 and 8087 processing is known as concurrency. Under ideal conditions, it effectively doubles the throughput of the processor. However, even when a steady stream of numeric instructions is being executed (meaning there is no concurrency), the numeric performance of the 8087 ALU is much greater than that of the 8086 alone.

The hardware interface between the 8086 and the 8087 is equally simple. Hardware handshaking is performed through two sets of pins. The RQ/GT pin is used when the 8087 needs to transfer operands, status, or control information to or from memory. Because the 8087 can transfer information to and from memory independent of the 8086, it must be able to become the "bus master," that is, the processor with read and write control of all the address, data and status lines. Only one unit is permitted to have control of these lines at a time; chaos would exist otherwise, like four people talking at once with each trying to understand the others.

The TEST/BUSY pin is used to manage the concurrency mentioned above. Whenever the 8087 is executing an instruction, it sets the BUSY pin on high. A single 8086 instruction (the WAIT instruction) tests the state of this pin. If this pin is high, the WAIT instruction will cause the 8086 to wait until the pin is returned to low. Therefore, to insure that the 8086 does not attempt to fetch a numeric instruction while the 8087 is still working on a previous numeric instruction, the WAIT instruction needs to be executed. The 8086/87/88 assembler, in addition to all Intel compilers, automatically inserts this WAIT instruction before most numeric instructions. Software polling can be used to determine the state of the BUSY pin if hardware handshaking is not desired.

Most other lines (address, status, etc.) are connected directly in parallel between the 8086 and the 8087. An exception to this is the 8087 interrupt pin which must be routed to an external interrupt controller. An example iAPX 86/20 system is shown in Figure 1. A more complete discussion of both the handshaking protocol between the 8086 and the 8087 and the internal operation of the 8087 can be found in the application note *Getting Started With the Numeric Data Processor,* AP-113 by Bill Rash, or by consulting the numerics section of the July 1981 *iAPX 86,88 Users Manual.*

In addition to the 8087 hardware, the 8086 is also supported by Intel compilers for both Pascal and FORTRAN. Code generated by these compilers can easily be combined with code generated from the other compiler, from the Intel 8086/87/88 macro assembler or the Intel PL/M compiler. In addition, these compilers produce in line code for the 8087 when numeric operations are required. By producing in line code rather than calls to floating point routines, the software overhead of an unnecessary procedure call and return is eliminated. The combination of both hardware co-processors and software support for the iAPX 86/20 provides for greater performance of both the end product, and its development effort.

ROUTINES IMPLEMENTED

All routines implemented in this application note were written entirely in either Pascal 86 or FORTRAN 86. In addition, a FORTRAN program available from IMSL[1] for use in solving linear programs was used. In each

[1]IMSL, Inc., Sixth Floor-NBC Building, 7500 Bellaire Boulevard, Houston, Texas, 77036. (713) 722-1927.

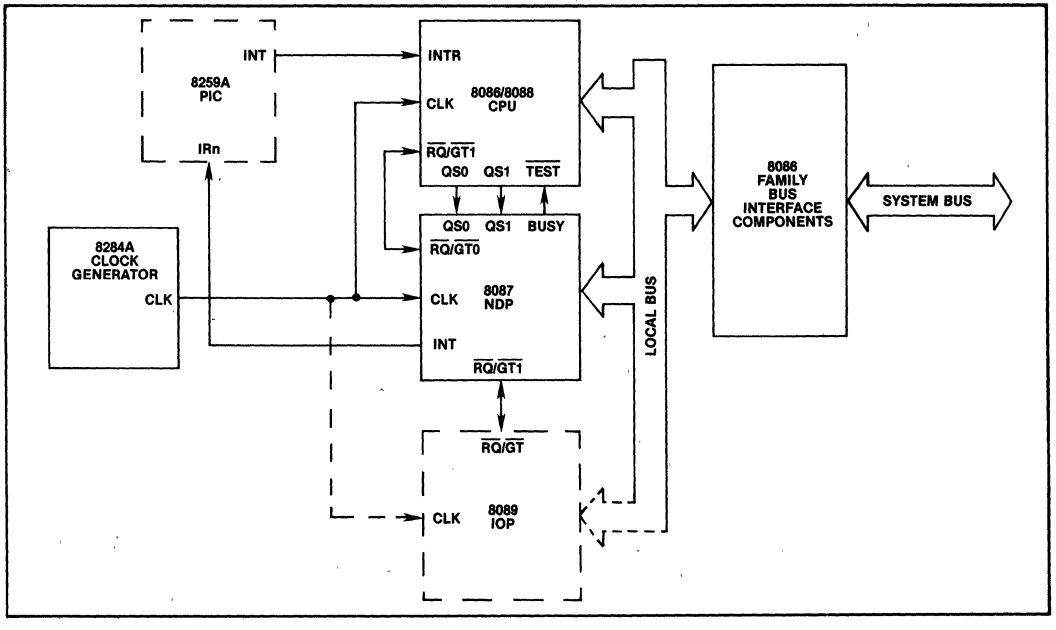


Figure 1. Typical 86/20 System

AFN-02184A

case, the routine was executed using a 5 MHz iAPX 86/20 on an iSBC86/12 board contained within an Intel Intellec™ Series III development system. The programs can be executed on any iAPX 86/20 (or iAPX 88/20) with sufficient memory, however. In general, the memory requirements for the programs were not substantial. Source listings for all routines written for this note are located in the appendix.

All routines were run using both the 8087 and the 8087 software emulator. The 8087 software emulator is a software package exactly emulating the internal operation of the 8087 using 8086 instructions. When the emulator is used, an 8087 is not required. The emulator is a software product available from Intel as part of the 8087 support library. The performance of the 8087 hardware is much better than that of the software emulator, as one would expect from a specialized floating point unit.

In some routines, values are quoted for the various data formats supported by the 8087. For real numbers, these formats are short real, long real, and temporary real. The differences among the three are in the number of bits allocated to represent a given floating point number.

In all real numbers, the data is split into three fields: the sign bit, the exponent field and the mantissa field. The sign bit indicates whether the number is positive or negative. The exponent and mantissa together provide the value of the number: the exponent providing the power of two of the number, and the mantissa providing the "normalized" value of the number. A "normalized" number is one which always lies within a certain range. By dividing a number by a certain power of two, most numbers can be made to lie between the numbers 1 and 2. The power of two by which the number must be divided to fit within this range is the exponent of the number, and the result of this division is the mantissa. This type of operation will not work on all numbers (for example, no matter what one divides zero by, the result is always zero), so the number system must allow for these certain "special cases."

As the size of the exponent grows, the range of numbers representable also grows, that is, larger and smaller numbers may be represented. As the size of the mantissa grows, the resolution of the points within this range grows. This means the distance between any two adjacent numbers decreases, or, to put it another way, finer detail may be represented. Short real numbers provide eight exponent bits and 23 significand or mantissa bits. Long real numbers provide 11 exponent bits and 52 significand bits. Temporary real numbers provide 15 exponent bits and 63 significand bits. These data formats are shown in Figure 2. Thus, of the three data formats implemented, short real provides the least amount of precision, while temporary real provides the greatest amount of precision. These levels of precision represent only the external mode of storage for the numbers; inside the 8087 all numbers are represented in temporary real precision. Numbers are automatically converted into the temporary real precision when they are loaded into the 8087. In addition to real format numbers, the 8087 automatically converts to and from external variables stored as 16, 32 or 64-bit integers, or 80-bit binary coded decimal (BCD) numbers.

Memory requirements also increase as precision increases. Whereas a short real number requires only four bytes of storage (32 bits), a long real number requires eight bytes (64 bits) and a temporary real number 10

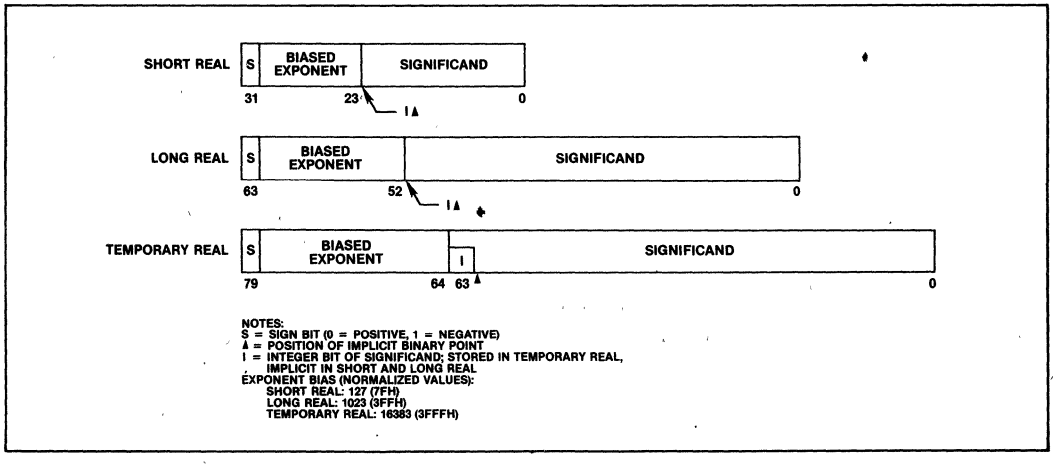


Figure 2. Data Formats

bytes (80 bits)! In many floating point processors, processing time also increases dramatically as precision is increased, making this another consideration in the choice of precision to be used by a routine. The differences in 8087 processing time among short real, long real and temporary real numbers is relatively insignificant, however. This makes the choice of which precision to use in an iAPX 86/20 system a function only of memory limitations and precision requirements.

Interest

Routines were written to calculate the final value of a fund when given the annual interest and the present value. Although the calculations required to generate individual interest values are rather short, the additional precision of the iAPX 86/20 can be used to generate better results. In addition, if a large number of interest calculations are to be performed (or if an interest rate type of calculation is used as part of an iterative model), the speed of the single interest rate calculation is important, as it will be performed very many times.

It is assumed that the interest will be compounded daily, which requires the calculation of the yearly effective rate. This value, which is the equivalent annual interest rate when interest is compounded daily, is determined by the following formula:

$$yer = (1 + (\frac{i}{np}))^{np} - 1$$

Where:

- **yer** is the yearly effective rate
- **i** is the annual interest rate
- **np** is the number of compounding periods per annum

Once the **yer** is determined, the final value of the fund can be determined by:

$$fv = (1 + yer) * pv$$

Where:

- **pv** is the present value
- **fv** is the future value

Results were obtained using short real, long real, and temporary real precision numbers when

- **ir** is set to 10% (0.1)
- **np** is set to 365 (for daily compounding)
- **pv** is set to $2,000,000

The results are shown in Table 1.

Table 1. Interest Rate Calculation Results

	yer	Final value
Short real	10.514%	$2,210,287.50
Long real	10.516%	$2,210,311.57
Temp real	10.516%	$2,210,311.57

The times required to calculate these results using FORTRAN 86 with both the 8087 and the 8087 emulator are shown in Table 2.

Table 2. Interest Rate Calculation Times

	8087	Emulator
Short real	1.052 ms	100.4 ms
Long real	1.058 ms	100.7 ms
Temp real	1.041 ms	100.8 ms

The difference in the final value between the short real and long real precision in this simple calculation is $24.07. Although the difference between short and long real precision results shown here is small, this difference would be signficant if the principal was larger, or if the period over which the interest was calculated was longer than a single year. Hence, the long real precision capability of the 8087 can provide most accurate results. Indeed, since the error calculated between the long real precision and temporary real precision results is in the thousandths of cents, the long real results are exactly correct, *to the penny*. Note that temporary real format allows for approximately *18* decimal digits of precision and the full precision of the numbers used in the calculation is not printed in the above table.

Annuities

Values for a frequently used type of annuity were calculated, using routines written in both FORTRAN and Pascal. An annuity is a type of fund which gathers interest at the same time the principal is changing. A mortgage is a type of annuity in which the principal is decreasing, whereas "the sinking fund" implemented here is a type of annuity in which the principal is increasing. In both cases, the interest is added to the principal.

THE SINKING FUND

The "sinking fund" could be characterized by an individual retirement account (IRA). In this fund, a fixed amount is placed in a savings fund each period. This fund also earns a certain amount of interest per period. The problem, then, is to calculate the final value of the

fund (after a certain number of periods). The example given calculates the value after 20 years of a fund in which payments of $1000 are made each month. The annual interest rate is given at 12% (0.12), but the interest is compounded daily.

The first step in solving the problem is to determine the interest rate per month. This is done in a similar manner to the way the effective annual rate is calculated; however, the number of compounding periods is set to the number of days in a month, rather than the number of days in a year. Once this is done, the final value of the annuity is determined by:

$$fv = pmt * \frac{((1 + irp)^{np} - 1)}{irp}$$

Where:

- **fv** is the final value
- **pmt** is the amount placed in the fund each period
- **irp** is the interest rate per period
- **np** is the number of periods

The short, long and temporary real precision results are shown in Table 3.

Table 3. Annuity Calculation Results

	Tot Contrib	Final value	Rate/period
Short	$240,000	$997,103.25	1.005%
Long	$240,000	$997,048.51	1.005%
Temp	$240,000	$997,046.51	1.005%

The times required to calculate these results using FORTRAN 86 with both the 8087 and the 8087 emulator are shown in Table 4. Notice that although the most significant four digits of the interest rates per period shown are the same, the final value using short real precision calculations is inaccurate by $56.74 compared to the final value using long or temporary real calculations.

Table 4. Annuity Calculation Times

	8087	Emulator
Short real	2.121 ms	222 ms
Long real	2.139 ms	229 ms
Temp real	2.106 ms	232 ms

Restocking Algorithms

A restocking algorithm determines when a company should replenish its stock of raw goods which make up its products. A restocking algorithm can be used to determine the restocking pattern if:

- the demand for the given product can be predicted
- carrying costs from month to month are known and fixed
- no shortages are allowed
- lead times are known and fixed

There are three methods commonly used to determine the restocking pattern:

1) the Fixed Economic Order Quantity (EOQ)
2) the Silver-Meal heuristic
3) the Wagner-Whitin method

Of the three, the Wagner-Whitin method is *guaranteed* to provide the *optional restocking pattern,* while the Silver-Meal heuristic may provide a good approximation to this pattern. The fixed Economic Order Quantity will not provide good results when the demand pattern is highly variable. Both the Wagner-Whitin method and the Silver-Meal heuristic are iterative methods in which many options are evaluated before the final restocking pattern is determined.

THE FIXED ECONOMIC ORDER QUANTITY

The simple Economic Order Quantity method may be used to select the number of items to be restocked at a time if the demand is constant. This number is determined by:

$$EQU = \sqrt{\frac{2AD}{vr}}$$

Where:

- A is the set-up cost
- D is the average demand for the period
- v is the variable demand cost per item
- r is the holding cost per item

As this method does not provide for period to period variability in demand, if this demand is variable, the performance of the method will obviously suffer. Its only advantage is simplicity.

THE SILVER-MEAL HEURISTIC

The Silver-Meal heuristic will provide an approximation to the optimal restocking pattern determined by the Wagner-Whitin method. It has been used rather than the Wagner-Whitin in application where better results were required than those supplied by the EOQ method, but where the available computing resources did not allow the use of the Wagner-Whitin method. This

method begins with the first month to be considered, then calculates the total replenishment and holding costs for this month, and a certain number of following months. As the number of months increases, the set-up charge per unit will decrease as it is distributed over more units. Also, however, as the number of units increases, the holding costs will increase. At a certain point, the holding costs will begin to increase at a greater rate than the set-up cost per unit falls. At this point, a "local minimum" of the replenishment cost function will have been realized. The heuristic stops here, and begins the process again with the following month until all the months of the period have been considered. This method may not provide the optimal solution, since it provides only a local minimum, rather than a global minimum. The cost function is not guaranteed to continue to rise once it has begun to rise. This means that the restocking cost may actually fall to a lower level after an initial rise. This method requires much fewer cost calculations than the Wagner-Whitin method, however.

THE WAGNER-WHITIN METHOD

The Wagner-Whitin method is the most computationally intensive method to be discussed. It also is guaranteed to produce the optimal results. It is an application of "dynamic programming." It starts with the last month of the period, determining in inverse order the optimal replenishment pattern for the given month if the inventory is assumed zero at the start of the month. It does this by calculating the replenishment cost for the given month and a number of subsequent months along with the holding costs for the stock replenished in the given month but carried over. The replenishment cost is the sum of the set-up charges and the per unit cost times the number of units acquired. The holding cost is the number of units held but not consumed in a given month. The total stocking costs for this option can then be determined by adding the replenishment cost, the holding cost and the optimal restocking cost for the month following the last one restocked in this iteration (since we have started from the last month of the period, the optimal restocking cost has already been determined for all months following the month being considered). The optimal restocking cost for the last month of the period is the restocking cost for that month alone. For example, if we are trying to determine the optimal restocking pattern from January through December of a year, the determination of the optimal restocking pattern for June might begin like this:

1) Determining restocking cost (startup cost, per part cost, etc.) for June alone.
2) Determine the holding costs (if June alone is being restocked, the holding cost will be zero).
3) Determine the total cost of this option. This will be the restocking cost determined in (1) added to the holding costs determined in (2) added to the optimal restocking cost from zero initial inventory determined previously (using this algorithm) for July.
4) Loop back to (1). However this time, restock for June and July, calculate the holding cost for the July stock, and use the optimal restocking cost from zero initial inventory for August.

This will continue until starting with June, requirements for the balance of the year are being restocked. As the algorithm continues, the cost of each new restocking period (that month and the number of months following it being restocked) for a particular month is compared with a previously determined minimum cost. If it is less, a new minimum cost has been determined, and this restocking pattern will replace the old one as the optimal restocking pattern for the month. As should be apparent, a "horizon" in which the stock will be known to go to zero must be determined in order for this algorithm to be used. While this may at first seem unrealistic, one can see that in any month where the demand for the product is relatively high, the stock will be allowed to go to zero, as the holding cost to that month will surpass the benefit in the restocking cost if the requirements were restocked in the previous month.

OVERALL PERFORMANCE CONSIDERATIONS

Generally, the better an algorithm is in determining an objective function, the greater the computer performance required to execute the algorithm. This is true here, with the most numeric intensive solution guaranteed to realize the optimal solution to the problem, whereas the simpler solutions will only provide approximations to this solution. A more complete explanation of these three methods can be found in Peterson and Silver[2].

EXAMPLE RESTOCKING PROBLEM

Routines were written in Pascal to show possible implementations of the Wagner-Whitin and Silver-Meal heuristic. The EOQ method's results were solved by hand and programmable calculator. The following example was used to demonstrate the results of these methods in solving a general stock management problem:

> A company manufactures video games in which a ROM programmed microcomputer

[2]Peterson, Rein, and Edward A. Silver, *Decision Systems For Inventory Management And Production Planning*, John Wiley & Sons, New York, 1979, pp 308-321.

is used. The manufacturer from which the company buys this microcomputer has an initial ROM set-up charge of $3000, with the cost per part varying from $20 in quantities of less than 500, $17.50 in quantities from 500 to 5000, and $15 in larger quantities. The holding cost is determined to be $0.40 per part. The company barely missed the Christmas rush with its introduction, but has determined that the monthly demand for the next two years will be:

Month	Demand	Month	Demand
January	500	July	3500
February	1500	August	2500
March	2500	September	5000
April	2000	October	7500
May	2000	November	9500
June	1000	December	10000

How should the company restock the microcomputers?

The first problem that must be solved (when using the Wagner-Whitin method) is the horizon to which the stock will be replenished. The criterion to be used is that the final month should be a month in which the demand in the subsequent month is relatively high. Choosing December as the final month would not produce the best results, as the requirements for January are low. Looking at the demand function, it can be seen that the requirements for September are relatively high, so August would be a good choice as the horizon month. It is assumed that the demand for the second year will be similar to the demand predicted for the first year. This allows extending the period of calculation beyond the first year up to the chosen horizon month. Given the total demand function, the part cost, the holding cost, and the startup cost, the problem may be plugged into the Wagner-Whitin, Silver-Meal and Economic Order Quantity methods, and the results calculated.

Using the EOQ with this demand function yields:

- D is 3150
- A is 3000
- v is $15.00
- r is 0.0229

This leads to an EOQ of 7418.

The results obtained from the Wagner-Whitin method, the Silver-Meal heuristic and the EOQ are shown in Table 5. The performance difference between the methods is apparent. Although using the Silver-Meal heuristic would save the business $12,949 over using the EOQ method, using the Wagner-Whitin method would save the business almost $25,000 over using the EOQ (surely below the cost of a small business computer!). The effect on the performance of the Silver-Meal heuristic of choosing a local minimum rather than a global minimum can be seen especially in the first few months in which it replenishes stock 5 times vs. 3 times for the Wagner-Whitin method. It should also be noted that the execution time of the Silver-Meal heuristic using the emulator is still greater than the execution time of the Wagner-Whitin method when the 8087 is used (and the execution time of the EOQ on the hand calculator was much greater than the execution time of either of the two iAPX 86/20 programs!). These results are also interesting when one realizes that until now the Economic Order Quantity method has been the most commonly used method of scheduling stocking intervals.

Linear Programming

Linear programming methods are very powerful ways of finding the optimal solution to operations problems. For example, if a number of different products can be made from a combination of limited resources as expressed by a set of equations, a linear program can be set up to determine the optimal number of each end product to make in order that a certain objective function is maximized. This objective function can be practically anything if it is a linear function—for example, insuring that profit is maximized, that the use of a certain facility is maximized, that shipping costs are minimized, etc. Various software packages are available on the market to solve linear programs. The package which was used in this example consisted of a set of FORTRAN subroutines available from IMSL[3]. To use the routines a FORTRAN program is written to set up the appropriate input arrays and call the routine. They could very easily be integrated into a friendly interactive user environment, where the increased performance of the 8087 would be especially apparent and welcomed.

[3]IMSL, Inc.

Table 5. Restocking Algorithm Results

Wagner-Whitin Method			Silver-Meal Heuristic		Economic Order Quantity	
Month	Number to Restock	Optimal Cost	Number to Restock	Optimal Cost	Number to Restock	Optimal Cost
1	6500	$985,200	500	$996,600	7418	$1,009,549
2	*		1500	$984,850	*	
3	*		7500	$995,600	*	
4	*		*		*	
5	6500	$879,700	*		7418	$888,810
6	*		*			
7	*		6000	$836,500	*	
8	7500	$776,000	*		7418	$769,137
9	*		5000	$742,500	*	
10	7500	$658,500	7500	$664,500	7418	$651,464
11	9500	$543,000	9500	$549,000	14836	$536,525
12	12000	$397,500	19500	$403,500	7418	$308,182
13	*		*		*	
14	*		*		*	
15	7500	$213,100	*		7418	$189,600
16	*		*		*	
17	*		*		*	
18	*		*		*	
19	6000	$94,000	6000	$94,000	3656	$67,980
20	*		*		*	
Total Hold Costs:		$16,200		$19,600		$31,409
Replenishment Costs:		$24,000		$27,000		$24,000
Times Replenished:		8		9		8
Total Cost:		$985,200		$996,600		$1,009,549
Time to calculate above values:						
Using 8087:		310 ms		20 ms		
Using emulator:		22.98 seconds		1.91 seconds		

THE SIMPLEX METHOD

The simplex method is an algorithm which may be used to solve linear programs. The problem is specified to the routine as an objective function (of a certain number of "products") and a set of constraints on the constituents of these products. The objective function specifies exactly how the products are combined to derive the objective function. The constraints specify how each of the constituents are combined to make up each of the products, and also specify the limits imposed on these various constituents.

The set of constraints is usually set up as a two-dimensional matrix, while the objective function is set up as a vector. The combination of the objective function and the set of constraining equations is known as the input tableau. The constraining equations may have both inequality relations (we must use less than 1000 eggs) and equality relations (we must use exactly 1000 eggs). The method itself requires all inequality relations to be converted to equality relations. This is done through the addition of "slack" and "surplus" variables, so called beacuse they fill up the slack or take up the surplus in an inequality relationship. Through many iterations, the method automatically reduces the inequality constraints in the original problem to equality constraints through the addition of these slack and surplus variables. "Artificial" variables are then added to the equation to form an initial set of basic variables or bases. This basis forms a feasible solution to the problem, although this solution is non-optimal. The object, however, is to find the optimal solution to the problem (the solution that optimizes the objective function). This initial form is called the *canonical* form. It transforms the original set of constraint equations and the objective function by the addition of artificial, slack and surplus variables.

After the problem has been set into canonical form, phase I of the problem is ready to begin. In this phase, "pivoting" is performed on the constraint variable matrix until all the coefficients on the modified objective function are less than zero. This pivoting operation is very similar to gaussian elimination. A certain variable in a certain row and column of the matrix is

divided by itself to become 1. Subsequently, every other variable in that row must be divided by this variable. All other variables in the column containing this variable are then eliminated by multiplying the variable set to one by the negative of the variable to be eliminated and then adding the result of this multiplication to the number being eliminated. In order for the matrix to remain valid, this operation must be performed on all other columns of the matrix as well, which leads to a large number of multiplies and divides.

Once phase I is complete, phase II must be initiated. This phase is required if any of the artificial variables remain in the solution as a basis. Through another round of pivoting, the remaining artificial variables are removed from the solution. What finally comes out is the optimal mix of the input variables so the objective function is maximized. A more complete description of both the simplex method and the revised simplex method can be found in Bradley, Hax, and Magnanti[4].

ROUTINE IMPLEMENTED

The linear program used in this example is the IMSL[5] routine "ZX3LP." This routine is the so-called "easy-to-use" linear program solver. It solves the linear program using the revised simplex method. On output, it provides not only the solution to the problem, but also what is called the dual solution. The dual solution gives information about how the solution could be enhanced. The objective function is input to the routine as a vector, while the constraining equations are input to the routine as a matrix. Both inequality and equality constraining equations may be used; the routine will automatically insert slack and surplus variables. The outputs of the routine are two vectors containing the "primal" solution and the dual solution. The routine also calculates the optimal value of the objective function. The version of the routine used was originally developed for the IBM 370/3033 mainframe computer. It required *no* modifications to run on the iAPX 86/20 using FORTRAN 86.

EXAMPLE PROBLEM

The following problem was input to the linear program routine:

> A small cookie company has four different products: chocolate chip cookies without walnuts, chocolate chip cookies with walnuts, brownies without walnuts, and brownies with walnuts. The recipes for the four are:

Chocolate Chip Cookies	Brownies
2 eggs	4 eggs
⅔ cup shortening	⅔ cup shortening
1 cup sugar	2 cups sugar
1 cup brown sugar	
1 tsp. vanilla	1 tsp. vanilla
2¼ cup flour	1¼ cup flour
1 tsp. baking soda	
	1 tsp. baking powder
1 tsp. salt	1 tsp. salt
12 oz. chocolate chips	
	4 oz baking chocolate
(1½ cup walnuts)	(⅔ cup walnuts)
0.15 hour oven time	0.5 hour oven time
0.25 hr mix time (w/o nuts)	0.25 hr mix time (w/o nuts)
0.45 hr mix time (w/nuts)	0.45 hr mix time (w/nuts)

> The available amounts of many of the ingredients have been set previously by contract and may not be altered. They are:

Item	Quantity
eggs	1000
sugar	600 cups
brown sugar	20 cups
baking chocolate	700 oz.
flour	600 subs
baking soda	150 tsp.
baking powder	150 tsp.
chocolate chips	1500 oz.
walnuts	125 cups
oven time	560 hours
mixing time	750 hours

> The amount of profit made for each type cookie is:

Cookie Type	Profit per Batch
chocolate chip w/o	$0.85
chocolate chip with	$0.95
brownies w/o	$1.10
brownies with	$1.25

> It is assumed that the cookie company can sell everything that it makes. How many of each kind of cookie should the company make in order that the profit is maximized?

The problem was set up into the input tableau. The objective function is:

$$Y = .85*X_1 + .95*X_2 + 1.1*X_3 + 1.25*X_4$$

[4]Stephen P. Bradley, Hax, Arnoldo C., and Magnanti, Thomas L., *Applied Mathematical Programming,* Addison-Wesley, Reading, Massachusetts, 1977.

[5]IMSL, Inc.

Table 6. Example Problem Input Tableau

$2X_1$	+ $2X_2$	+ $4X_3$	+ $4X_4$	=	1000	(eggs)
X_1	+ X_2	+ $2X_3$	+ $2X_4$	=	600	(sugar)
X_1	+ X_2			=	200	(b. sugar)
		$4X_3$	+ $4X_4$	=	700	(b. choc)
$2.25X_1$	+ $2.25X_2$	+ $1.25X_3$	+ $1.25X_4$	=	600	(flour)
X_1	+ X_2			=	150	(b. soda)
		X_3	+ X_4	=	150	(b. powder)
$12X_1$	+ $12X_2$			=	1500	(c. chips)
	$.5X_2$	+ $.65X_4$		=	125	(walnuts)
$.15X_1$	+ $.15X_2$	+ $.5X_3$	+ $.5X_4$	=	560	(oven time)
$.25X_1$	+ $.45X_2$	+ $.25X_3$	+ $.45X_4$	=	750	(mix time)

Where the variable X_1 is the number of batches of chocolate chip cookies without nuts, X_2 is the number of batches of chocolate chip cookies with nuts, X_3 is the number of batches of brownies without nuts, and X_4 is the number of batches of brownies with nuts. The input tableau is shown in Table 6. These were put into the proper input matricies of the ZX3LP program, and the following results were generated:

profit	$299.25
batches of choc chips w/o	70
batches of choc chips with	55
batches of brownies w/o	0
batches of brownies with	150

In addition, the dual solution shows that the single ingredient most limiting the profit of the cookie company is the availability of baking powder, and that for every additional unit (teaspoon) of baking powder available, the profit of the company will increase 1.12 cents.

The calculation times are:

with 8087	1.01 seconds
with emulator	46.78 seconds
with PDP11/45	0.7 seconds
with IBM 3033[6]	0.07 seconds

The results show that the performance of the iAPX 86/20 is close to the performance of the mini-computer. In addition, the performance is only a little more than an order of magnitude below the performance of the IBM mainframe, a "maxi" computer with an execution rate of 5 MIPS, and a CPU/hour cost of around $800! A comparison of results between the iAPX 86/20 and the emulator verifies the speed of the 8087 is required to provide results in a reasonable period of time. The power and ease of use of this type of sophisticated numerical method combined with an "electronic worksheet" type of program could be a major advance in the "state of the art" of small business machine software.

CONCLUSIONS

The types of routines demonstrated in this note show that there are many classes of numeric intensive software which are (or should be) commonly used in everyday business operations. With the introduction of the iAPX 86/20, these types of applications are finally within the performance limits of microcomputers selling for a fraction of the cost of the previously required mini- or maxi-computers. In addition, the availability of both Pascal and FORTRAN compilers for the iAPX 86/20 eases the problem of software generation and availability for the processor. Because of the portable nature of these high-level languages, a minimum of effort is required to generate or to port software to the iAPX 86/20 from existing systems. With this kind of numeric intensive software support, the 8087 will be an essential part of the next generation of small business computers.

[6]Non-Intel computers used were a PDP 11/45 mini-computer with 256K MOS RAM, and a FP11-B floating point unit running the UNIX operating system during a period of light load. The program was compiled using the UNIX F77 FORTRAN compiler, and an IBM 370/3033 mainframe computer running the VM/CMS operating system during a period of medium load (the program, however, did not get swapped out of memory during execution). The IBM FORTRAN G compiler was used.

APPENDIX A

Contents

PAGE

```
FORTRAN-86 COMPILER
:F6:INTST.FOR

SERIES-III FORTRAN-86 COMPILER X023
COMPILER INVOKED BY: FORT86.86  :F6:INTST.FOR

      c
      c       this program provides the yearly effective rate(double and
      c       single precision) and final value when the interest rate
      c       (ir), the number of compounding periods (np),
      c       the present value (pv) are specified.
      c
  1           real    pv,ir,fv,yer
  2           real*8  fvd,yerd
  3           tempreal fvt,yert
  4           integer*2 np,csv
  5           integer*4 count,rtimer
      c
      c $2,000,000., at an interest rate of 10% with daily compounding for 1 year
      c
  6           pv=2000000.
  7           ir=.1
  8           np=365
      c
      c set rounding control to single precision
      c
  9           call stcw87(csv)
 10           csv=csv .and. #fcffh
 11           call ldcw87(csv)
      c
 12           yer=(1+(ir/np))**np - 1
 13           fv=(1 + yer)*pv
      c
      c set rounding control to double precision
      c
 14           csv=csv .or. #200h
 15           call ldcw87(csv)
      c
 16           yerd=(1+(ir/np))**np - 1
 17           fvd=(1 + yerd)*pv
      c
      c set rounding control to temp real precision
      c
 18           csv=csv .or. #100h
 19           call ldcw87(csv)
      c
 20           yert=(1+(ir/np))**np - 1
 21           fvt=(1 + yert)*pv
      c
      c print results
      c
 22           print *,'single precision: yer=',yer,'fv=',fv
 23           print *,'double precision: yer=',yerd,'fv=',fvd
 24           print *,'temp real precision: yer=',yert,'fv=',fvt
 25           stop
 26           end
```

```
SERIES-III Pascal-86, V1.1


Source File: :F1:ANNP1.PAS
Object File: :F1:ANNP1.OBJ
Controls Specified:  CODE.

              SOURCE TEXT: :F1:ANNP1.PAS
              (*  ANNUITIES: type 1, the sinking fund
               *  if one were to place $1000 a month into a savings fund which
               *  earns 12% per annum, compounded daily, what will be the value
               *  of the fund after 20 years???
               *)
              module annuity;
              public cel;
                  function mqery2x(y,x: real):real;  (* takes y to the x *)
              program annuity(input,output);

              var
                  ir,      (* the annual interest rate *)
                  fv,      (* the final value *)
                  pmt,     (* the amount of the payment *)
                  irp:     (* the interest rate per period *)
                       real;
                  np:      (* the number of periods *)
                      integer;

              begin
              (* insert calculation values *)
                  ir := 0.12;
                  pmt := 1000;

                  np := 12 * 20;       (* 20 years of months *)

              (* calculate the effective interest rate per period *)
                  irp := mqery2x((1+(ir/365.0)),365.0/12.0)-1;
              (* effective monthly rate *)
              (* calculate the future value *)
                  fv := pmt * (mqery2x((1+irp),np)-1)/irp;

              (* print results *)
                  writeln('the effective monthly rate is',irp:18);
                  writeln('the future value of the annuity is',fv:12:2);
                  writeln('the total contribution to the annuity is',np*pmt:12:2);
              end.
```

```
FORTRAN-86 COMPILER
:F1:ANNU1.FOR

SERIES-III FORTRAN-86 COMPILER X023
COMPILER INVOKED BY: FORT86.86  :F1:ANNU1.FOR

      c
      c  ANNUITIES: type 1, the sinking fund
      c       if you place in a savings fund $1000.00 a month, and it
      c       earns an interest rate of 12% per annum compounded daily,
      c       what will be the value of the fund after 20 years?
      c
  1           real ir,pv,fv,pmt,irp
  2           real*8 fvd,irpd
  3           tempreal fvt,irpt
  4           integer*2 cwv
  5           integer np

  6           ir = .12
  7           pmt = 1000.
      c
      c the number of periods is the number of months in 20 years!(one period
      c       is one month
      c
  8           np = 20*12
      c
      c set the 8087 to single precision mode
      c
  9           call stcw87(cwv)
 10           cwv=cwv .and. #fcffh
 11           call ldcw87(cwv)
      c
      c first calculate the effective interest rate per period
      c
 12           irp = (1+(ir/365.))**(365./12.) - 1
      c
      c then calculate the future value
      c
 13           fv = pmt * ((1 +irp)**np - 1)/irp
      c
 14           print *,'single precision values:'
 15           print *,'the effective rate per month is',irp
 16           write(6,800)fv
 17           write(6,801)np*pmt
 18   800     format('the future value of the annuity is',f18.2)
 19   801     format('the total contribution to the annuity is',f18.2)
      c
      c set the 8087 to double precision mode
      c
 20           cwv=cwv .or. #200h
 21           call ldcw87(cwv)
      c
      c first calculate the effective interest rate per period
      c
 22           irpd = (1+(ir/365.))**(365d0/12d0) - 1
      c
      c then calculate the future value
      c
 23           fvd = pmt * ((1 +irpd)**np - 1)/irpd
      c
 24           print *,'double precision values:'
```

```
FORTRAN-86 COMPILER
:F1:ANNU1.FOR

  25            print *,'the effective rate per month is',irpd
  26            write(6,800)fvd
  27            write(6,801)np*pmt
        c
        c set the 8087 to temp real precision mode
        c
  28            cwv=cwv .or. #100h
  29            call ldcw87(cwv)
        c
        c first calculate the effective interest rate per period
        c
  30            irpt = (1+(ir/365.))**(365t0/12t0) - 1
        c
        c then calculate the future value
        c
  31            fvt = pmt * ((1 +irpt)**np - 1)/irpt
        c
  32            print *,'temp real precision values:'
  33            print *,'the effective rate per month is',irpt
  34            write(6,800)fvt
  35            write(6,801)np*pmt
  36            stop
  37            end
```

```
SERIES-III Pascal-86, V1.1


Source File: :F6:SMCT.PAS
Object File: :F6:SMCT.OBJ
Controls Specified: <none>.

            SOURCE TEXT: :F6:SMCT.PAS
            (* This is going to try to find the optimal replacement cost
             *  for a rather variable demand product over 20 months, when
             *  the demand is known, an example could be a video game, using
             *  a single chip ROM programmed microcomputer with an initial set
             *  up charge of $3000.00, demand varies a lot with peak in october
             *  and november(for Christmas), droops in may(vacations), etc.
             *  The cost per part varies from $20.00 per part up to 500,
             *  $17.50 per part from 500 to 5000, and $15.00 above 5,000.
             *  The Sliver-Meal heuristic is going to be used.
             *)
            module silver_meal;
            public timers;
                function rtimer:integer;
                procedure stimer;
            program silver_meal(input,output);
            const
                months = 20;
                monthspl = 21;
                setupcost = 3000.0;
                holdcost = 0.4;
                reallarge = 1.0e10;
                reallargei = 32000;
            var
                repl:           (* first time stock goes to 0 for a given month *)
                    array[1..months] of integer;
                tomake,         (* the number of boxes to make in a month *)
                require:        (* number of boxes required in a given month *)
                    array[1..monthspl] of real;
                trcut,
                holdcostv:      (* holding costs *)
                    array[1..months] of real;
                cost,           (* calculated cost in a given situation *)
                cost1,          (* production cost *)
                cost2,          (* holding cost *)
                totalcost,      (* the total cost of it all *)
                lastcost,       (* used in determining the total cost *)
                totalholdcost:  (* the total hold cost *)
                    real;
                i,j,k:      (* counters *)
                    integer;
                totcnt,         (* accumulated number of boxes in a batch *)
                holdcnt:        (* number of boxed holding *)
                    real;
                count:          (* the 10 ms count *)
                    integer;

            begin
                require[1] := 500;
                require[2] := 1500;
                require[3] := 2500;
                require[4] := 2000;
```

```
SOURCE TEXT: :F6:SMCT.PAS
    require[5] := 2000;
    require[6] := 1000;
    require[7] := 3500;
    require[8] := 2500;
    require[9] := 5000;
    require[10] := 7500;
    require[11] := 9500;
    require[12] := 10000;
    require[13] := 500;
    require[14] := 1500;
    require[15] := 2500;
    require[16] := 2000;
    require[17] := 2000;
    require[18] := 1000;
    require[19] := 3500;
    require[20] := 2500;  (* stop here, because the next month is much
                             higher can assume will restock then *)
    require[monthspl] := reallargei;

    stimer;                          (* start the timer *)

    i := 1;
    while i <= months do begin       (* i is the month working on *)
        trcut[i] := reallarge;
        totcnt := 0;

        j := i;
        while j <= monthspl do begin
            totcnt := totcnt + require[j];
            if totcnt < 500 then cost1 := 20.0 * totcnt
            else if totcnt < 5000 then cost1 := 17.5 * totcnt
            else cost1 := 15.0 * totcnt;
            cost2 := 0.0;
            holdcnt := totcnt;
            for k := i to j - 1 do begin
                holdcnt := holdcnt - require[k];
                cost2 := cost2 + holdcnt * holdcost;
            end;
            cost := (setupcost + cost2 + cost1)/(j - i + 1);
            if cost < trcut[i] then begin
                trcut[i] := cost;
                tomake[i] := totcnt;
                holdcostv[i] := cost2;
            end
            else begin
                repl[i] := j;
                i := j;
                j := monthspl;
            end;
            j := j + 1;
        end;
    end;

    count := rtimer;
    j := 1;
```

```
SERIES-III Pascal-86, V1.1


            SOURCE TEXT: :F6:SMCT.PAS
                writeln('month  restock#  optimal cost per period');
                totalcost := 0;
                for i := 1 to months do begin
                    if i = j then begin
                        write(i:5,'  ',tomake[i]:6,'    ',trcut[i]:10:2);
                        writeln(' * restocking now');
                        j := repl[j];
                        lastcost := trcut[i];
                        totalcost := totalcost + lastcost;
                    end
                    else begin
                        totalcost := totalcost + lastcost;
                        writeln(i:5);
                    end;
                end;
                i := 1;
                j := 0;
                totalholdcost := 0.0;
                while i <= months do begin
                    totalholdcost := totalholdcost + holdcostv[i];
                    j := j + 1;
                    i := repl[i];
                end;
                writeln('the total hold cost is',totalholdcost:12:2);
                writeln('stock gets replenished',j:4,' times');
                writeln('replenishment cost is',j*setupcost:12:2);
                writeln('the total cost thingy is',totalcost);
                writeln('the 10 ms count is ',count);
            end.


Summary Information:

PROCEDURE                OFFSET    CODE SIZE     DATA SIZE    STACK SIZE
SILVER_MEAL               0108H  05F7H  1527D  01ACH   428D  000EH    14D

Total                            06FFH  1791D  01ACH   428D  0042H    66D

 135 Lines Read.
   0 Errors Detected.
 41% Utilization of Memory.
```

```
SERIES-III Pascal-86, V1.1


Source File: :F6:WAGCT.PAS
Object File: :F6:WAGCT.OBJ
Controls Specified: <none>.

            SOURCE TEXT: :F6:WAGCT.PAS
            (* This is going to try to find the optimal replacement cost
             *  for a rather variable demand product over 20 months, when
             *  the demand is known, an example could be a video game, using
             *  a single chip ROM programmed microcomputer with an initial set
             *  up charge of $3000.00, demand varies a lot with peak in october
             *  and november(for Christmas), droops in may(vacations), etc.
             *  The cost per part varies from $20.00 per part up to 500,
             *  $17.50 per part from 500 to 5000, and $15.00 above 5,000.
             *)
            module wag_with;
            public timers;
                function rtimer:integer;
                procedure stimer;
            program wag_with(input,output);
            const
                months = 20;
                monthspl = 21;
                setupcost = 3000.00;     (* mask set up charge *)
                holdcost = 0.4;          (* cost per part of maintaining inventory*)
                reallarge = 1.0e9;
            var
                require,          (* number of chips required in a given month *)
                tomake:           (* the number of chips to make in a month *)
                    array[1..months] of real;
                repl:             (* first time stock goes to 0 for a given month *)
                    array[1..months] of integer;
                optwz:            (* optimum cost for a given month with zero stock
                                     to start with *)
                    array[1..monthspl] of real;
                holdcostv:        (* holding costs *)
                    array[1..months] of real;
                cost,             (* calculated cost in a given situation *)
                cost1,            (* production cost *)
                cost2,            (* holding cost *)
                totalcost,        (* the total cost of it all *)
                totalholdcost:    (* the total hold cost *)
                    real;
                i,j,k:       (* counters *)
                    integer;
                totcnt,           (* accumulated number of chips in a batch *)
                holdcnt:          (* number of boxed holding *)
                    real;
                count:            (* 10 ms count *)
                    integer;

            begin
                optwz[monthspl] := 0;
                require[1] := 500;
                require[2] := 1500;
                require[3] := 2500;
                require[4] := 2000;
```

SERIES-III Pascal-86, V1.1

SOURCE TEXT: :F6:WAGCT.PAS

```
    require[5] := 2000;
    require[6] := 1000;
    require[7] := 3500;
    require[8] := 2500;
    require[9] := 5000;
    require[10] := 7500;
    require[11] := 9500;
    require[12] := 10000;
    require[13] := 500;
    require[14] := 1500;
    require[15] := 2500;
    require[16] := 2000;
    require[17] := 2000;
    require[18] := 1000;
    require[19] := 3500;
    require[20] := 2500;    (* stop here, because the next month is much
                               higher can assume will restock then *)

    stimer;
    for i := months downto 1 do begin   (* i is the month working on *)
        optwz[i] := reallarge;
        totcnt := 0;
        for j := i to months do begin   (* j is the option working on *)
            totcnt := totcnt + require[j];
            cost1 := setupcost+optwz[j+1];
            if totcnt <= 500 then cost1 := cost1 + 20.0*totcnt
            else if totcnt <= 5000 then cost1 := cost1 + 17.5*totcnt
            else cost1 := cost1 + 15.0*totcnt;
            cost2 := 0.0;
            holdcnt := totcnt;
            for k := i to j - 1 do begin
                holdcnt := holdcnt - require[k];
                cost2 := cost2 + holdcnt * holdcost;
            end;
            cost := cost1 + cost2;
            if cost < optwz[i] then begin
                optwz[i] := cost;
                repl[i] := j + 1;
                tomake[i] := totcnt;
                holdcostv[i] := cost2;
            end;
        end;
    end;
    count := rtimer;

    j := 1;
    writeln('month  restock#  optimal cost');
    for i := 1 to months do begin
        write(i:5,'  ',tomake[i]:6,'     ',optwz[i]:10:2);
        if i = j then begin
            writeln(' * restocking now');
            j := repl[j];
        end
        else writeln;
    end;
```

```
SERIES-III Pascal-86, V1.1


            SOURCE TEXT: :F6:WAGCT.PAS
                i := 1;
                j := 0;
                totalholdcost := 0.0;
                while i <= months do begin
                    totalholdcost := totalholdcost + holdcostv[i];
                    j := j + 1;
                    i := repl[i];
                end;
                writeln('the total hold cost is',totalholdcost:12:2);
                writeln('stock gets replenished',j:4,' times');
                writeln('replenishment cost is',j*setupcost:12:2);
                writeln('the 10 ms count is ',count);
            end.


Summary Information:

PROCEDURE                OFFSET    CODE SIZE     DATA SIZE    STACK SIZE
WAG_WITH                  00E5H  0576H  1398D  01A8H   424D  000EH    14D

Total                            065BH  1627D  01A8H   424D  0042H    66D

 119 Lines Read.
   0 Errors Detected.
 41% Utilization of Memory.
```

```
FORTRAN-86 COMPILER
:F1:COOKIE.FOR

SERIES-III FORTRAN-86 COMPILER X023
COMPILER INVOKED BY: FORT86.86  :F1:COOKIE.FOR

      c
      c this routine will solve a linear problem using the IMSL fortran
      c       library.  the IMSL routine used is "zx3lp" which solves the problem
      c       using the revised simplex method.
      c
   1          integer ia,n,ml,m2,iw(37),ier
   2          real*8 a(13,4),b(13),c(4),rw(206),psol(11),dsol(13),s
   3          integer*4 rtimer,count

   4         data a/2.,1.,1.,0.,2.25,1.,0.,12.,0.,.15,.25,0.,0.,
     *                2.,1.,1.,0.,2.25,1.,0.,12.,.5,.15,.45,0.,0.,
     *                4.,2.,0.,4.,1.25,0.,1.,0.,0.,.5,.25,0.,0.,
     *                4.,2.,0.,4.,1.25,0.,1.,0.,.65,.5,.45,0.,0./

   5         data b/1000.,600.,200.,700.,600.,150.,150.,1500.,125.,560.,750.,0.,0./
   6         data c/.85,.95,1.10,1.25/
      c
      c       n is the number of variables
      c       ml is the number of inequality constraints
      c       m2 is the number of equality constraints
      c       ia is the declared number of columns of a
      c
   7          ml = 11
   8          m2 = 0
   9          n = 4
  10          ia = 13

  11          print *,'the input tableau:'
  12          do 100 i=1,ia-2
  13          write(6,800)a(i,1),a(i,2),a(i,3),a(i,4),b(i)
  14   800    format(4f10.4,' <= ',f10.4)
  15   100    continue

  16          call stimer
  17          call zx3lp(a,ia,b,c,n,ml,m2,s,psol,dsol,rw,iw,ier)
  18          count = rtimer()

  19          print *,'ier = ',ier
  20          print *,'the final value of the objective function(profit!) is:',s
  21          print *,'batches of chocolate chip w/o walnuts:',psol(1)
  22          print *,'batches of chocolate chip with walnuts:',psol(2)
  23          print *,'batches of brownies without walnuts:',psol(3)
  24          print *,'batches of brownies with walnuts:',psol(4)
  25          print *,'the dual solutions follow:'
  26          do 200 i=1,ia-2
  27          print *,'var',i,' = ',dsol(i)
  28   200    continue
  29          print *,'the calculation time here (in seconds...) is: ',count/100.
  30          stop
  31          end
```

intel

APPLICATION NOTE

AP-144

October 1983

Three Dimensional Graphics Application of the iAPX 86/20 Numeric Data Processor

Ken Shoemaker
Microcomputer Applications

Order Number 210384-001

INTRODUCTION

As the performance of microcomputers has improved, these machines have been used in many applications. With the introduction of 16-bit microprocessors (along with the associated CPU enhancements, especially the integer multiply instruction) the operations required to manipulate graphic representations of three-dimensional objects were made easier. Only integer values could be used to define figures, however, because only integer multiplies were supported in hardware. While software floating point routines existed, the speed at which a general purpose microprocessor could execute even the simplest floating point operation precluded the use of these routines because of the number of floating point operations which must be performed to manipulate all but the simplest of objects.

The lack of high performance floating point math or the restriction of using only integer representations severely limits the types and sizes of objects that can be defined. Imagine limiting everything in the universe to be less than 32,000 millimeters long, high, or wide! This limitation could severely impact any system that is used to model real world objects. An example of such an application is a Computer Aided Design (CAD) system. If real or floating point numbers are used, however, practically any object can be defined (after all, there are only 9,397,728,000,000,000,000 millimeters in a light year(!), well within the range of floating point numbers). With the introduction of the iAPX 86/20, the performance required to execute the requisite operations on floating point representations of three-dimensional figures has finally been achieved in a microprocessor solution, at a microprocessor price.

The iAPX 86/20 features the Intel 8086 with the 8087 numerics co-processor. This combination allows for high performance, high precision numeric operations. This performance is especially important in the graphics routines implemented in this note because of the large number of floating point operations performed for each line drawn. In addition, the precision is required to maintain the image quality of the represented figures.

This application note shows the fundamental components of a three-dimensional graphics package. As stated before, if the objects are to be described in real size, floating point values must be used. Since the operations performed require many multiplies and divides, a high performance floating point arithmetic unit is a must. Note that the operations to be performed by this software are not those of a "bit map" controller: single chip devices performing this specialized task are or will soon be available. Because they are special-purpose devices, they can also execute this task quickly, offloading the task from the general purpose microprocessor allowing the processor to perform other work in parallel. In addition, since the size of the memory used in a bit-mapped controller is constrained (one could hardly have unlimited memory for the refresh map), only integer math is required. This graphics package is a much higher level type of routine, where the inputs are three-dimensional line drawing commands (which could be fed into a bit map controller).

The three-dimensional graphics package implemented in this note allows for the entry of three-dimensional figures, the manipulation of these figures, the setting of the viewer's location, the size of the picture to be seen, and the position of the picture on the graphics output device. Along the way, it performs perspective transformations, window clipping and projection. All figures are defined using floating point numbers. Thus, any figure may be defined "real size" without pre-scaling. This means that the size of the figure defined within the package may be the actual size of the object, i.e. the size of the object is not arbitrarily limited by the machine, whether the object be a sub-nuclear particle, or a cellestial body.

iAPX 86/20 HARDWARE OVERVIEW

The iAPX 86/20 is a 16-bit microprocessor based on the Intel 8086 CPU. The 8086 CPU features eight internal general purpose 16-bit registers, memory segmentation, and many other features allowing for compact, efficient code generation from high-level language compilers. When augmented with the 8087, it becomes a vehicle for high-speed numerics processing. The 8087 adds eight 80-bit internal floating point registers, and a floating point arithmetic logic unit (ALU) which can speed floating point operations by up to 100 times over other software floating point simulators or emulators.

The 8086 and 8087 execute a single instruction stream. The 8087 monitors this stream for numeric instructions. When a numeric instruction is decoded, the 8086 generates any needed memory addresses for the 8087. The 8087 then begins instruction execution automatically. No other software interface is required, unlike other floating point processors currently available where, for example, the main processor must explicitly write the floating point numbers and commands into the floating point unit. The 8086 then continues to execute non-numeric instructions until another 8087 instruction is encountered, whereupon it must wait for the 8087 to complete the previous numeric instruction. The parallel 8086 and 8087 processing is known as concurrency. Under ideal conditions, it effectively doubles the throughput of the processor. However, even when a steady stream of numeric insructions is being executed (meaning there is no concurrency), the numeric perfor-

mance of the 8087 ALU is much greater than that of the 8086 alone.

The hardware interface between the 8086 and the 8087 is equally simple. Hardware handshaking is performed through two sets of pins. The RQ/GT pin is used when the 8087 needs to transfer operands, status, or control information to or from memory. Because the 8087 can access memory independently of the 8086, it must be able to become the "bus master," that is, the processor with read and write control of all the address, data and status lines.

The TEST/BUSY pin is used to manage the concurrency mentioned above. Whenever the 8087 is executing an instruction, it sets the BUSY pin high. A single 8086 instruction (the WAIT instruction) tests the state of this pin. If this pin is high, the WAIT instruction will cause the 8086 to wait until the pin is returned low. Therefore, to insure that the 8086 does not attempt to fetch a numeric instruction while the 8087 is still working on a previous numeric instruction, the WAIT instruction needs to precede most numeric instructions (the only class of instructions which do not need to be preceded by a WAIT instruction are those which access the control registers of the 8087). The 8086/87/88 assembler, in addition to all INTEL compilers, automatically inserts this WAIT instruction before most numeric instructions. Software polling can be used to determine the state of the BUSY pin if the hardware handshaking is not desired.

Most other lines (address, status, etc.) are connected directly in parallel between the 8086 and the 8087. An exception to this is the 8087 interrupt pin. This signal must be routed to an external interrupt controller. An example iAPX 86/20 system is shown in Figure 1. A more complete discussion of both the handshaking protocol between the 8086 and the 8087 and the internal operation of the 8087 can be found in the application note *Getting Started With the Numeric Data Processor,* Ap Note #113 by Bill Rash, or by consulting the numerics section of the July 1981 *iAPX 86, 88 Users Manual.*

In addition to the 8087 hardware, the 8086 is also supported by Intel compilers for both Pascal and FORTRAN. Code generated by these compilers can easily be combined with code generated from the other compiler, from the Intel 8086/87/88 macro assembler, or the Intel PL/M compiler. In addition, these compilers produce in-line code for the 8087 when numeric operations are required. By producing in-line code rather than calls to floating point routines, the software overhead of an unnecessary procedure call and return is eliminated.

The combination of both hardware co-processors and software support for the iAPX 86/20 provides for greater performance of the end product, and a quicker, easier development effort.

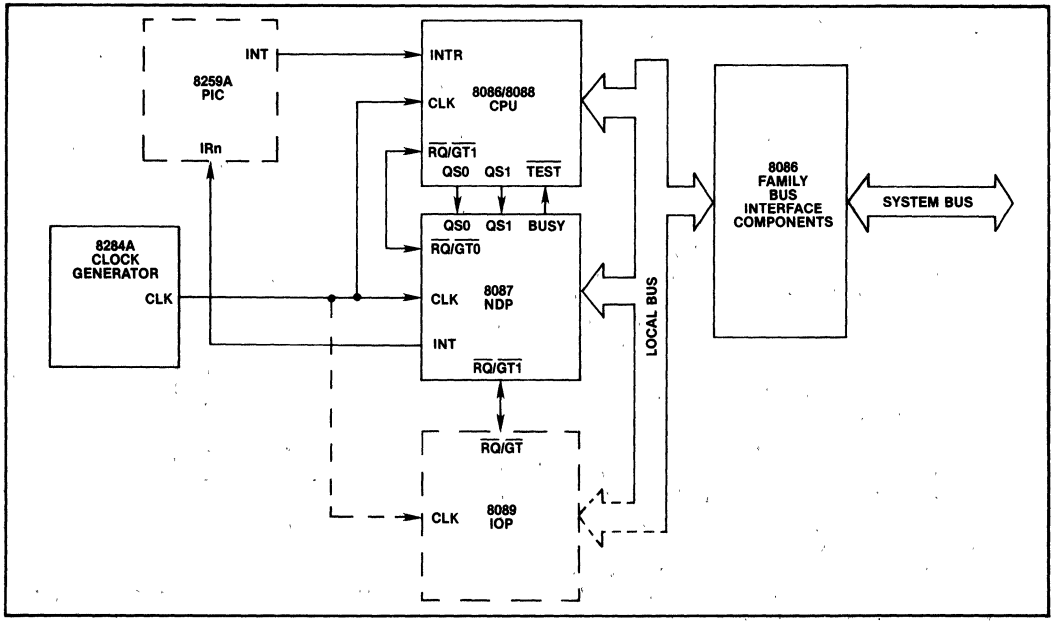


Figure 1. Example 86/20 System

AFN-02185A

THREE-DIMENSIONAL GRAPHICS FUNDAMENTALS

The charter in life of a three-dimensional graphics package is to take a three-dimensional rendering of an object and to transform it such that it can be accurately represented on the two-dimensional surface of a graphics output device. To fulfill these requirements, the graphics package must:

- **Allow for the entry of three-dimensional data.** Since all figures inside the package are represented as a series of points in three-dimensional space, there must be a way of entering these figures into the computer.
- **Perform the current transformation.** This transformation rotates, translates and scales the three-dimensional object throughout three-dimensional space. Example rotates, translates and scales are shown in Figures 2–11. In all diagrams, the first coordinate indicated is X, the second Y, the third Z. The viewpoint is the location of the viewer in three-dimensional space in relationship to an arbitrarily chosen but consistent origin.

  Translations are movements of the object in three-dimensional space. Example translations are shown in Figures 3–5. Figure 3 shows a translation of two units in the plus Z direction. Since the viewpoint is ten units up along the Z axis, this moves the cube one-fifth the distance toward the viewer, or in other words, the cube seems to get larger. Figure 4 shows the same cube translated two units in the plus X direction. Since the cube is four units on a side, this moves the cube such that the viewer is looking straight down one side of the cube. The viewer is also looking straight down a side in Figure 5.

  Rotations are movements of the object in three-dimensional space about the three-coordinate axis: X, Y, and Z. The rotation of the object must specify both the magnitude of the rotation, and the axis about which the rotation must take place. Example rotates are shown in Figures 6–8. Figure 6 shows the cube rotated 45 degrees about the Z axis. Since the viewpoint is straight up the Z axis, the cube is seen to keep its same face towards the viewer. Figure 7 shows the cube rotated 45 degrees about the X axis. Here, the cube no longer shows the same face it has previously. The face previously turned directly toward the viewer has been rotated such that the edge between this face and another face is immediately before the viewer. The same is also shown in the rotation about the Y axis in Figure 8.

Scaling is the multiplication of all coordinates of the points defining a figure by a constant number such that the object becomes larger or smaller. Example scales are shown in Figures 9–11. This scaling need not be uniformly performed for all dimensions of an object. If, for example, the Z coordinates of a cube are all scaled to be twice as large as they originally were, the image shown in Figure 9 would be produced. Notice here that the X and Y coordinates have not been altered; only the Z coordinates are twice as large as they originally were, or alternatively, the front and back of the cube are closer and farther away from the viewer than in the original, unaltered cube. Figure 10 shows this same operation being performed on the X coordinates, while Figure 11 shows this operation being performed on Y coordinates.

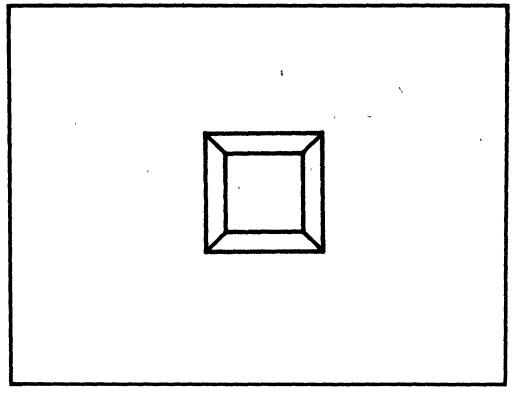


Figure 2. 2 x 2 x 2 Cube Centered at (0,0,0) Viewed from (0,0,10)

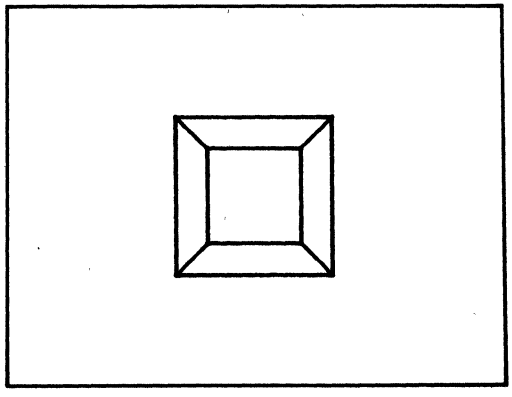


Figure 3. Same Cube and Viewpoint, +2 Z Translation

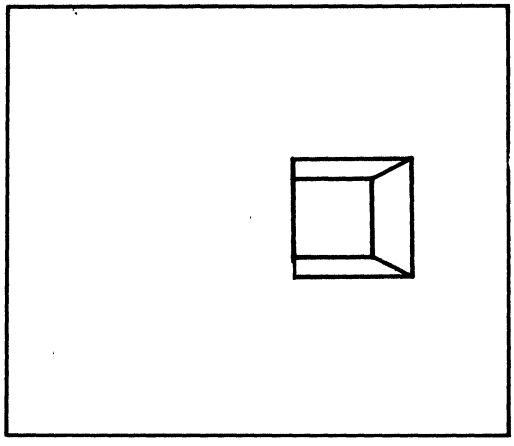


Figure 4. Same Cube, Viewpoint, +2 X Translate

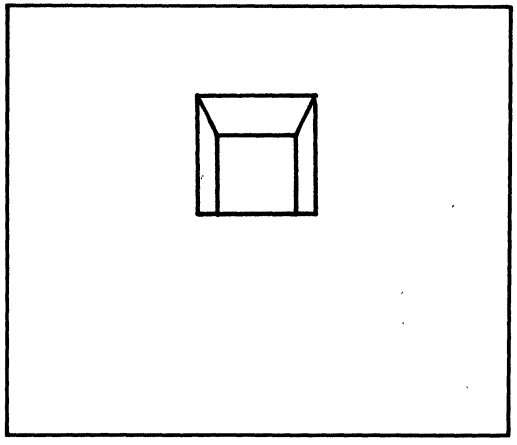


Figure 5. Same Cube, Viewpoint, +2 Y Translate

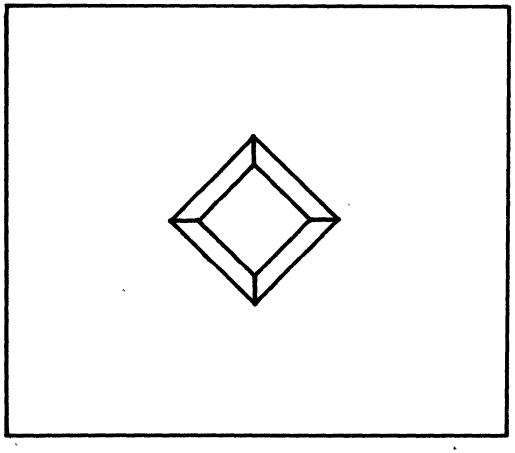


Figure 6. Same Cube, Viewpoint, 45 Degree Rotation About Z

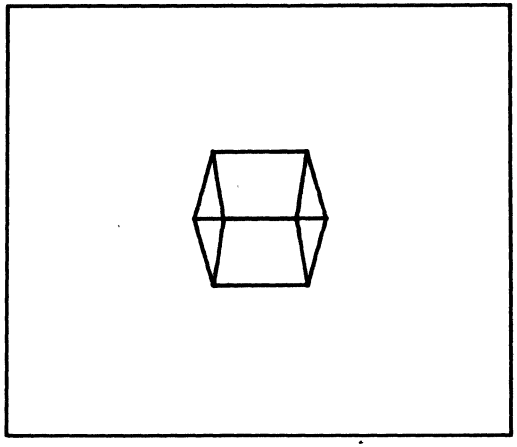


Figure 7. Same Cube, Viewpoint, 45 Degree Rotation About X

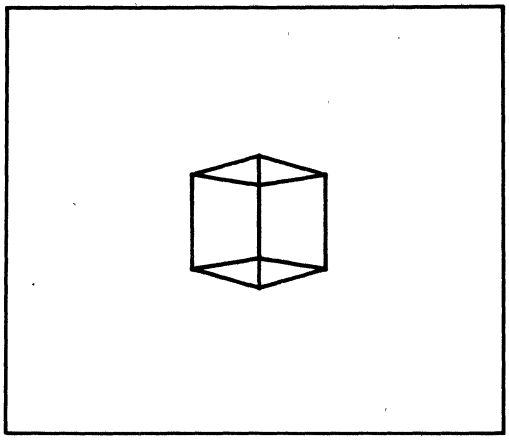


Figure 8. Same Cube, Viewpoint, 45 degree Rotation About Y

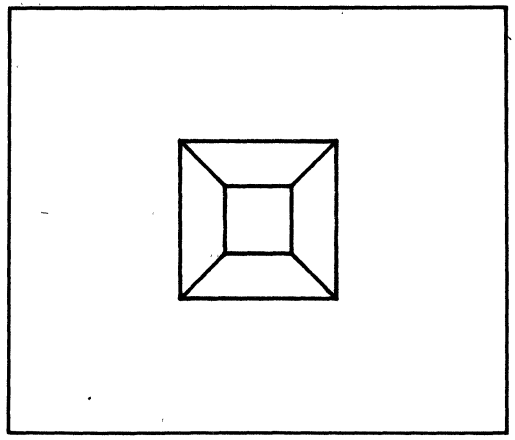


Figure 9. Same Cube, Viewpont 2 x Scale of Z

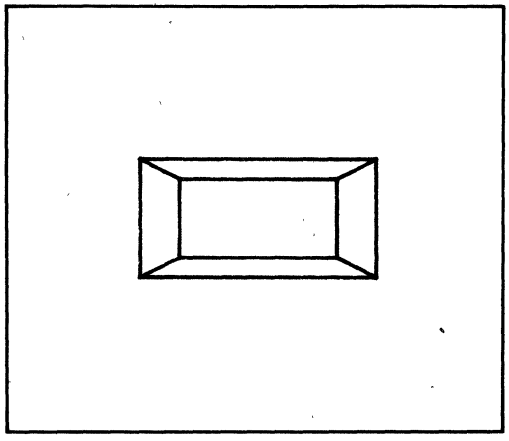


Figure 10. Same Cube, Viewpoint, 2 x Scale of X

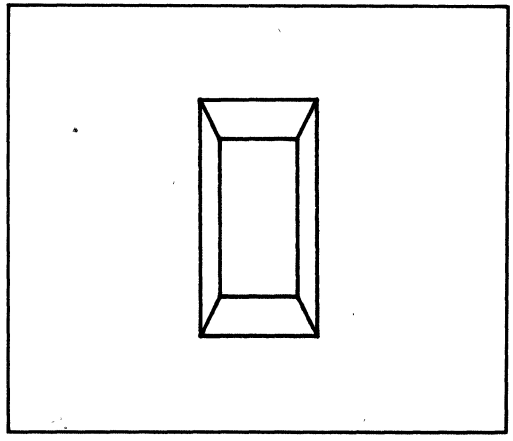


Figure 11. Same Cube, Viewpoint, 2 x Scale of Y

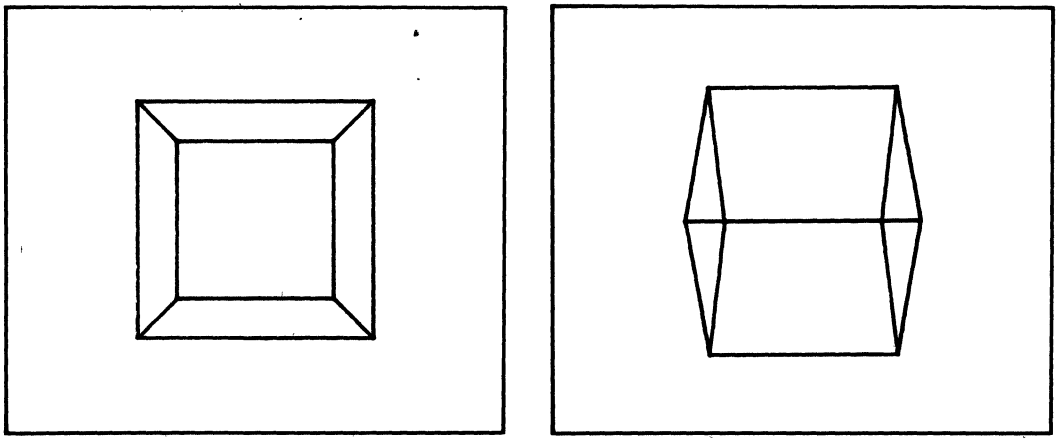


Figure 12. 2 x 2 x 2 Cube Centered at (0,0,0) Viewed from (0,0,10) Then from (10,10,0)

- **Perform the viewing transformation.** This transformation moves and rotates the three dimensional figure according to the viewer's location and orientation (the direction the viewer is facing) in space. An example of changing the view location is shown in Figure 12. Again, this location, or viewpoint, is the viewer's location with relation to an arbitrarily chosen origin.
- **Perform Z-clipping on the three-dimensional data.** This insures that only data in front of the viewer are displayed. In addition, it allows that objects beyond a certain distance from the viewer will not be displayed.
- **Project the three-dimensional data onto a two dimensional surface.** The objects must be projected onto a two-dimensional surface according to the laws of perspective. By changing the "vanishing point," interesting effects are also possible. An example of this is shown in Figure 13. Here, the first figure shows exaggerated perspective (that is, the difference in perceived size between the front face and the back face of the cube is exaggerated), where the second figure shows the object with subdued perspective (the difference in the perceived sizes of the front and back faces is much less than in the first figure). Exaggerated perspective is generated for objects close to the viewer, while subdued perspective is generated for objects distant from the viewer. Note that the same figure, with the same dimensions, is shown in both figures; only the perspective values have been changed.
- **Perform X-Y clipping on the projected data.** This cuts off lines in the projected data extending beyond the specified "window."
- **Perform the window to viewport transformation.** This takes the two-dimensional projected values and scales them according to the relative sizes of the "window" and the "viewport."

The "window" describes the size of the viewer's portal into the data, whereas the "viewport" describes the size and position of this portal on the graphics output device. Whereas the window's size is determined by the size of the input data, the viewport size is determined by the physical characteristics of the graphics display device. For example, the viewport coordinates of a certain CRT display may be constrained to be between 0 and 1023 in both the X and Y dimensions, whereas the window limits are determined only by the maximum size of numbers the computer can store. Thus, for maximum generality and utility, floating point numbers must be used to represent the three-dimensional figures.

A good reference to the techniques used in this three-dimensional graphics implementation can be found in Newman and Sproull [1].

[1] Newman, William M. and Robert F. Sproull, *Principles of Interactive Computer Graphics,* McGraw-Hill Book Company, New York, 1979.

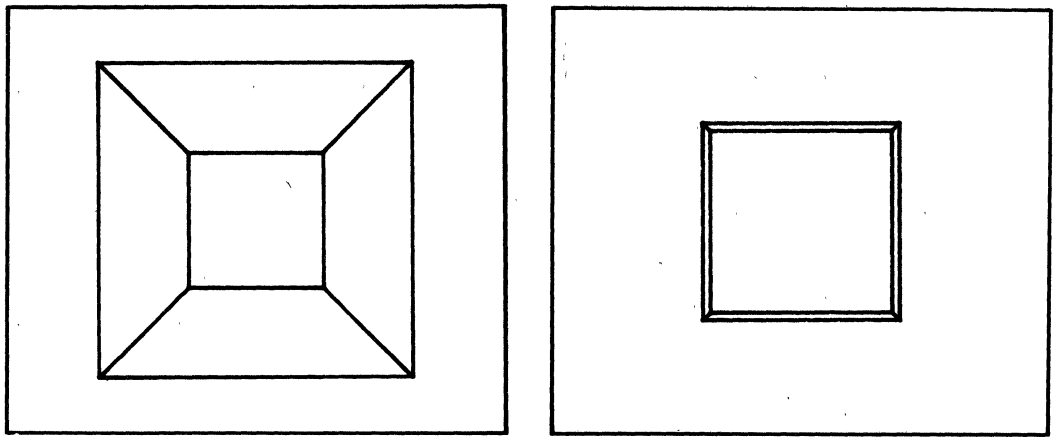


Figure 13. Example Cube Shown with Exaggerated Perspective, then with Subdued Perspective

IMPLEMENTATION

Three-dimensional graphics systems can be split into three functional modules: the input hardware, the processing hardware, and the output hardware. The graphics software is executed by the processing hardware and is used to receive figure definitions from the input hardware, store them in one form or another, and manipulate them such that they can be displayed on the output hardware.

Input hardware can range from the common typewriter keyboard to sophisticated three-dimensional input devices. Output hardware can range from a plotter to a storage tube terminal to a bit-mapped raster scan display or a vector drawing CRT.

The processing hardware can range from general purpose minicomputers to very fast, specialized graphics processing hardware. General purpose computers are used because they allow applications programs to be written in higher level languages. Specialized hardware is sometimes employed when very fast manipulations are required, such as in the real time graphics applications found in flight simulators. This specialized hardware can be used to perform whole matrix transformations. Many applications do not require figures to be drawn real time (on the order of one complete picture every 1/30 sec), however, and can be satisfied by the performance of the general purpose computer alone. A typical application which is satisfied by these latter requirements is a Computer Aided Design (CAD) system. However, since these graphics systems often exist in an interactive environment, picture processing delays greater than a few seconds for simple figures, or greater than a few minutes for very complex figures cannot be tolerated. Because of these processing requirements, a mini-computer with a hardware floating point unit has been required to drive these graphics systems. However, with the introduction of the 8087, the floating point processing performance required by these systems can finally be met in a microcomputer solution.

The microcomputer system used in this three-dimensional graphics application is a general purpose microcomputer embodied in the iAPX 86/12 board found in an Intel Intellec® Series III development system. All routines implemented in this application note were written entirely in FORTRAN using the Intel FORTRAN 86 compiler. Any iAPX 86/20 (or iAPX 88/20) with enough memory can be used to execute the programs, however. The amount of memory required depends on the number and complexity of the figures to be displayed. The source code for all routines used in this note are given in the appendix.

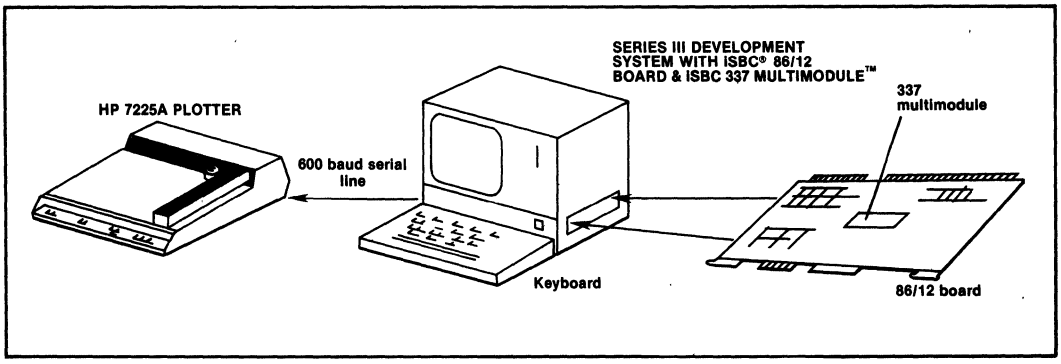


Figure 14. Computer System Used in This Graphics Implementation

The graphics output device used was a HP 7225A flat bed plotter. Communications were performed using the RS232 serial link on the 86/12 board. The communications speed of the line to the plotter was 600 baud. Because of the number of lines drawn in the more complex figures, the physical characteristics of the plotter, and the communications line speed, the amount of time required to draw a large picture was a function of the plotter speed, not the execution speed of the iAPX 86/20. As a result, all times quoted in this note do not reflect the plotting time. Only the time up to placing the ASCII character into the buffer of a serial communications chip is included for all machines quoted. Higher speed graphics display devices (which are not limited by the physical characteristics of plotters) can use the speed of the iAPX 86/20 to full advantage.

The graphics input device used was the standard alphanumeric keyboard attached to the development system. This allows entry of figures, as well as control of the graphics system. Input can also be fetched from disk storage, however, to allow for greater speed in defining large figures. A block diagram of the hardware system used in this implementation is shown in Figure 14.

All routines were run using both the 8087 and the 8087 software emulator. The 8087 software emulator is a software package exactly emulating the internal operation of the 8087 using 8086 instructions. When the emulator is used, an 8087 is not required. The emulator is a software product available from Intel as part of the 8087 support library. The performance of the 8087 hardware is much better than that of the software emulator, as one would expect from a specialized hardware floating point unit.

The 8087 supports various data formats. For real numbers, these formats are short real (or single precision), long real (or double precision), and temporary real (or extended precision). The differences among the three are in the number of bits allocated to represent a given floating point number.

In all real numbers, the data is split into three fields: the sign bit, the exponent field and the mantissa field. The sign bit shows whether the number is positive or negative. The exponent and mantissa together provide the value of the number: the exponent providing the power of two of the number, and the mantissa providing the "normalized" value of the number.

A "normalized" number is one that always lies within a certain range. By dividing a number by a certain power of two, most numbers can be made to lie between the numbers 1 and 2. The power of two by which the number must be divided to fit within this range is the exponent of the number, and the result of this division is the mantissa. This type of operation will not work on all numbers (for example, no matter what one divides zero by, the result is always zero), so the number system must allow for these certain "special cases."

As the size of the exponent grows, the range of numbers representable also grows, that is, larger and smaller numbers may be represented. As the size of the mantissa grows, the resolution of the points within this range grows. This means the distance between any two adjacent numbers decreases, or, to put it another way, finer detail may be represented. Short real numbers provide 8 exponent bits and 23 significand or mantissa bits. Long real numbers provide 11 exponent bits and 52 significand bits. Temporary real numbers provide 15 exponent bits and 64 significand bits. These data formats are shown in Figure 15. Thus, of the three data formats implemented, short real provides the least amount of precision, while temporary real provides the greatest amount of precision. These levels of precision represent only the external mode of storage for the numbers; inside the 8087 all numbers are represented to temporary real precision. Numbers are automatically converted into the temporary real precision when they are loaded in-

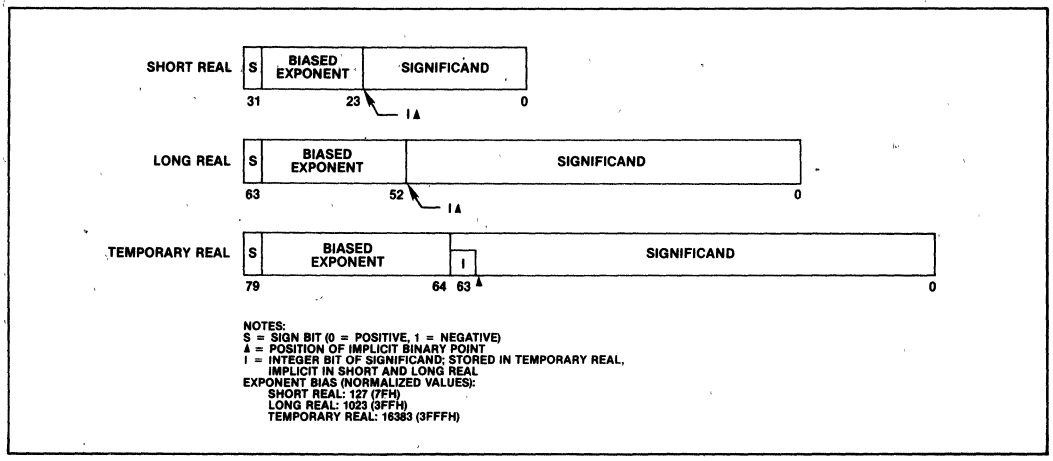


Figure 15. Floating Point Data Formats

to the 8087. In addition to real format numbers, the 8087 automatically converts to and from external variables stored as 16, 32 or 64-bit integers, or 80-bit binary coded decimal (BCD) numbers.

Memory requirements also increase as precision increases. Whereas a short real number requires only four bytes of storage (32 bits), a long real number requires eight bytes (64 bits) and a temporary real number ten bytes (80 bits). In many floating point processors, processing time also increases dramatically as precision is increased, making this another consideration in the choice of precision to be used by a routine. The differences in 8087 processing time among short real, long real and temporary real numbers are insignificant compared to the processing time, however, since all operations are performed to the internal 80-bit precision. This makes the choice of which precision to use in an iAPX 86/20 system a function only of memory limitations and precision requirements.

Double precision numbers were chosen for this graphics implementation because they allow a very wide range of numbers to be represented with high precision. This is important, since the package allows the user to magnify small parts of defined figures. Without the precision gained by using double precision numbers, the image of the object could easily be distorted under such scrutiny.

Three-Dimensional Figure Description and User Interface

The graphics user interface implemented in this note is both functional and simple. It does not require the use of specialized three-dimensional input hardware. All input data is keyed in through the keyboard.

The package allows for definition of figures for future use within the graphics package. This feature could be useful in generating multiple views of a certain object. It requires that the object be "defined" at the beginning of the session, but then allows the user to view the object from any location, with any rotation, scale, or translation.

Commands to the graphics package consist of a set of alphanumeric commands followed by the necessary numeric constants. To enter commands to the graphics package, one enters an alphanumeric command enclosed within the single quotes followed by the appropriate numeric arguments. The maximum number of arguments required by any command is six. If less than six arguments are entered on a line, the line must be terminated by the '/' character, however. These requirements (having the command enclosed within single quotes, explicitly terminating the line) are a result of using the list-directed input format of FORTRAN.

The commands recognized by the graphics processor are:

comment *arg1*. This command instructs the graphics processor to ignore the next *arg1* lines. This can be used to insert comments within the graphics commands.

define *arg1*. This command instructs the graphics processor that the next N lines (up to the **enddef** command) are to be entered into an internal buffer for future reference as figure *arg1*. The graphics commands are not interpreted, i.e. they do not cause figures to be drawn as they are entered. In this way, three-dimensional objects may be defined, or to put it another way, placed into an internal display list. Up to ten objects may be defined using the current version of the program. This may be increased to the limits of available memory. Currently there is internal storage space for up to 500 total graphics commands. These may be spread in any combination among the ten figures. This number may also be modified to reflect memory restrictions.

enddef. This command terminates a figure definition, and returns control back to the main graphics processor.

call *arg1*. This command causes the graphics processor to fetch graphics commands from the internal buffer of the previously defined figure number *arg1*.

line *arg1 arg2 arg3 arg4 arg5 arg6*. This command causes a line to be drawn in three-dimensional space from the point *arg1, arg2, arg3* to the point *arg4, arg5, arg6*. The current object rotation, object scale, object translation, viewer location, window, and viewport are used.

plot *arg1 arg2 arg3 arg4*. This command causes a line to be drawn from the endpoint of the last line plotted to the point *arg1, arg2, arg3* using the "pencode" *arg4*. The current pencodes supported are '2' (indicating that a solid line is to be drawn), and '3' (indicating that no line is to be drawn; this is used only to change the location of the plot head). Additional pencodes could be implemented allowing for dashed lines, dotted lines, etc.

ident. This command causes the "current" matrix to be set to the identify matrix. This causes all rotates to be set to zero, all translates to be set to the origin, and all scales to be set to one.

push. This command causes the current matrix to be pushed onto a 10 location matrix stack. The current matrix is not altered.

pop. This command causes the matrix stack to be popped into the current matrix.

rotate *arg1 arg2 arg3*. This command causes the viewer's perception of the three-dimensional figure to be rotated around the X, Y, and Z axis by *arg1, arg2* and *arg3*. The angles are in degrees. The definition of an object is not altered.

translate *arg1 arg2 arg3*. This command causes the viewer's perception of the three-dimensional figure to be translated in the X, Y, and Z directions by *arg1, arg2* and *arg3*. Again, the definition of an object is not altered.

scale *arg1 arg2 arg3*. This command causes the viewer's perception of the three-dimensional figure to be scaled in the X, Y and Z directions by *arg1, arg2,* and *arg3*.

window *arg1 arg2*. This command sets up the window parameters. These parameters determine the visible side to side portion of the projected images. This amounts to placing an infinitely tall pyramid within three-dimensional space with the viewing location located at its apex (looking down). All objects within this pyramid will be visible; all objects outside this pyramid will not be visible.

viewport *arg1 arg2 arg3 arg4*. This command sets up the viewport parameters. These parameters determine the size and location of the above window on the plotter surface. The center of the area on the plotter surface is given by *arg1, arg2* with the X and Y half sizes given by *arg3, arg4*. The plotter is assumed to have an X dimension between 0 and 12, and a Y dimension between 0 and 10. The translation to the dimensions the plotter recognizes is done in a lower level plotter interface routine. By performing this task in a lower level of software, the package is made more general.

viewpoint *arg1 arg2 arg3 arg4 arg5 arg6*. This command sets up the "viewing" transformation. *arg1, arg2, arg3* represent the location of the viewer in three-dimensional space, while *arg4, arg5, arg6* represent the "lookat" location in three-dimensional space. Together they form a vector pointing to the area to be viewed whose length determines the perspective variables (only single point perspective is currently implemented).

zclip *arg1 arg2.* This command sets up the "Z-clipping" parameters. These determine the visible distance in front of the viewer. *Arg1* specifies the near boundary of the viewing area while *arg2* specifies the far boundary of the area. Together with the window command, it defines a solid delimiting the visible objects from the not-visible objects.

cube *arg1 arg2 arg3 arg4 arg5 arg6.* This command draws a cube centered at *arg1, arg2, arg3* with half-widths of *arg4, arg5* and *arg6.*

arrow. This command draws an arrow from (0,0,0) to (1,0,0).

pyramid *arg1 arg2 arg3 arg4 arg5 arg6.* This command draws a four-sided pyramid whose base is centered at *arg1, arg2, arg3* and whose half-widths are *arg4, arg5, arg6.* The X half-width *arg4* is used as the height of the pyramid.

current. This command prints the current matrix on the terminal.

printdef. This command prints the definition of the given figure.

startt. This command starts the 10 ms timer on the iSBC 86/12 board.

readt. This command stops the 10 ms timer on the iSBC 86/12 board and prints the 10 ms count on the terminal.

end. This command stops execution of the graphics package, prints the total numbers of points plotted and "success!!!" on the terminal, and returns control back to ISIS.

Internal Operation of the Package

All internal operations are performed using 1 by 4 or 4 by 4 double precision real matrices. Points are defined in 1 by 4 double precision vectors where the first three coordinates are used to hold the X, Y and Z location of the point. The fourth location is always set to one, and is used when the point is projected onto a two-dimensional plane. In most cases, the routine performing the task outlined is named the same thing as the name of the task outlined (within the six-character limit imposed by FORTRAN). The order the routines are described is roughly the order a line would encounter them on its way from existing as a three-dimensional entity inside the machine to a line drawn on the bed of a plotter. All routine names are set in **boldface.**

THE CURRENT TRANSFORMATION

If each object were to be modified whenever a translate, rotate, or scale were to be performed, performance of the package could be quite slow. In addition, the original definition of the figure would be lost (although not irreversibly). If there were a method of performing these three operations at a single time, allowing the original definition of an object to remain unaltered, both the performance and ease of use of the graphics package would be enhanced.

One way in which these operations can be combined is by using what is called the "current" matrix. The current matrix is a 4 by 4 double precision real matrix. It numerically represents any combination of rotates, translates and scales in any order. The matrix is multiplied by each 1 by 4 point definition vector on its way to being plotted. The result of this multiplication is a point that has been rotated, scaled, and translated the proper amount. If this matrix is the identity matrix, the point will pass through unaltered. Thus, the identity matrix represents no scaling, translating, and rotating. This multiplication is performed in the routine **pline** lines 20 and 21.

When a rotate, scale, or translate command is interpreted, the current matrix is multiplied by another 4 by 4 matrix representing only this transformation. Since matrix multiplication is not commutative, the order these operations are performed in is preserved. This is important, because, for example, a rotate before a translate is not the same as a rotate after a translate because all rotations are performed pivoting around the origin (see Figure 16). Initially, the current matrix is set to the identity matrix. The first operation is performed relative to state of the current matrix immediately preceding the operation.

Parameters are set up into the current matrix through the rotate, scale, translate, ident, push, and pop operations. Each name describes the function of the operation performed. The routines performing these tasks (in order) are: **rotate, scale, transl, ident, push,** and **pop. Ident** is included to allow all rotates and translates to be set to zero and all scales to be set to one. The **push** and **pop** operations are included in order that figures may save the state of the current matrix, while subsequently performing operations altering it. This is important when a large figure is defined as a set of parts, each of which may merely be rotations, etc., of a simpler list of parts.

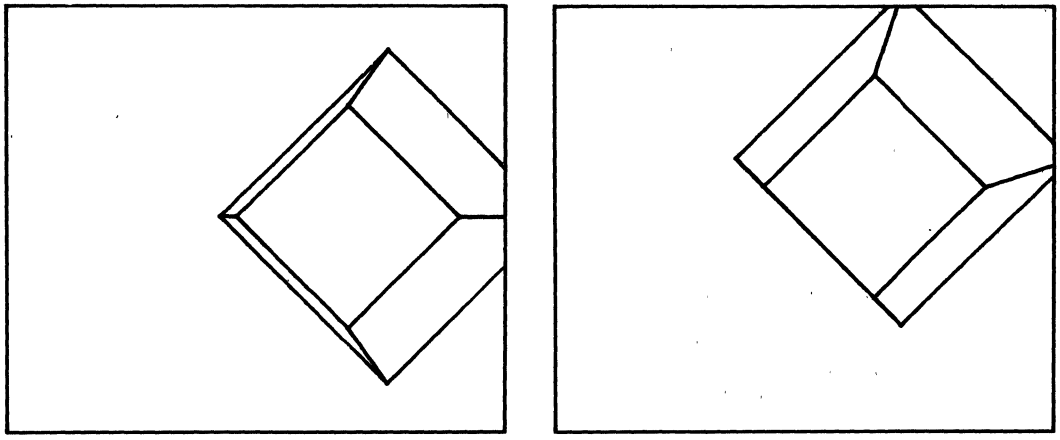


Figure 16. Example Cube Viewed from (0,0,10) First Rotated then Translated then Translated then Rotated.

THE VIEWING TRANSFORMATION

Before an object can be plotted, the viewpoint of the viewer must be known. This information provides the location of the viewer in three-dimensional space, and the direction the viewer is pointing. It is incorporated into the 4 by 4 "view" matrix. It is another rotation performed on the object in order that it is viewed from the proper viewing angle. All points are passed through the view matrix after they are passed through the current matrix. What comes out of these two transformations is a set of points located in the proper relative positions in three-dimensional space when the figure is rotated, translated, and scaled by the operations performed on the current matrix, and is also rotated properly by the operations set in the view matrix.

The view matrix is set up by the viewpoint command. This command will place in the view matrix the proper rotations in order that the image of the object will be correct. The routine performing this task is the **viewpn** routine.

Z-CLIPPING

All points passed through the current and view matrices are located at their proper locations in three-dimensional space. However, only a portion of this space is visible to the viewer. Specifically, objects behind the viewer will not be visible. Every point of an object has been mapped to the viewer's space, however, including those behind the viewer. These "invisible" points are removed by an operation called "Z-clipping." Simply, it examines the Z parameter of every point being considered and determines if it is in front of the viewer. In addition, one may not wish to display lines a great distance from the viewer. These lines may be removed by a similar process. The only complication of clipping is the action performed if only part of the line is visible. In this instance, the point where the line leaves the visible area must be calculated. The method used to calculate this point in this implementation is the method of "like triangles."

The Z-clipping parameters are set through the command zclip in the routine **zclip.** The arguments to this command are used to determine the visible distance in front of the viewer. The first argument sets the minimum distance in front of the viewer before any line will be visible. Legal values for this parameter are anything greater than zero. The second argument sets the far distance beyond which no lines will be visible. Any value larger than the first argument may be used for this parameter. The clipping itself is performed in the routine **zclipp.**

PROJECTION

Projection maps the three-dimensional points previously encountered and projects them onto a two-dimensional plane. Only single-point perspective is currently supported in the package. Here, the projection is performed by using the Z parameter to modify the X and Y parameters. As the points get more distant, their deviation from the center of the picture should get smaller, if the X and Y parameters remain constant. Most people are aware of this effect. For example, if you look down a set of railroad tracks, the rails seem to converge, even though the distance between the rails is constant (see Figure 17). Two or three-point perspective would be easy to implement; all one must do is generate the projected X and Y parameters by using the non-projected X and Y parameters in addition to using the Z parameter.

This projection is performed in the graphics package by multiplying the 1 by 4 point location vector by a 4 by 4 "projection" matrix. This matrix is simply the identity matrix except the perspective value is placed in location (3,4) of the matrix.

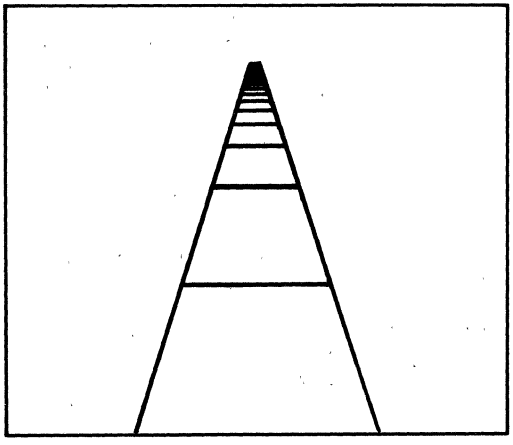


Figure 17. Two Rails, Vanishing into the Distance

This value is calculated from the viewpoint parameters. After the matrix multiply, the only element modified in the 1 by 4 point definition vector is the last one (the one which is supposed to have the value of one). After the multiplication, this location will contain the number representing the modification which must be performed on the X and Y parameters of the vector to exhibit the projection. When this vector is "normalized," the point will have been projected using the rules of single-point perspective. This normalization is performed by dividing every element in the vector by the last element of the vector. Thus, the Z element of the original vector has modified the X and Y elements. If two or three-point perspective is desired, one must only place perspective values in locations (1,4) and (2,4) of the projection matrix; all subsequent processing will be identical. The routines performing these operations are: **viewpn** (sets up vanishing point for perspective), **projct** (sets up the projection matrix, and performs the perspective multiplication), and **norm** (normalizes the vector).

X-Y CLIPPING

Once the data is projected onto a two-dimensional plane, X-Y clipping must be performed. This operation could also be performed on the three-dimensional data, but by deferring it until after the data have been projected, the calculations required are simpler. This is not true for Z-clipping, since once the data are projected onto a plane, the Z parameter is no longer in its original form.

X-Y clipping is performed by comparing X and Y parameters with the window values set up by the window command. This comparison is a bit more complicated than the comparison required by Z clipping, however, as two clipping parameters are involved. There are nine possible regions in which each endpoint of a line may reside. For example, some of these regions are: within the X and Y window regions, less than the X window region but within the Y region, less than the X region and less than the Y region, etc. If one or both of the endpoints of the line are within the visible region, then at least part of the line will be visible. Also, even if neither of the endpoints of the line is in the visible region, part of the line may still be visible. One must therefore determine whether any part of this line would be visible. A simple way of performing the task is to assign a bit of a word for each of less than and greater than the X and Y window limits. This requires four bits. The value of the X and Y parameters are then each compared with the window limits. If the value exceeds the limit of the window, the corresponding bit of this point descriptor is set. After this "code" has been determined for both of the points, the codes for two endpoints are bit-wise ANDed together (an extension to FORTRAN '77 available in FORTRAN 86 allows this operation). If the result of this ANDing is zero, then part of the line would be visible. If, however, it is not zero, then the entire line lies outside the visible area. If only part of the line is visible, then the point where it leaves the visible area must be calculated. The point where the line leaves the viewing area is calculated using the same "like triangle" method used when Z-clipping is performed.

The routines performing these operations are **wtovp** (calls the **xyclip** routine with the proper parameters), **xyclip** (performs the actual clipping), **code** (returns the binary code for the point position in relation to the window), and **ppush** (calculates the point at which line leaves the visible area).

WINDOW TO VIEWPORT TRANSFORMATION

Finally, after the points have been processed through all of the above, comes their day of glory. Because the lines have been clipped, they are constrained to be within the given window. Remember, however, that the values for this window are in "real world" units. These sizes could be measured in inches or miles. These are not generally suitable for plotting on a graphics output device. In order for the "window" to be displayed on the graphics output device, one more transformation must be performed: the window to viewport transformation. A viewport represents a physical location and size on the graphics output device. The viewport command sets up the appropriate parameters for this transformation. It requires four arguments, which allow the viewport to be moved around the graphics display surface, and allow the size of the viewport to be set. Notice that the viewport and the window are not constrained to the same aspect ratios, that is, the ratios between the vertical sizes and the horizontal sizes of the window and viewport need not be the same. If these ratios are not the same, the figures will be distorted. Performing this transformation is simply a matter of scaling the windowed values to fill the viewport. The code performing this transformation is contained within the **wtovp** routine.

PLOTTER INTERFACE

This graphics package was written to interface to a Hewlett-Packard 7225A flat bed plotter. Communications were performed through an RS232 serial link at 600 baud. Physically, this is done using the 8251 serial controller on the iSBC® 86/12 board inside the Intellec® Series III. The plotter has a smart interface. The commands it accepts are in ASCII, and are on the level of "lower the pen," and "draw a line from the current pen position to another pen position." The routines performing these operations are **plot** (determines the characters needing to be sent to the plotter), **ponum** (converts a floating point number to an ASCII representation of the integer value of the truncated floating point number), **putout** (handles the interface to the 8251 serial controller chip) and **plots** (initializes the baud rate generator and 8251 serial controller chip on the iSBC® 86/12 board).

PERFORMANCE MEASUREMENTS

The above routines were compiled using the Intel FORTRAN 86 compiler and exeucted on an Intellec® Series III development system. The 8086 hardware consists of an Intel iSBC® 86/12 board with the 8087 in the iSBC® 337 card. The iAPX 86/20 (the 8086 with the 8087) operate with a clock frequency of 5 MHz. The on board memory (64K DRAM) inserts between one and three wait states per memory fetch. In addition, owing to the size of the memory arrays, the program size, and the memory reguirements of the Series III, off board memory was required to run the program.

The times shown in the table do not show the plotting time; only the time to generate the output that would be sent to the plotter is given. This is because the physical speed limitation of the plotter used would not allow the iAPX 86/20 system to produce the plotting commands at its maximum computational speed. The plotter required approximately half an hour to 45 minutes to actually draw the second demonstration picture.

For each line plotted, five 1 by 4 times 4 by 4 matrix multiplies must be performed along with a non-trivial amount of other floating point operations, such as divides and compares. For example, when clipping is performed, the line endpoint values must be compared to the clipping parameters. If only part of the line is visible, then the point the line leaves the visible area must be calculated. This requires twelve additional floating point operations. Another example is in the window to viewport transformation. For each line drawn, four floating point multiplies, four floating point divides, and four floating point adds must be performed.

In addition, whenever the rotation, scale, translation or viewpoint is changed, 4 by 4 matrix multiplies must be performed. In addition, various trigonometric routines, such as sines and cosines, must be performed to set up the rotation parameters into the matrix.

The performance measurements are given in Table 1.

Table 1. Performance Measurements

	Picture Number	
	One	Two
number of points in picture	117	9131
number of points actually plotted	117	6114
execution time of the 86/20(sec)	2.84	188
execution time of the 86 with 87 emulator(sec)	144.77	9801
exection time of PDP11/45(sec)[2]	1.7	120

[2]A PDP11/45 mini-computer with 256K MOS RAM, and a FP11-B floating point unit running the UNIX operating system during a period of light load. The program was compiled using the UNIX F77 FORTRAN compiler.

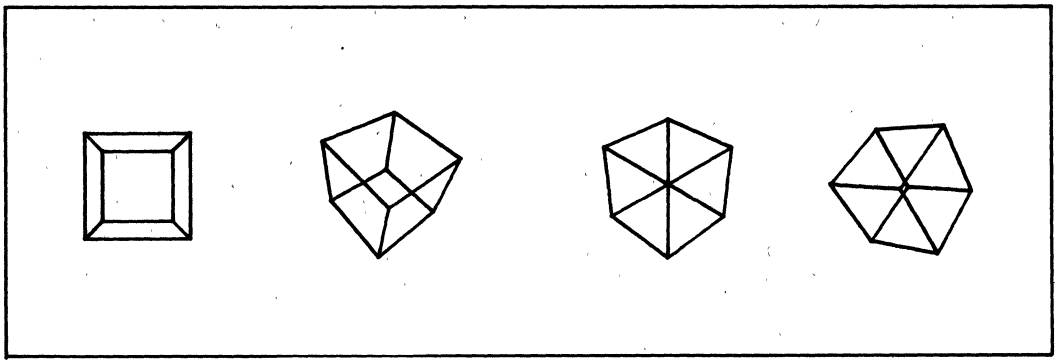


Figure 18. Demonstration Picture 1

The results show that the performance of the iAPX 86/20 is close to the performance of the mini-computer. The figures drawn are shown in Figure 17 for Picture 1 and Figure 18 for Picture 2. The graphics commands required to generate Picture 1 are given in Appendix B. Picture 2 shows three views of a single shuttle. (Hint: you are looking out the window of one of the shuttles!) The shuttle is defined only once in the input data. Another point to notice is that each shuttle is a conglomeration of parts. For example, the shuttle wing is defined only once in input data. The complete shuttle contains two views of this same wing, translated and rotated to attach to the appropriate location on the fuselage of the shuttle itself. The engine nozzles take this same approach a bit further. The complete nozzle is defined only once, and is attached in three places on each shuttle. In addition, each nozzle is made up of replications of the same circle scaled and translated through space. Each circle is, in turn, composed of four views of one quarter-circle, each rotated a proper amount to form one complete circle.

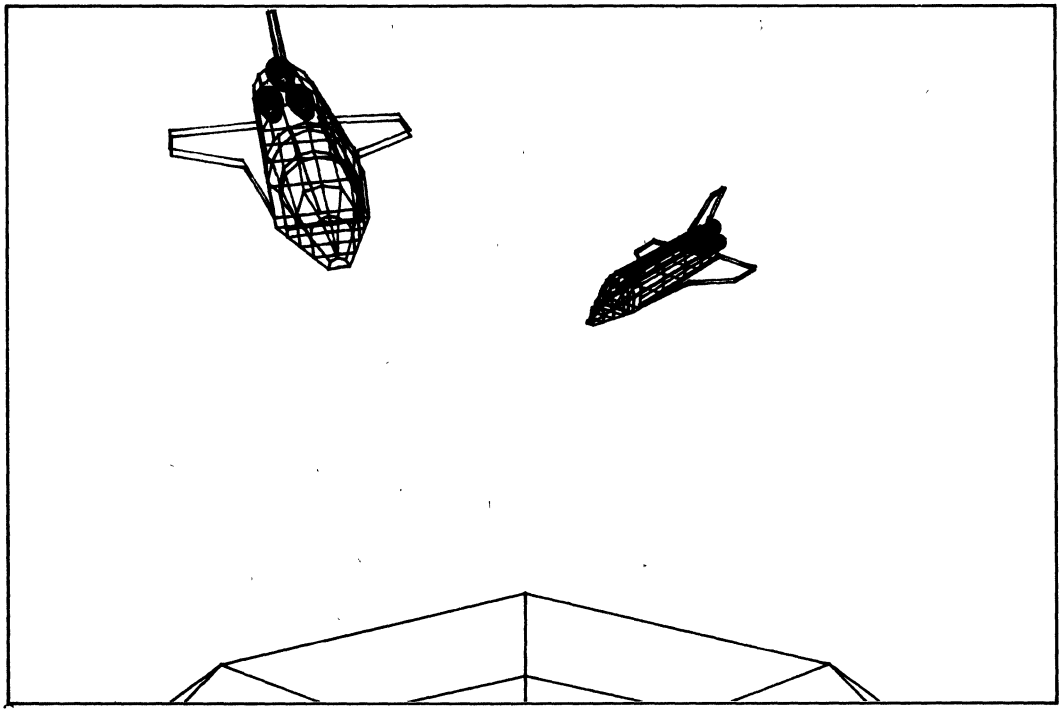


Figure 19. Demonstration Picture 2

CONCLUSIONS

The routines demonstrated in this note show that the types of operations required to manipulate and display a three-dimensional figure on a two-dimensional surface are far from trivial, involving very many floating point operations. With the introduction of the iAPX 86/20, the floating point performance required by this type of application is finally within the performance limits of microcomputers selling for a fraction of the cost of the previously required mini- or maxi-computers. Examples of systems in which this performance is required are Computer Aided Design (CAD) or Computer Aided Manufacturing (CAM) systems. In addition, the availability of a full ANSI 77 standard FORTRAN compiler (FORTRAN 86) for the iAPX 86/20 enhances the production or transportation of existing software to the machine. This combination of high performance hardware with high performance software allows the iAPX 86/20 to fill applications never before filled by a microprocessor.

APPENDIX A Contents

PAGE

```
      c
      c this is the main routine of the graphics program.  basically
      c  it sets up default parameters for the rest of the routines, then
      c  enters an infinite loop, alternatively fetching lines from the input
      c  (using routine getl) and sending them to be processed by the graphics
      c  processor (proc)
      c
 1            common /windoe/wxh,wyh
 2            common /viewp/vxh,vyh,vxc,vyc
 3            real*8 wxh,wyh,vxh,vyh,vxc,vyc
 4            common /matrix/currm,view,curp
 5            real*8 currm(4,4),view(4,4),curp(4)
 6            common /clip/hither,yon,dee
 7            real*8 hither,yon,dee
 8            common /stacks/stackp,sspace
 9            real*8 sspace(10,4,4)
10            integer stackp
11            common /defns/darg1,darg2,darg3,darg4,darg5,darg6,darg7,entry,tailp,ends
12            character*10 darg1(500)
13            real*8 darg2(500),darg3(500),darg4(500),darg5(500),darg6(500),darg7(500)
14            integer entry(10),ends(10)
15            integer tailp
16            common /cstack/cnum,cnump
17            integer cnum(10),cnump
18            common /penpos/xpos,ypos,pcount
19            real*8 xpos,ypos
20            integer*4 pcount

      c  initialize the plotting package
21            call plots
      c       initialize the stack pointer
22            stackp = 1

      c  set up a few defaults
23            wxh = 10.
24            wyh = 10.
25            vxh = 5.
26            vyh = 5.
27            vxc = 5.
28            vyc = 5.
29            hither = 1.
30            yon = 100.
31            dee = 10.
32            tailp = 1
33            cnump = 1
34            xpos = -1.
35            ypos = -1.
36            pcount = 0
37            print *,'GRAPHICS program entered!!!'
      c
      c  initialize the current matrix
      c
38            call ident(currm)
      c
      c and process all the input lines
      c
39    100             call getl
40                    call proc
41            goto 100
42            end
```

```
      c
      c       getl(line)
      c
      c       fetches the next line from the input file, and grabs the first 7
      c       things from it, the first being an alpha command contained within
      c       (') and the rest being numbers.  If less than 6 number are input
      c       the input line must be terminated by a (/) in order for the
      c       read statement to be correctly interpreted.  The arguments are then
      c              placed in the common block "args".  When the 'end' command is
      c              encountered, "success" is printed on the terminal, and the
      c              graphics program terminates.
      c
43            subroutine getl
44            common /args/arg1,arg2,arg3,arg4,arg5,arg6,arg7
45            character*10 arg1
46            real*8 arg2,arg3,arg4,arg5,arg6,arg7

47            read (5,*)arg1,arg2,arg3,arg4,arg5,arg6,arg7
48            if(arg1 .eq. 'end') then
49                   call plot(0.,0.,999)
50                   print *,'success!!!'
51                   stop
52            endif
53            return
54            end


      c
      c       proc
      c
      c       proc() does all the processing for a line.  It gets its arguments
      c       from the common block args, and does it's thing
      c
55            subroutine proc

56            common /matrix/currm,view,curp
57            real*8 currm(4,4),view(4,4),curp(4)
58            common /args/arg1,arg2,arg3,arg4,arg5,arg6,arg7
59            character*10 arg1
60            real*8 arg2,arg3,arg4,arg5,arg6,arg7
61            common /clip/hither,yon,dee
62            real*8 hither,yon,dee
63            common /cstack/cnum,cnump
64            integer cnum(10),cnump
65            integer i
66            integer*4 rtimer,countt

      c
      c              determine the command entered. (HUGE if-then-else if-,etc) and
      c              call the appropriate routine with the correct arguments
      c
67            if(arg1 .eq. 'comment') then
68                   i = 1
69    100            read(5,800)
70                   i = i + 1
71                   if(i .le. int(arg2)) goto 100
72    800            format(a1)
73            else if(arg1 .eq. 'define') then
74                   i = int(arg2)
75                   call defn(i)
76                   call printd(i)
77            else if(arg1 .eq. 'call') then
78                   cnum(cnump) = int(arg2)
79                   cnump = cnump + 1
80                   if(cnump .gt. 10) then
81                          print *,'call nesting level too deep, sorry'
82                          cnump = 10
83                   endif
84                   call callit(cnum(cnump - 1),cnump - 1)
85                   cnump = cnump - 1
86            else if(arg1 .eq. 'line') then
```

```
 87                 call pline(arg2,arg3,arg4,arg5,arg6,arg7,2)
 88           else if(argl .eq. 'plot') then
 89                 i = int(arg5)
 90                 call pplot(arg2,arg3,arg4,i)
 91           else if(argl .eq. 'ident') then
 92                 call ident(currm)
 93           else if(argl .eq. 'push') then
 94                 call push(currm)
 95           else if(argl .eq. 'pop') then
 96                 call pop(currm)
 97           else if(argl .eq. 'rotate') then
 98                 call rotate(arg2,arg3,arg4,currm)
 99           else if(argl .eq. 'translate') then
100                 call transl(arg2,arg3,arg4,currm)
101           else if(argl .eq. 'scale') then
102                 call pscale(arg2,arg3,arg4,currm)
103           else if(argl .eq. 'window') then
104                 call window(arg2,arg3)
105           else if(argl .eq. 'viewport') then
106                 call viewpr(arg2,arg3,arg4,arg5)
107           else if(argl .eq. 'viewpoint') then
108                 call viewpn(arg2,arg3,arg4,arg5,arg6,arg7)
109           else if(argl .eq. 'zclip') then
110                 call zclip(arg2,arg3)
111           else if(argl .eq. 'cube') then
112                 call cube(arg2,arg3,arg4,arg5,arg6,arg7)
113           else if(argl .eq. 'arrow') then
114                 call arrow
115           else if(argl .eq. 'pyramid') then
116                 call pyrmd(arg2,arg3,arg4,arg5,arg6,arg7)
117           else if(argl .eq. 'current') then
118                 call printm(currm)
119           else if(argl .eq. 'printdef') then
120                 i = int(arg2)
121                 call printd(i)
122           else if(argl .eq. 'startt') then
123                 call stimer
124           else if(argl .eq. 'readt') then
125                 countt = rtimer()
126                 print *,'the time (in seconds) from the last startt is:',countt/100.
127           else
128                 print *,'error, command ',argl,'unknown'
129           endif

130           return
131           end

      c
      c     ident(matrx)
      c
      c     ident() sets the given 4 X 4 matrix to the identity matrix.
      c
132           subroutine ident(matrx)
133           real*8 matrx(4,4)
134           integer i,j

135           do 100 i=1,4
136                 do 100 j=1,4
137                       matrx(i,j) = 0.
138     100   continue
139           do 110 i=1,4
140                 matrx(i,i) = 1.
141     110   continue
142           return
143           end
```

```
      c
      c       subroutine defn(number) defines figure number.  the defined figure
      c               is contained in a large common block "defns" which contains
      c               enough space for a total of 500 commands.  comments are not
      c               stored along with the define commands to save space.  the variables
      c               entry and ends contain the starting and ending indexes of the
      c               10 possible defined figures
      c
144           subroutine defn(number)
145           integer number
146           common /defns/darg1,darg2,darg3,darg4,darg5,darg6,darg7,entry,tailp,ends
147           character*10 darg1(500)
148           real*8 darg2(500),darg3(500),darg4(500),darg5(500),darg6(500),darg7(500)
149           integer entry(10),ends(10)
150           integer tailp
151           common /args/arg1,arg2,arg3,arg4,arg5,arg6,arg7
152           character*10 arg1
153           real*8 arg2,arg3,arg4,arg5,arg6,arg7
154           integer i

155           entry(number) = tailp
156           print *,'start of define is at',tailp

157     100   call getl
      c
      c       check for terminate of define
      c
158           if(arg1 .eq. 'enddef') then
159                   ends(number) = tailp
160                   print *,'end of figure define is at',tailp
161                   return
162           else if(arg1 .ne. 'comment') then
163                   darg1(tailp) = arg1
164                   darg2(tailp) = arg2
165                   darg3(tailp) = arg3
166                   darg4(tailp) = arg4
167                   darg5(tailp) = arg5
168                   darg6(tailp) = arg6
169                   darg7(tailp) = arg7
170                   tailp = tailp + 1
171                   if(tailp .gt. 500) then
172                           print *,'define memory overrun!!!'
173                           tailp = 500
174                   endif
175           else
176                   i = 1
177     150           read(5,800)
178                   i = i + 1
179                   if(i .le. int(arg2)) goto 150
180     800           format(a1)
181           endif
182           goto 100
183           end
```

```
      c
      c       subroutine printd(number) prints the defined figure commands
      c
184           subroutine printd(number)
185           integer number
186           common /defns/darg1,darg2,darg3,darg4,darg5,darg6,darg7,entry,tailp,ends
187           character*10 darg1(500)
188           real*8 darg2(500),darg3(500),darg4(500),darg5(500),darg6(500),darg7(500)
189           integer entry(10),ends(10)
190           integer tailp
191           integer i

192           i = entry(number)
193     100   if(i .eq. ends(number)) return
194           write(6,800)darg1(i),darg2(i),darg3(i),darg4(i),darg5(i),darg6(i),darg7(i)
195     800   format(a10,6f11.4)
196           i = i + 1
197           goto 100
198           end


      c
      c       subroutine callit(number,nest)  causes the defined figure number to
      c               be input to the graphics processor, nesting level must be provided
      c               to allow pseudo-recursive type calls...
      c
199           subroutine callit(number,nest)
200           integer number,nest
201           common /defns/darg1,darg2,darg3,darg4,darg5,darg6,darg7,entry,tailp,ends
202           character*10 darg1(500)
203           real*8 darg2(500),darg3(500),darg4(500),darg5(500),darg6(500),darg7(500)
204           integer entry(10),ends(10)
205           integer tailp
206           common /args/arg1,arg2,arg3,arg4,arg5,arg6,arg7
207           character*10 arg1
208           real*8 arg2,arg3,arg4,arg5,arg6,arg7
209           integer i(10)

210           i(nest) = entry(number)
211     100   if(i(nest) .eq. ends(number)) return
212           arg1 = darg1(i(nest))
213           arg2 = darg2(i(nest))
214           arg3 = darg3(i(nest))
215           arg4 = darg4(i(nest))
216           arg5 = darg5(i(nest))
217           arg6 = darg6(i(nest))
218           arg7 = darg7(i(nest))
219           call proc
220           i(nest) = i(nest) + 1
221           goto 100
222           end


      c
      c       printm(matrx)
      c
      c       printm prints out the given 4x4 double precision matrix
      c
223           subroutine printm(matrx)
224           real*8 matrx(4,4)
225           integer i

226           do 100 i=1,4
227                   write(6,800)matrx(i,1),matrx(i,2),matrx(i,3),matrx(i,4)
228     100   continue
229     800   format(4f15.4)
230           return
231           end
```

```
      c
      c       pline(x,y,z,a,b,c,s)
      c
      c       pline() draws a line from (x,y,z) to (a,b,c) with pencode s, using
      c       the current window, viewpoint, viewport, etc.
      c
 1            subroutine pline(x,y,z,a,b,c,s)
 2            real*8 x,y,z,a,b,c
 3            integer s
 4            common /matrix/currm,view,curp
 5            real*8 currm(4,4),view(4,4),curp(4)
 6            logical zclipp,junk
 7            real*8 tmpf(4),tmpt(4)

 8            tmpf(1) = x
 9            tmpf(2) = y
10            tmpf(3) = z
11            tmpf(4) = 1.
12            tmpt(1) = a
13            tmpt(2) = b
14            tmpt(3) = c
15            tmpt(4) = 1.
16            curp(1) = a
17            curp(2) = b
18            curp(3) = c
19            curp(4) = 1.
      c
      c  perform translations, and viewing translation
      c

20            call mmultl(tmpf,currm,tmpf)
21            call mmultl(tmpt,currm,tmpt)
22            call mmultl(tmpf,view,tmpf)
23            call mmultl(tmpt,view,tmpt)
      c
      c  perform zclipping on both points...
      c
24            if(zclipp(tmpf,tmpt).eq. .false.) goto 200
25            junk=zclipp(tmpt,tmpf)
      c
      c  project the vector into 2-D
      c
26            call projct(tmpf)
27            call projct(tmpt)
      c
      c  do x/y clipping, the window to viewport transform, and plot the vector
      c
28            call wtovp(tmpf,tmpt,s)
29    200     return
30            end
```

```
      c
      c       pplot(x,y,z,t)
      c
      c       plot a line from the current position to (x,y,z) using pencode t.
      c       Basically, sets up a call to pline from the current position
      c       to the new position using the appropriate pencode.
      c
31            subroutine pplot(x,y,z,t)
32            real*8 x,y,z
33            integer t
34            common /matrix/currm,view,curp
35            real*8 currm(4,4),view(4,4),curp(4)

36            call pline(curp(1),curp(2),curp(3),x,y,z,t)
37            return
38            end


      c
      c       push(matrix)
      c
      c       push() pushes the given matrix onto the matrix stack, checks
      c       for stack overflow, and won't let you!!!!  Does not alter the
      c       current matrix.
      c
39            subroutine push(matrix)
40            real*8 matrix,sspace(4,4,10)
41            integer stackp
42            dimension matrix(4,4)
43            common /stacks/stackp,sspace

44            if(stackp .gt. 10) then
45                    print *,'stack overflow'
46                    return
47            end if
48            call copym(sspace(1,1,stackp),matrix)
49            stackp=stackp+1
50            return
51            end


      c
      c       pop(matrix)
      c
      c       pop() pops the top of stack into the given matrix.  Checks for
      c       stack underflow, and again won't let you do it!!!!
      c
52            subroutine pop(matrix)
53            real*8 matrix,sspace(4,4,10)
54            integer stackp
55            dimension matrix(4,4)
56            common /stacks/stackp,sspace

57            stackp=stackp-1
58            if(stackp .lt. 1) then
59                    print *,'stack underflow'
60                    stackp = 1
61                    return
62            end if
63            call copym(matrix,sspace(1,1,stackp))
64            return
65            end
```

```
      c
      c       rotate(x,y,z,matrix)
      c
      c       rotate() pre-concatenates the given (x,y,z) rotation, to the
      c       supplied matrix(usually the current matrix).  x,y,z are given
      c              in degrees.
      c
 1            subroutine rotate(x,y,z,matrix)
 2            real*8 x,y,z,matrix
 3            dimension matrix(4,4)
 4            real*8 tmp
 5            dimension tmp(4,4)

 6            call ident(tmp)
 7            tmp(2,2) = cos(x * 0.01745329)
 8            tmp(3,3) = tmp(2,2)
 9            tmp(2,3) = sin(x * 0.01745329)
10            tmp(3,2) = - tmp(2,3)

11            call mmult4(tmp,matrix,matrix)

12            call ident(tmp)
13            tmp(1,1) = cos(y * 0.01745329)
14            tmp(3,3) = tmp(1,1)
15            tmp(3,1) = sin(y * 0.01745329)
16            tmp(1,3) = - tmp(3,1)

17            call mmult4(tmp,matrix,matrix)

18            call ident(tmp)
19            tmp(1,1) = cos(z * 0.01745329)
20            tmp(2,2) = tmp(1,1)
21            tmp(1,2) = sin(z * 0.01745329)
22            tmp(2,1) = - tmp(1,2)

23            call mmult4(tmp,matrix,matrix)

24            return
25            end


      c
      c       translate(x,y,z,matrix)
      c
      c       translate() pre-concatenates the given tranlation (x,y,z) to the
      c       given matrix(usually the current matrix).
      c
26            subroutine transl(x,y,z,matrix)
27            real*8 x,y,z,matrix
28            dimension matrix(4,4)

29            real*8 tmp
30            dimension tmp(4,4)

31            call ident(tmp)
32            tmp(4,1) = x
33            tmp(4,2) = y
34            tmp(4,3) = z

35            call mmult4(tmp,matrix,matrix)

36            return
37            end
```

```
c
c       pscale(x,y,z,matrix)
c
c       pscale pre-concatenates the given scaling (x,y,z) onto the
c       given matrix.
c
38      subroutine pscale(x,y,z,matrix)
39      real*8 x,y,z,matrix
40      dimension matrix(4,4)

41      real*8 tmp
42      dimension tmp(4,4)

43      call ident(tmp)
44      tmp(1,1) = x
45      tmp(2,2) = y
46      tmp(3,3) = z

47      call mmult4(tmp,matrix,matrix)

48      return
49      end

c
c       window(a,b) viewport(a,b,c,d)
c
c       these two routines set up the global variables according to the
c       given parameters.
c
50      subroutine window(a,b)
51      real*8 a,b
52      real*8 wxh,wyh
53      common /windoe/wxh,wyh

54      wxh = a
55      wyh = b
56      return
57      end

c
58      subroutine viewpr(a,b,c,d)
59      real*8 a,b,c,d
60      real*8 vxh,vyh,vxc,vyc
61      common /viewp/vxh,vyh,vxc,vyc

62      vxc = a
63      vyc = b
64      vxh = c
65      vyh = d
66      call mplot(vxc - vxh,vyc - vyh,3)
67      call mplot(vxc + vxh,vyc - vyh,2)
68      call mplot(vxc + vxh,vyc + vyh,2)
69      call mplot(vxc - vxh,vyc + vyh,2)
70      call mplot(vxc - vxh,vyc - vyh,2)
71      return
72      end
```

```
      c
      c       viewpoint(a,b,c,d,e,f)
      c
      c       viewpoint sets up the viewing transformation for the given
      c       to and from points---the eye position is (a,b,c) the lookat
      c       position is (d,e,f).
      c
 1            subroutine viewpn(a,b,c,d,e,f)
 2            real*8 a,b,c,d,e,f

 3            real*8 angle
 4            real*8 tmp(4,4),tmpp(4)
 5            common /matrix/currm,view,curp
 6            real*8 currm(4,4),view(4,4),curp(4)
 7            common /clip/hither,yon,dee
 8            real*8 hither,yon,dee

      c
      c                initialize the viewing transformation
      c
 9            call ident(view)
      c
      c                move lookat position to origin
      c
10            call transl(-d,-e,-f,view)
      c
      c                rotate view matrix per the lookat angle
      c
11            a = a - d
12            b = b - e
13            c = c - f
14            angle = - atan2(a,c)
15            call ident(tmp)
16            tmp(1,1) = cos(angle)
17            tmp(3,3) = tmp(1,1)
18            tmp(3,1) = sin(angle)
19            tmp(1,3) = - tmp(3,1)

20            call mmult4(view,tmp,view)

21            angle = atan2(b,sqrt(a*a + c*c))
22            call ident(tmp)
23            tmp(2,2) = cos(angle)
24            tmp(3,3) = tmp(2,2)
25            tmp(2,3) = sin(angle)
26            tmp(3,2) = - tmp(2,3)

27            call mmult4(view,tmp,view)

28            a = a + d
29            b = b + e
30            c = c + f
31            tmpp(1) = a
32            tmpp(2) = b
33            tmpp(3) = c
34            tmpp(4) = 1.

35            call mmult1(tmpp,view,tmpp)

36            dee = tmpp(3)
37            return
38            end
```

```
      c
      c       zclip(a,b)
      c
      c       zclip() sets up the global clipping parameters, a is the hither,
      c       b the yon, does not allow the hither plane to be behind the
      c       viewer, nor does it allow the yon to be between the viewer
      c       and the hither.
      c
39            subroutine zclip(a,b)
40            real*8 a,b
41            real*8 hither,yon,dee
42            common /clip/hither,yon,dee

43            if(a .lt. 0) then
44                    print *,'bad hither parameter'
45                    a = 0
46            end if
47            if(b .lt. a) then
48                    print *,'bad yon parameter'
49                    b = a + 100
50            end if
51            hither = a
52            yon = b
53            return
54            end


      c
      c       zclipping(vect1,vect2)
      c
      c       zclipping() performs the zclipping on vect1 using the global
      c       zclipping parameters.  Modifies ONLY vect1, returns true if
      c       a portion of the vector indicated by (clipped)vect1 and vect2
      c       will be visible in the scene.
      c
55            logical function zclipp(vect1,vect2)
56            real*8 vect1(4),vect2(4)
57            common /clip/hither,yon,dee
58            real*8 hither,yon,dee

59            real*8 htr,yn

60            htr = dee - hither
61            yn = dee - yon

62            zclipp = .true.

63            if(vect1(3) .gt. htr) then
64                    if(vect2(3) .gt. htr) then
65                            zclipp = .false.
66                    else
      c
      c       you must modify the x and y parameters (according to like triangles)
      c       when the z parameter is modified!!!
      c
67                            vect1(1) = (vect1(1) - vect2(1))*((htr - vect2(3))/
          *                                 (vect1(3) - vect2(3))) + vect2(1)
68                            vect1(2) = (vect1(2) - vect2(2))*((htr - vect2(3))/
          *                                 (vect1(3) - vect2(3))) + vect2(2)
69                            vect1(3) = htr
70                            zclipp = .true.
71                    end if
72            else if(vect1(3) .lt. yn) then
73                    if(vect2(3) .lt. yn) then
74                            zclipp = .false.
75                    else
```

```
76                          vectl(1) = (vect2(1) - vectl(1))*((yn - vectl(3))/
     *                                   (vect2(3) - vectl(3))) + vectl(1)
77                          vectl(2) = (vect2(2) - vectl(2))*((yn - vectl(3))/
     *                                   (vect2(3) - vectl(3))) + vectl(2)
78                          vectl(3) = yn
79                          zclipp = .true.
80                  end if
81            end if
82            return
83            end

      c
      c       projct(vector)
      c
      c       projct() projects the given vector to a point in 2-D space using
      c       the global "dee" parameter, for single point perspective.
      c
 1            subroutine projct(vector)
 2            real*8 vector(4)
 3            common /clip/hither,yon,dee
 4            real*8 hither,yon,dee
 5            real*8 tmp(4,4)

 6            call ident(tmp)

 7            if(dee .ne. 0) then
 8                    tmp(3,4) = - 1 / dee
 9            else
10                    tmp(3,4) = - 1000000000.
11            endif

12            call mmultl(vector,tmp,vector)
13            call norm(vector)
14            return
15            end


      c
      c       norm(vector)
      c
      c       norm() normalizes the given vector.
      c
      c
16            subroutine norm(vector)
17            real*8 vector(4)

18            vector(1) = vector(1) / vector(4)
19            vector(2) = vector(2) / vector(4)
20            vector(3) = vector(3) / vector(4)
21            vector(4) = 1.
22            return
23            end
```

```
      c
      c       wtovp(from,to,pencode)
      c
      c       wtovp() takes the projected from and to points, and:
      c               1: does x/y clipping on the window
      c               2: does the window to viewport translation
      c               3: plots the transformed points onto the device
      c
24            subroutine wtovp(from,to,pencde)
25            real*8 from(4),to(4)
26            integer pencde
27            common /windoe/wxh,wyh
28            real*8 wxh,wyh
29            common /viewp/vxh,vyh,vxc,vyc
30            real*8 vxh,vyh,vxc,vyc
31            logical xyclip
32            real*8 xp,yp

33            if(xyclip(from,to)) then
34                    xp = (from(1)) * vxh / wxh + vxc
35                    yp = (from(2)) * vyh / wyh + vyc

36                    call mplot(xp,yp,3)
37                    xp = (to(1)) * vxh / wxh + vxc
38                    yp = (to(2)) * vyh / wyh + vyc

39                    call mplot(xp,yp,pencde)
40            endif
41            return
42            end


      c
      c       xyclip(from,to)
      c
      c       xyclip() performs the x/y clipping on both the from and t
      c       vectors in the window cooridinates.  Returnes false if
      c       none of the vector would be visible.
      c
43            logical function xyclip(from,to)
44            real*8 from(4),to(4)
45            integer*2 cf,ct

46            xyclip = .false.
47    100             call code(from,cf)
48                    call code(to,ct)
49            if((cf .and. ct) .ne. 0) goto 105

50                    if(cf .ne. 0) call ppush(cf,from,to)
51                    if(ct .ne. 0) call ppush(ct,to,from)
52            if((cf + ct) .ne. 0) goto 100
53            xyclip = .true.

54    105     return
55            end
```

```
      c
      c       code(vector,flag)
      c
      c       code() returns the binary code in flag for vector indicating
      c       it's position relative to the window.
      c
 1            subroutine code(vector,flag)
 2            real*8 vector(4)
 3            integer flag
 4            common /windoe/wxh,wyh
 5            real*8 wxh,wyh
 6            real*8 tmp

 7            flag = 0

 8            tmp = vector(1)
 9            if(tmp .lt. - wxh) flag = 1
10            if(tmp .gt. wxh) flag = flag + 2
11            tmp = vector(2)
12            if(tmp .lt. -wyh) flag = flag + 4
13            if(tmp .gt. wyh) flag = flag + 8
14            return
15            end


      c
      c       ppush(flag,to,from)
      c
      c       ppush() pushes "to" towards "from" according to flag, which
      c       contains the code returned by code().  used to insure that the
      c               line exits the window at the correct point
      c
16            subroutine ppush(flag,to,from)
17            real*8 to(4),from(4)
18            integer flag
19            common /windoe/wxh,wyh
20            real*8 wxh,wyh

21            if((flag .and. 1) .ne. 0) then

22                    to(2) = ((-wxh - from(1))
          *                   /(to(1) - from(1)))*(to(2) - from(2)) + from(2)
23                    to(1) = -wxh
24              endif
25              if((flag .and. 2) .ne. 0) then

26                    to(2) = ((wxh - from(1))
          *                   /(to(1) - from(1)))*(to(2) - from(2)) + from(2)
27                    to(1) = wxh
28              endif
29              if((flag .and. 4) .ne. 0) then

30                    to(1) = ((-wyh - from(2))
          *                   /(to(2) - from(2)))*(to(1) - from(1)) + from(1)
31                    to(2) = -wyh
32              endif
33              if((flag .and. 8) .ne. 0) then

34                    to(1) = ((wyh - from(2))
          *                   /(to(2) - from(2)))*(to(1) - from(1)) + from(1)
35                    to(2) = wyh
36              endif
37              return
38              end
```

```
      c
      c       copym(dst,src)
      c
      c       copym() copies the src 4X4 matrix to the dst 4X4 matrix.
      c
39            subroutine copym(dst,src)
40            real*8 dst(16),src(16)

41            integer i

42            do 100 i = 1,16
43            dst(i) = src(i)
44      100   continue
45            return
46            end


      c
      c       mplot(arg1,arg2,arg3)
      c
      c       mplot() calls plot with arg1,arg2,arg3.  inserted as another level
      c               of indirection in order to allow the actual plot commands to be
      c               written to a file, etc.
      c
47            subroutine mplot(arg1,arg2,arg3)
48            real*8 arg1,arg2
49            integer arg3

50            call plot(arg1,arg2,arg3)
51            return
52            end


      c
      c       cube(arg1,arg2,arg3,arg4,arg5,arg6)
      c
      c       cube() generates a cube centered at (arg1,arg2,arg3) with
      c       arg4,arg5,arg6 as it's half widths
      c
 1            subroutine cube(arg1,arg2,arg3,arg4,arg5,arg6)
 2            real*8 arg1,arg2,arg3,arg4,arg5,arg6

 3            call pline(arg1-arg4,arg2-arg5,arg3-arg6,arg1+arg4,arg2-arg5,arg3-arg6,2)
 4            call pplot(arg1+arg4,arg2+arg5,arg3-arg6,2)
 5            call pplot(arg1-arg4,arg2+arg5,arg3-arg6,2)
 6            call pplot(arg1-arg4,arg2-arg5,arg3-arg6,2)
 7            call pplot(arg1-arg4,arg2-arg5,arg3+arg6,2)
 8            call pplot(arg1+arg4,arg2-arg5,arg3+arg6,2)
 9            call pplot(arg1+arg4,arg2+arg5,arg3+arg6,2)
10            call pplot(arg1-arg4,arg2+arg5,arg3+arg6,2)
11            call pplot(arg1-arg4,arg2-arg5,arg3+arg6,2)
12            call pline(arg1+arg4,arg2-arg5,arg3-arg6,arg1+arg4,arg2-arg5,arg3+arg6,2)
13            call pline(arg1+arg4,arg2+arg5,arg3-arg6,arg1+arg4,arg2+arg5,arg3+arg6,2)
14            call pline(arg1-arg4,arg2+arg5,arg3-arg6,arg1-arg4,arg2+arg5,arg3+arg6,2)
15            return
16            end
```

```
      c
      c
      c       arrow()
      c
      c       arrow() draws a sort-of arrow from (0,0,0) to (1,0,0)
      c
17            subroutine arrow()

18            call pline(0.,0.,0.,1.,0.,0.,2)
19            call pline(1.,0.,0.,.8,.2,0.,2)
20            call pline(1.,0.,0.,.8,0.,.2,2)
21            call pline(1.,0.,0.,.8,-.2,0.,2)
22            call pline(1.,0.,0.,.8,0.,-.2,2)
23            return
24            end


      c
      c       pyrmd(arg1,arg2,arg3,arg4,arg5,arg6)
      c
      c       pyrmd() draws a pyramid with the center of it's base at
      c       (arg1,arg2,arg3) and half x,y,z widths of arg4,arg5,arg6.
      c       The height is the x half width.
      c
25            subroutine pyrmd(arg1,arg2,arg3,arg4,arg5,arg6)
26            real*8 arg1,arg2,arg3,arg4,arg5,arg6

27            real*8 height

28            call pline(arg1-arg4,arg2-arg5,arg3-arg6,arg1+arg4,arg2-arg5,arg3-arg6,2)
29            call pplot(arg1+arg4,arg2+arg5,arg3-arg6,2)
30            call pplot(arg1-arg4,arg2+arg5,arg3-arg6,2)
31            call pplot(arg1-arg4,arg2-arg5,arg3-arg6,2)

32            height = arg4 - arg1
33            call pline(arg1-arg4,arg2-arg5,arg3-arg6,arg1,arg2,arg3+height,2)
34            call pline(arg1+arg4,arg2-arg5,arg3-arg6,arg1,arg2,arg3+height,2)
35            call pline(arg1-arg4,arg2+arg5,arg3-arg6,arg1,arg2,arg3+height,2)
36            call pline(arg1+arg4,arg2+arg5,arg3-arg6,arg1,arg2,arg3+height,2)
37            return
38            end
```

```
      c
      c               subroutine mmult4(mp1,mp2,mpr)
      c
      c               subroutine mmult4 multiplies the mp1 4x4 matrix
      c               and multiplies it by the mp2 4x4 matrix.  the result is
      c               placed in the mpr 4x4 matrix.  internal results are placed
      c               in a temporary matrix, then copied over in order that one of
      c               the operands may be used as the destination matrix
      c
 1            subroutine mmult4(mp1,mp2,mpr)
 2            real*8 mp1(4,4),mp2(4,4),mpr(4,4)
 3            real*8 acc
 4            real*8 temp(4,4)
 5            integer i,j,k

 6            do 100 i=1,4
 7                    do 100 j=1,4
 8                            acc = 0.
 9                            do 110 k=1,4
10                                    acc = acc + mp1(i,k)*mp2(k,j)
11     110                    continue
12                            temp(i,j) = acc
13     100    continue
14            do 120 i=1,4
15                    do 120 j=1,4
16                            mpr(i,j) = temp(i,j)
17     120    continue
18            return
19            end


      c
      c               subroutine mmult1(mp1,mp2,mpr)
      c
      c               subroutine mmult1 multiplies the mp1 4 position vector
      c               by the mp2 4x4 matrix.  the result is put in the mpr 4
      c               position vector.  results are calculated into a temporary
      c               vector, then copied over so that the mp1 vector may be used
      c               as the destination of the result
      c
20            subroutine mmult1(mp1,mp2,mpr)
21            real*8 mp1(4),mp2(4,4),mpr(4)
22            real*8 acc
23            real*8 temp(4)
24            integer i,j,k

25            do 100 j=1,4
26                    acc = 0.
27                    do 110 k=1,4
28                            acc = acc + mp1(k)*mp2(k,j)
29     110                    continue
30                    temp(j) = acc
31     100    continue
32            do 120 i=1,4
33            mpr(i) = temp(i)
34     120    continue
35            return
36            end
```

```
      c
      c       subroutine plot(x,y,penc)
      c
      c               subroutine plot plots a line from the current pen position to
      c               the given pen position using the pencode given.  The possible
      c               pen codes are:
      c                       2: pen down
      c                       3: pen up
      c                       999: terminate plotting
      c
      c               the actual interface described here if for the serial port on the
      c               iSBC 86/12a board connected to an HP7225A flat bed plotter.  no
      c               handshaking is done.
      c
 1            subroutine plot(x,y,penc)
 2            real*8 x,y
 3            integer penc
 4            common /penpos/xpos,ypos,pcount
 5            real*8 xpos,ypos
 6            integer*4 pcount

 7            pcount = pcount + 1

 8            if(penc .eq. 999) then
 9                    call putout('P')
10                    call putout('U')
11                    call putout(';')
12                    print *,'the number of points plotted is:',pcount
13                    goto 200
14            endif
15            if((xpos.eq.x).and.(ypos.eq.y)) then
16                    if(penc .eq. 2) then
17                            call putout('P')
18                            call putout('D')
19                            call putout(';')
20                            call putout('P')
21                            call putout('U')
22                            call putout(';')
23                    else
24                            goto 200
25                    endif

26            else
27                    if(penc .eq. 3) then
28                            call putout('P')
29                            call putout('U')
30                    else if(penc .eq. 2) then
31                            call putout('P')
32                            call putout('D')
33                    else
34                            call putout('P')
35                            call putout('U')
36                            call putout(';')
37                            goto 200
38                    endif
```

```
39                      call putout(';')
40                      call putout('P')
41                      call putout('A')
42                      if(x .gt. 12) x = 12
43                      call ponum(x)
44                      call putout(',')
45                      if(y .gt. 10) y = 10
46                      call ponum(y)
47                      call putout(';')
48              endif
49              xpos = x
50              ypos = y
51      200     return
52              end


        c
        c       subroutine ponum(number)
        c
        c               subroutine ponum takes the given double precision real number,
        c               truncates it to integer, then runs the resultant integer out
        c               the iSBC 86/12a serial port.  leading zeros are suppressed.
        c               the maximum number is 99999!!!
        c
53              subroutine ponum(number)
54              real*8 number
55              character lookup(9)
56              logical flag
57              integer multip(5)
58              integer work

59              data lookup/'1','2','3','4','5','6','7','8','9'/
60              data multip/10000,1000,100,10,1/
61              flag = .false.
62              if(number .lt. 0) number = 0.
63              number = number * 800.
64              do 100 i = 1,5
65              work = aint(number / real(multip(i)))
66              if(work .eq. 0) then
67                      if(flag) call putout('0')
68              else
69                      call putout(lookup(work))
70                      flag = .true.
71              endif
72              number = number - work * multip(i)
73      100     continue
74              if(.not. flag) call putout('0')
75              return
76              end
```

```
      c
      c       subroutine plots
      c
      c               subroutine plots initialized the iSBC86/12 board baud rate
      c               generator(really part of the 8253 timer) and serial line.
      c               the given numbers will set it up for 600 baud, 8 bits, no
      c               parity
      c
 77           subroutine plots
 78           common /penpos/xpos,ypos
 79           real*8 xpos,ypos

 80           xpos = 10000.
 81           ypos = 10000.
 82           call output(#0d6h,intl(#0b6h))
 83           call output(#0d4h,intl(#80h))
 84           call output(#0d4h,intl(0))

 85           call output(#0dah,intl(#72h))
 86           call wastet
 87           call output(#0dah,intl(#25h))
 88           call wastet
 89           call output(#0dah,intl(#62h))
 90           call wastet
 91           call output(#0dah,intl(#0ceh))
 92           call wastet
 93           call output(#0dah,intl(#27h))
 94           return
 95           end


      c
      c       subroutine putout(c)
      c
      c               subroutine putout puts the character given out on the iSBC 86/12
      c               board serial line (checks for transmitter empty, loops on not empty,
      c               on empty puts out the character)
      c
 96           subroutine putout(c)
 97           character c
 98           integer*1 status

 99   100             call input(#0dah,status)
100                  status = status .and. 4
101           if(status .eq. 0) goto 100

102           call output(#0d8h,intl(ichar(c)))
103           return
104           end


      c
      c       subroutine wastet
      c
      c               subroutine wastet wastes a little bit of time while the 8253 gets
      c               its act together
      c
105           subroutine wastet
106           return
107           end
```

APPENDIX B

```
run :f5:graph
'define' 1 /
'viewport' 2.5 2.5 2 2 /
'viewpoint' 10 10 10 0 0 0 /
'window' 10 10 /
'cube' 0 0 0 2 2 2 /
'viewport' 7.5 2.5 2 2 /
'rotate' 15 15 15 /
'cube' 0 0 0 2 2 2 /
'viewport' 2.5 7.5 2 2 /
'ident' /
'viewpoint' 10 0 0 0 0 0 /
'cube' 0 0 0 2 2 2 /
'viewport' 7.5 7.5 2 2 /
'rotate' 30 30 30 /
'cube' 0 0 0 2 2 2 /
'enddef' /
'call' 1 /
'end' /
```

1. INTRODUCTION

As state of the art technology has increased the number of transistors possible on a single integrated circuit, these devices have attained new, higher levels of both performance and functionality. Riding this crest are the Intel 80186 and 80286 microprocessors. While the 80286 has added memory protection and management to the basic 8086 architecture, the 80186 has integrated six separate functional blocks into a single device.

The purpose of this note is to explain, through example, the use of the 80186 with various peripheral and memory devices. Because the 80186 integrates a DMA unit, timer unit, interrupt controller unit, bus controller unit and chip select and ready generation unit with the CPU on a single chip (see Figure 1), system construction is simplified since many of the peripheral interfaces are integrated onto the device.

The 80186 family actually consists of two processors: the 80186 and 80188. The only difference between the two processors is that the 80186 maintains a 16-bit external data bus while the 80188 has an 8-bit external data bus. Internally, they both implement the same processor with the same integrated peripheral components. Thus, except where noted, all 80186 information in this note also applies to the 80188. The implications of having an 8-bit external data bus on the 80188 are explicitly noted in appendix I. Any parametric values included in this note are taken from the iAPX 186 Advance Information data sheet, and pertain to 8Mhz devices.

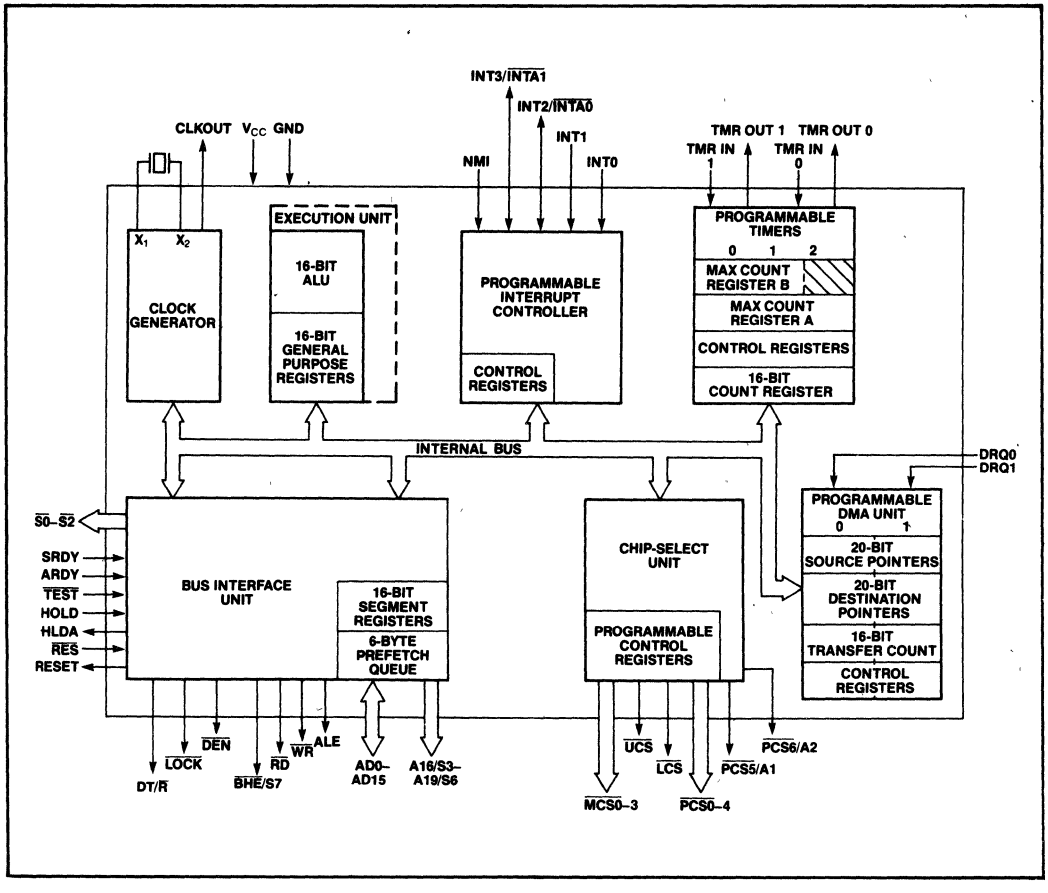


Figure 1. 80186 Block Diagram

2. OVERVIEW OF THE 80186

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088 and 80286. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX,BX, CX,DX) which may be used as operands in most arithmetic operations in either 8 or 16 bit units. It also features four 16-bit "pointer" registers (SI,DI,BP,SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS,DS,SS,ES) are provided allowing simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is left shifted 4 bits and then is added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 2). Any carry out of this addition is ignored. The result of this addition is a 20-bit physical address which is presented to the system memory.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides a wealth of conditional branch and other control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is executing, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another instruction, it takes it out of the queue. Effective processor throughput is increased by adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 provides many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce object code sizes for the 80186. These new instructions are also included in the 80286. A complete description of the architecture and instruction execution of the 80186 can be found in volume I of the iAPX86/186 users manual. The algorithms for the new instructions are also given in appendix H of this note.

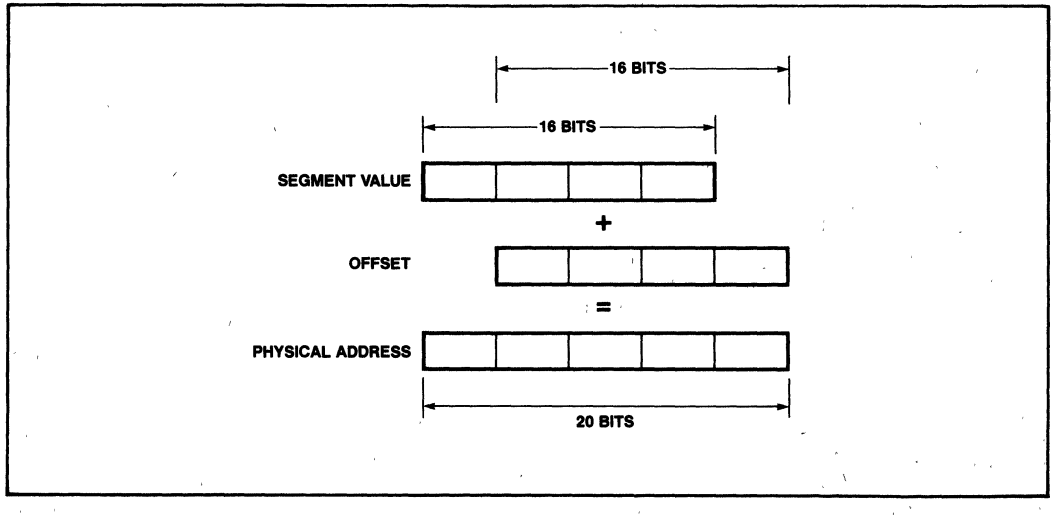


Figure 2. Physical Address Generation in the 80186

2.3 DMA Unit

The 80186 includes a DMA unit which provides two high speed DMA channels. This DMA unit will perform transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data to an internal register, and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations. This is normally difficult, since odd data bytes are transferred on the upper 8 data bits of the 16-bit data bus, while even data bytes are transferred on the lower 8 data bits of the data bus.

Each DMA channel maintains independent 20-bit source and destination pointers which are used to access the source and destination of the data transferred. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be independently incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which may be used to terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The 80186 includes a timer unit which contains 3 independent 16-bit timer/counters. Two of these timers can be used to count external events, to provide waveforms derived from either the CPU clock or an external clock of any duty cycle, or to interrupt the CPU after a specified number of timer "events." The third timer counts only CPU clocks and can be used to interrupt the CPU after a programmable number of CPU clocks, to give a count pulse to either or both of the other two timers after a programmable number of CPU clocks, or to give a DMA request pulse to the integrated DMA unit after a programmable number of CPU clocks.

2.5 Interrupt Controller

The 80186 includes an interrupt controller. This controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to two external 8259A interrupt controllers. In addition, it can be configured as a slave controller to an external interrupt controller to allow complete compatibility with an 80130, 80150, and the iRMX® 86 operating system.

2.6 Clock Generator

The 80186 includes a clock generator and crystal oscillator. The crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all 80186 system timing derives. The CPU clock is externally available, and all timing parameters are referenced to this externally available signal. The clock generator also provides ready synchronization for the processor.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 8086 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in either memory or I/O space in order that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These ready bits control an integrated wait state generator. This allows a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, each set of ready bits includes a bit which determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0-3 and peripheral devices 4-6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals. For more information concerning the interfacing and accessing of the integrated 80186 peripherals not included in this note, please consult the 80186 data sheet, or volume II of the iAPX86/186 users manual.

3. USING THE 80186

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to the 8086 bus structure. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate the speed access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles initiated by the 80186 integrated DMA unit.

In the following discussion, all timing values given are for an 8 MHz 80186. Future speed selections of the part may have different values for the various parameters.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1, T_2, T_3, T_w and T_4. Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number or wait states (T_w) inserted in each bus cycle. This number can vary from 0 to positive infinity.

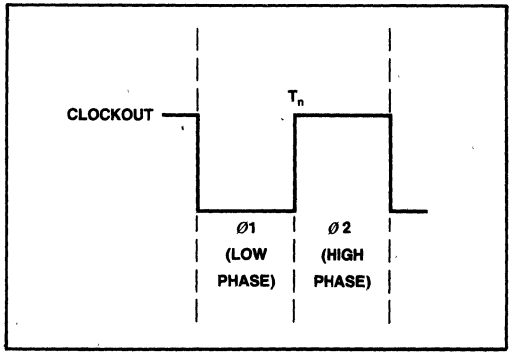


Figure 3. T-state in the 80186

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) which occur during the low and high levels of the CPU clock respectively (see Figure 3).

Different types of bus activity occur for all of the T-states (see Figure 4). Address generation information occurs during T_1, data generation during T_2, T_3, T_w and

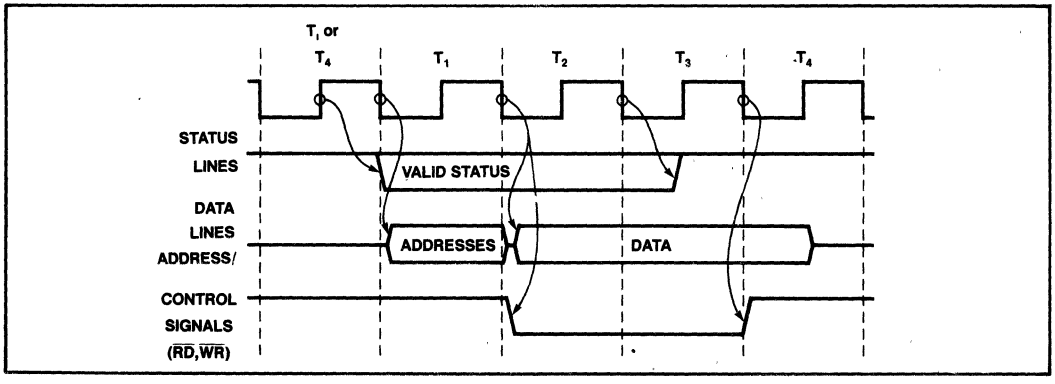


Figure 4. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0-AD15
address/status	A16/S3-A19-S6,$\overline{BHE}$/S7
co-processor control	$\overline{TEST}$
local bus arbitration	HOLD,HLDA
local bus control	ALE,$\overline{RD}$,$\overline{WR}$,DT/$\overline{R}$,$\overline{DEN}$
multi-master bus	$\overline{LOCK}$
ready (wait) interface	SRDY,ARDY
status information	$\overline{S0}$-$\overline{S2}$

T_4. The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_i). Because information concerning an impending bus cycle occurs during the T-state immediately before the first T-state of the cycle itself, two different types of T_4 and T_i can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediatly followed by an idle T state.

During the first type of T_4 or T_i, status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} (55ns) after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_i the status outputs remain inactive (high), since no bus cycle is to be started. This means that the decision per the nature of a T_4 or T_i state (i.e., whether it is immediately followed by a T_i or a T_1) is decided at the beginning of the T-state immediately preceding the T_4 or T_i (see Figure 5). This has consequences for the bus latency time (see section 3.3.2 on bus latency).

3.1.2 PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 6 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater then t_{CLAV} (44ns) after the beginning of T_1, and remain valid at least t_{CLAX} (10ns) after the end of T_1. The ALE signal is driven high in the middle of the T state (either T_4 or T_i) immediately preceding T_1 and is driven low in the middle of T_1, no sooner than t_{AVAL} (30 ns) after addresses become valid. This parameter (t_{AVAL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} (30 ns) after ALE goes inactive to satisfy address latch hold times of strobe inactive to address invalid.

Because ALE goes high long before addresses become valid, the delay through the address latches will be chiefly the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. For the Intel 8282 latch, this parameter is t_{IVOV}, the input valid to output valid delay when strobe is held active (high). Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 8288 bus controller, and keeps it high throughout the 8086 or 8288 ALE high time (i.e., the 80186 ALE pulse is wider).

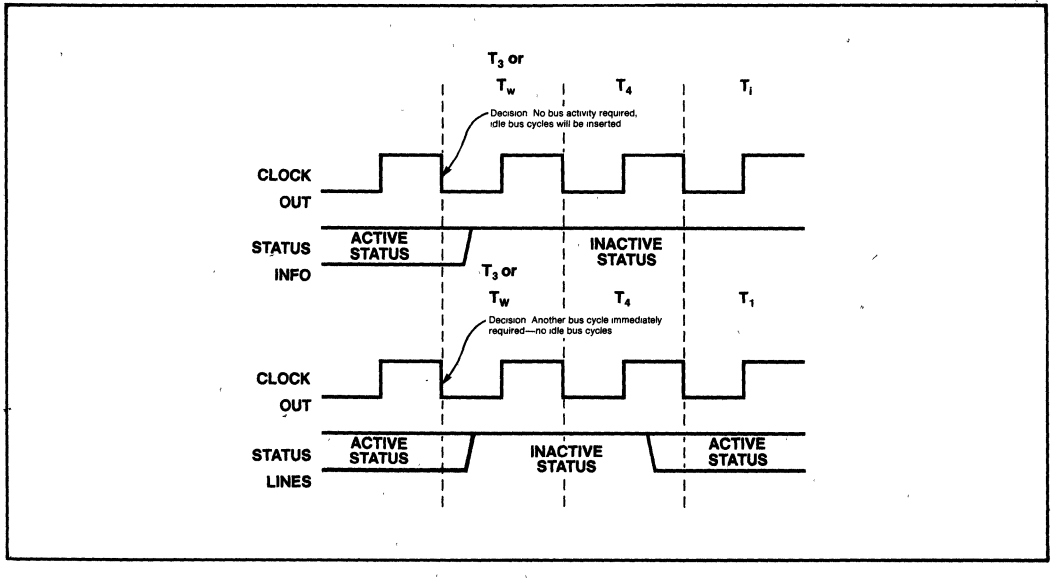
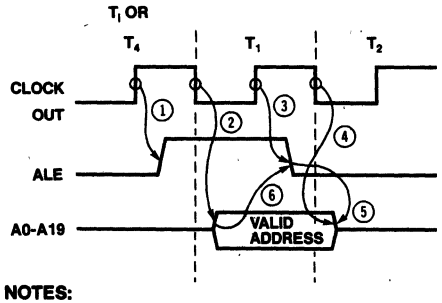


Figure 5. Active-Inactive Status Transitions in the 80186



NOTES:
1. t_{CHLH}: Clock high to ALE high=35 ns max
2. t_{CLAV}: Clock low to address valid=44 ns max
3. t_{CHLL}: Clock high to ALE low=35 ns max
4. t_{CLAX}: Clock low to address invalid (address hold from clock low)=10 ns min
5. t_{LLAX}: ALE low to address invalid (address hold from ALE)=30 ns min
6. t_{AVAL}: Address valid to ALE low (address setup to ALE)=30 ns min

Figure 6. Address Generation Timing of the 80186

A typical circuit for latching physical addresses is shown in Figure 7. This circuit uses 3 8282 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 80186. Typically, the upper 4 address bits are used only to select among various memory components or subsystems, so when the integrated chip selects (see section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time (t_{IVOV}) of the Intel 8282) for the circuit is:

$$t_{CLAV}\ (44ns) + t_{IVOV}\ (30ns) = 74ns$$

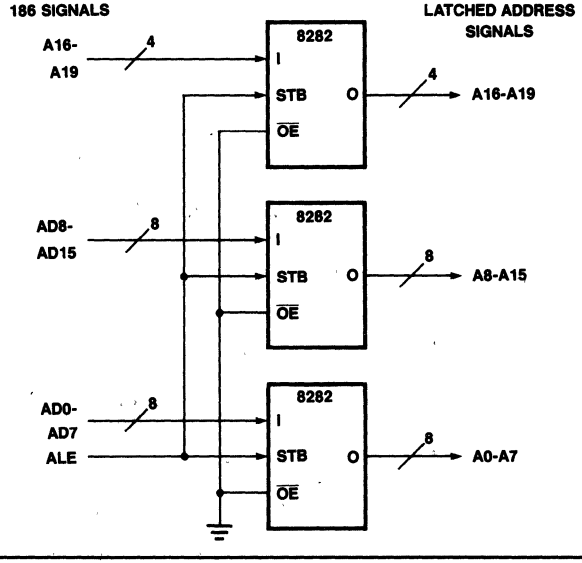


Figure 7. Demultiplexing the Address Bus of the 80186

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. Examples of these are the 80130 and 80150 operating system firmware chips, and the 2186 8K x 8 iRAM. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

One more signal is generated by the 80186 to address memory: $\overline{BHE}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus (see section 3.1.3 on data bus operation section). Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1-A19, NOT A0-A19. This provides 512K unique *word* addresses, or 1M unique BYTE addresses.

Of course, $\overline{BHE}$ is not present on the 8 bit 80188. All data transfers occur on the 8 bits of the data bus.

3.1.3 80186 DATA BUS OPERATION

Throughout T_2, T_3, T_w, and T_4 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either in bytes or in words. All memory is byte addressable, that is, the upper and lower byte of a 16-bit word each have a unique byte address by which they may be individually accessed, even though they share a common word address (see Figure 3-6).

All bytes with even addresses (A0 = 0) reside on the lower 8 bits of the data bus, while all bytes with odd addresses (A0 = 1) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{BHE}$ is driven high, and the data transfer occurs on D0–D7 of the data bus. Whenever an access is made to only the odd byte, $\overline{BHE}$ is driven low, A0 is driven high, and the data transfer occurs on D8–D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{BHE}$ are driven low and the data transfer occurs on D0–D15.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8–D15, the second to access the even byte at the next sequential word address on D0–D7. For example, in Figure 8, byte 0 and byte 1 can be individually accessed (read or written) in two separate bus cycles (byte accesses) to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle (word access) to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the half of the data bus not being accessed is ignored. When byte writes are made, the data driven on the half of the data bus not being written is indeterminate.

3.1.4 80188 DATA BUS OPERATION

Because the 80188 externally has only an 8 bit data bus, the above discussion about upper and lower bytes of the data bus does not apply to the 80188. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access the lower byte of the word; the second to access the upper byte of the word.

Any 80188 access to the integrated peripherals must be done 16 bits at a time: thus in this special case, a word access will occur in a single bus cycle in the 80188. The

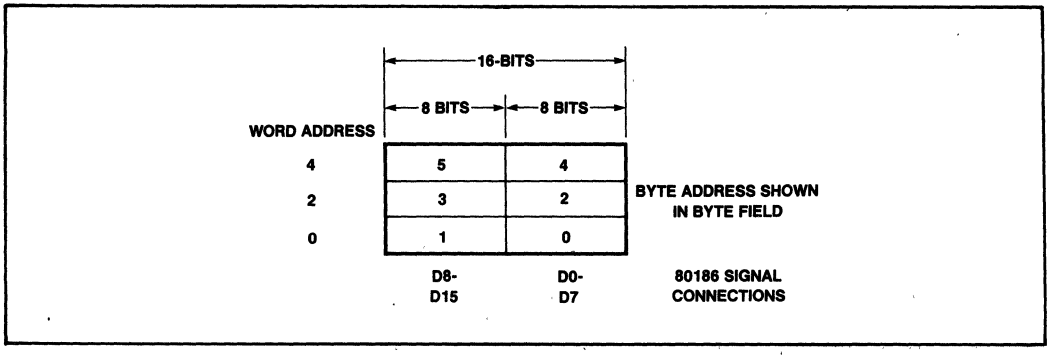


Figure 8. Physical Memory Byte/Word Addressing in the 80186

external data bus will record only a single byte being transferred, however.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186 (200pF, sinking 2mA, sourcing 400uA, roughly twice that of the 8086), this bus may not require additional buffering in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it floats its output drivers after a read *BEFORE* the 80186 begins to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until addresses active for the next bus cycle (t_{RHAV}) which has a minimum value of 85ns. If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving these lines concurrently. Note, this parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

If buffers are required, the 80186 provides a $\overline{DEN}$ (Data ENable) and DT/$\overline{R}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{DEN}$ and DT/$\overline{R}$ signals are activated during all bus cycles, whether or not the cycle addresses buffered devices. The $\overline{DEN}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write) (that is, any time during an active bus cycle when address information is not being generated on the address/data pins). In most systems, the $\overline{DEN}$ signal should NOT be directly connected to the $\overline{OE}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{DEN}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable input of a bi-directional buffer.

The DT/$\overline{R}$ signal determines the direction of data propagation through the bi- directional bus buffers. It is high whenever data is being driven out from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{DEN}$ signal, it may be directly connected to bus buffers, since this signal does not usually directly enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 9. Note that the A side of the 8286 buffer is connected to the 80186, the B side to the external device. The B side of the buffer has greater drive capacity than the A side (since it is meant to drive much greater loads). The DT/$\overline{R}$ signal can directly drive the T (transmit) signal of the buffer, since it has the correct polarity for this configuration.

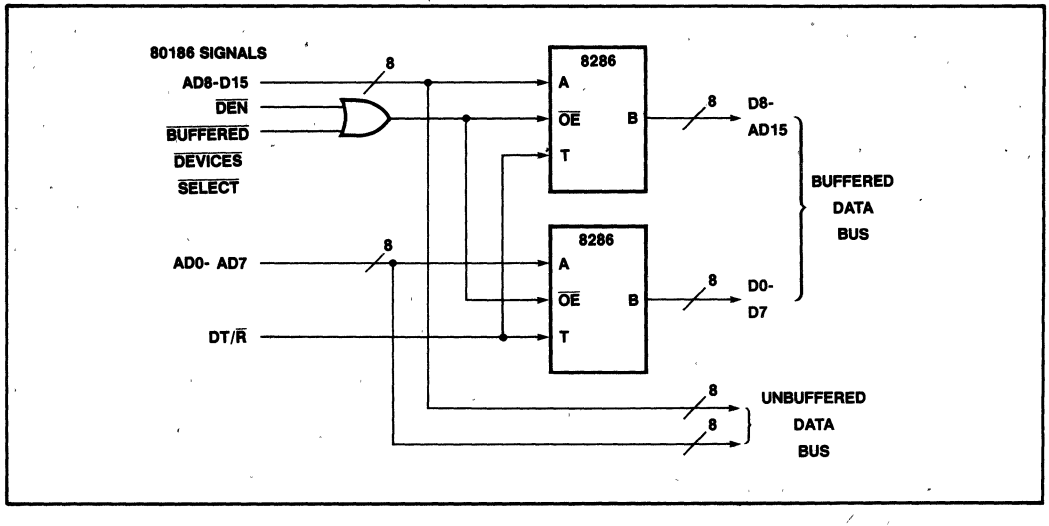
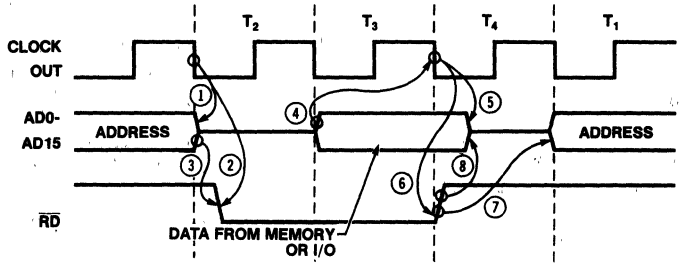


Figure 9. Example 80186 Buffered/Unbuffered Data Bus

AFN-210973



1. t_{CLAZ}: Clock low until address float=35 ns max
2. t_{CLRL}: Clock low until RD active = 70 ns max
3. t_{AZRL}: Address float until RD active = 0 ns min
4. t_{DVCL}: Data valid until clock low (data input set-up time) = 20 ns min*
5. t_{CLDX}: Clock low until data invalid (data input hold time from clock) = 10 ns min*
6. t_{CLRH}: Clock low until RD high = 10 ns min
7. t_{RHAV}: RD high until addresses valid = 85 ns min
8. t_{RHDX}: Read high until data invalid (data input hold from RD) = 0 ns min*

* Input requirements of 80186, all others are output characteristics

Figure 10. Read Cycle Timing of the 80186

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals RD, WR, LOCK and TEST. In addition, the 80186 provides the status signals S0-S2 and S6 from which all other required bus control signals can be generated.

3.1.6.1 RD and WR

The RD and WR signals strobe data to or from memory or I/O space. The RD signal is driven low off the beginning of T_2, and is driven high off the beginning of T_4 during all memory and I/O reads (see Figure 10). RD will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4. RD will not go inactive until the processor's data hold time (10ns) has been satisfied.

Note that the 80186 does not provide separate I/O and memory RD signals. If separate I/O read and memory read signals are required, they can be synthesized using the S2 signal (which is low for all I/O operations and high for all memory operations) and the RD signal (see Figure 11). It should be noted that if this approach is used, the S2 signal will require latching, since the S2 signal (like S0 and S1) goes to a passive state well before the beginning of T_4 (where RD goes inactive). If S2 was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as S2 goes to the passive state (high). The status signals may be latched using ALE in an identical fashion as is used to latch the address signals (often using the spare bits in the address latches).

Often the lack of a seperate I/O and memory RD signal

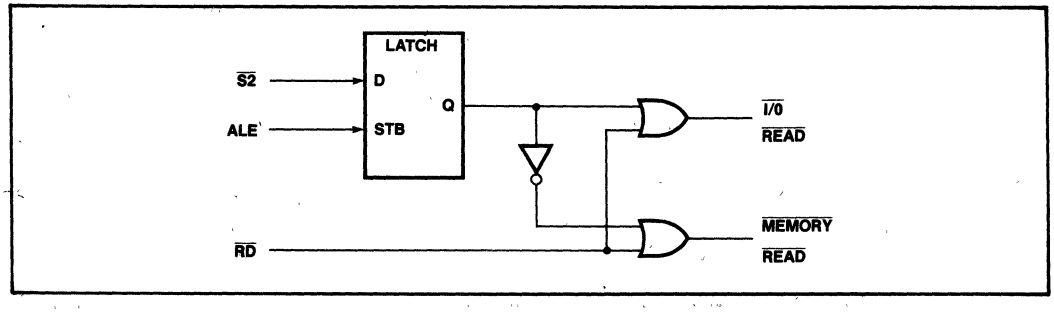
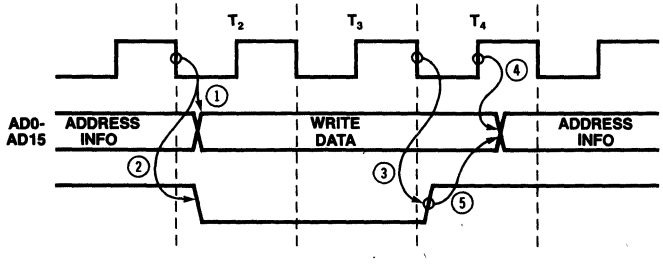


Figure 11. Generating I/O and Memory Read Signals from the 80186

is not important in an 80186 system. Each of the 80186 chip select signals will respond on only one of memory or I/O accesses (the memory chip selects respond only to accesses memory space; the peripheral chip selects can respond to accesses in either I/O or memory space, at programmer option). Thus, the chip select signal enables the external device only during accesses to the proper address in the proper space.

The $\overline{WR}$ signal is also driven low off the beginning of T_2 and driven high off the beginning of T_4. Like the $\overline{RD}$ signal, the $\overline{WR}$ signal is active for all memory and I/O writes, and also like the $\overline{RD}$ signal, separate I/O and memory writes may be generated using the latched $\overline{S2}$ signal along with the $\overline{WR}$ signal (see Figure 12). More importantly, however, is the active going edge of write. At the time $\overline{WR}$ makes its active (high to low) transition, valid write data is NOT present on the data bus. This has consequences when using this signal as a write enable signal for DRAMs and iRAMs since both of these devices require that the write data be stable on the data bus at the time of the inactive to active transition of the $\overline{WE}$ signal. In DRAM applications, this problem is solved by a DRAM controller (such as the Intel 8207 or 8203), while with iRAMs this problem may be solved by placing cross-coupled NAND gates between the CPU and the iRAMs on the $\overline{WR}$ line (see Figure 13). This will delay the active going edge of the $\overline{WR}$ signal to the iRAMs by a clock phase, allowing valid data to be driven onto the data bus.



1. t_{CLDV}: Clock low until data valid = 44 ns max
2. t_{CVCTV}: Clock low until $\overline{WR}$ active = 70 ns max
3. t_{CVCTX}: Clock low until $\overline{WR}$ inactive = 55 ns max
4. t_{CHDX}: Clock high until data invalid = 10 ns min
5. $\overline{WR}$ inactive until data invalid = t_{CLCH} min $- t_{CVCTX} + t_{CHDX}$
   $= 55 - 55 + 10$
   $= 10$ ns

Figure 12. Write Cycle Timing of the 80186

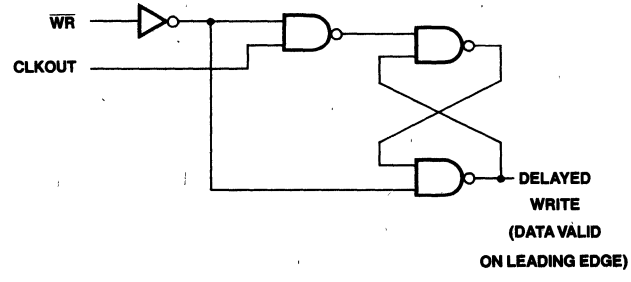


Figure 13. Synthesizing Delayed Write from the 80186

3.1.6.2 Queue Status Signals

If the $\overline{RD}$ line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter "queue status" mode. When in this mode, the $\overline{WR}$ and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 ($\overline{WR}$) are given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines $\overline{S0}$-$\overline{S2}$ using an external 8288 bus controller (see below). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device. $\overline{RD}$ is the ONLY three-state or input pin on the 80186 which is supplied with a pullup or pulldown device.

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which are used to indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 5). The possible status line encodings and their interpretations are given in Table 3. The status lines are driven to their inactive state in the T state (T_3 or T_w) immediately preceding T_4 of the current bus cycle.

The status lines may be directly connected to an 8288 bus controller, which can be used to provide local bus control signals or multi-bus control signals (see Figure 14). Use of the 8288 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals may be used to provide local bus control signals, while an 8288 is used to provide multi-bus control signals, for example.

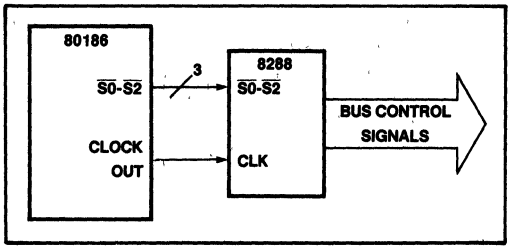


Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S2}$	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The 80186 provides two additional status signals: S6 and S7. S7 is equivalent to $\overline{BHE}$ (see section 3.1.2) and appears on the same pin as $\overline{BHE}$. $\overline{BHE}$/S7 changes state, reflecting the bus cycle about to be run, in the middle of the T state (T_4 or T_i) immediately preceding T_1 of the bus cycle. This means that $\overline{BHE}$/S7 does not need to be latched, i.e., it may be used directly as the $\overline{BHE}$ signal. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2, T_3, T_4 and T_w. In the 8086 family, all central processors (e.g., the 8086, 8088 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transfering data for the DMA unit.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address derives. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will *ALWAYS* be low.

3.1.6.4 $\overline{TEST}$ and $\overline{LOCK}$

Finally, the 80186 provides a $\overline{TEST}$ input and a $\overline{LOCK}$ output. The $\overline{TEST}$ input is used in conjunction with the

processor WAIT instruction. It is typically driven by a processor extension (like the 8087) to indicate whether it is busy. Then, by executing the WAIT (or FWAIT) instruction, the central processor may be forced to temporarily suspend program execution until the processor extension indicates that it is idle by driving the TEST line low.

The $\overline{LOCK}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. This signal is used to indicate to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources (see AP-106, "Multiprogramming with the iAPX88 and iAPX86 Microsystems," by George Alexy (Sept. 1980)).

On the 80186, the $\overline{LOCK}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive 3 T-states after the beginning of the last DATA cycle of the locked transfer. On the 8086, the $\overline{LOCK}$ signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the $\overline{LOCK}$ signal before the first LOCKED data cycle is performed. Since $\overline{LOCK}$ is active before the processor requires the bus for the data transfer, opcode pre-fetching can be LOCKED. However, since the 80186 does not activate the $\overline{LOCK}$ signal until the processor is ready to actually perform the locked transfer, locked pre-fetching will not occur with the 80186.

Note that the $\overline{LOCK}$ signal does not remain active until the end of the last data cycle of the locked transfer. This may cause problems in some systems if, for example, the processor requests memory access from a dual ported RAM array and is denied immediate access (because of a DRAM refresh cycle, for example). When the processor finally is able to gain access to the RAM array, it may have already dropped its $\overline{LOCK}$ signal, thus allowing the dual port controller to give the other port access to the RAM array instead. An example circuit which can be used to hold $\overline{LOCK}$ active until a RDY has been received by the 80186 is shown in Figure 15.

3.1.7 HALT TIMING

A HALT bus cycle is used to signal the world that the 80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two important ways.

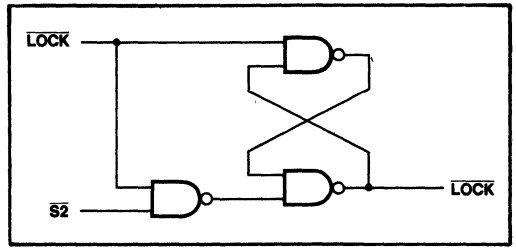


Figure 15. Circuit Holding $\overline{Lock}$ Active Until Ready is Returned

The first way in which a HALT bus cycle differs from a normal bus cycle is that since the processor is entering a halted state, none of the control lines ($\overline{RD}$ or $\overline{WR}$) will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the $\overline{S0}$-$\overline{S2}$ status lines go to their passive state (all high) during T_2 of the bus cycle, well before they go to their passive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the $\overline{S0}$-$\overline{S2}$ status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the DMA transfer will run. In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 bus (see section 4.4.1).

3.1.8 8288 AND 8289 INTERFACING

The 8288 and 8289 are the bus controller and multimaster bus arbitration devices used with the 8086 and 8088. Because the 80186 bus is similar to the 8086 bus, they can be directly used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 8288 bus controller generates control signals ($\overline{RD}$, $\overline{WR}$, ALE, DT/$\overline{R}$, $\overline{DEN}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines $\overline{S0}$-$\overline{S2}$ of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 8288 just as in an 8086 system. Using the 8288 with the 80186 does not prevent using the 80186 control signals directly. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the

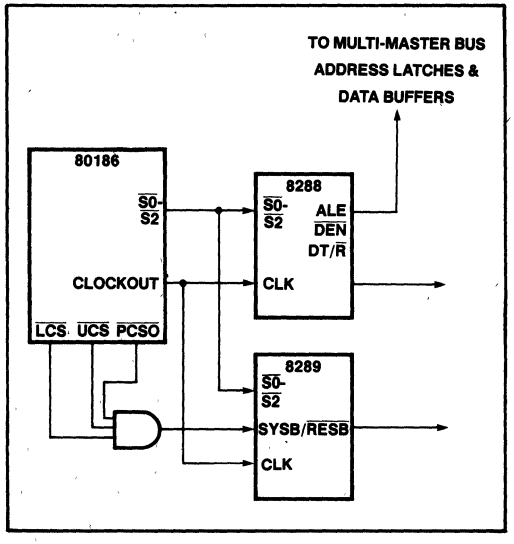


Figure 16. 80186/8288/8289 Interconnection

8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus arbiter arbitrates the use of a multi-master system bus among various devices each of which can become the bus master. This component also decodes status lines $\overline{S0}$-$\overline{S2}$ of the processor directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the processor to insert wait states (T_w) into a CPU bus cycle. This allows slower devices to respond to CPU service requests (reads or writes). Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of ARDY or SRDY need be active to terminate a bus cycle. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers, and to any area where the chip select ready bits indicate that the external ready should be ignored.

The timing required by the two RDY lines is different. The ARDY line is meant to be used with asynchronous ready inputs. Thus, inputs to this line will be internally synchronized to the CPU clock before being presented to the processor. The synchronization circuitry used with the ARDY line is shown in Figure 17. Figure 18A and 18B show valid and invalid transitions of the ARDY line (and subsequent wait state insertion). The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite level (either high or low) before its output is latched into the second flip-

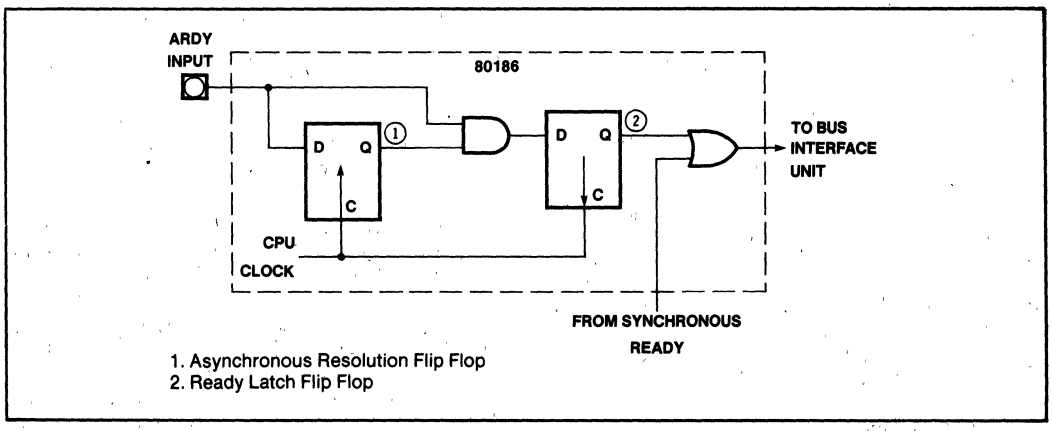
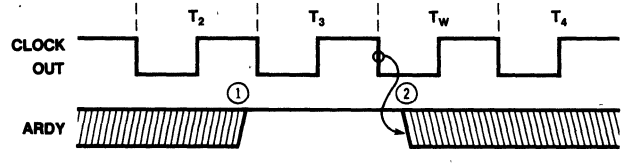


Figure 17. Asynchronous Ready Circuitry of the 80186

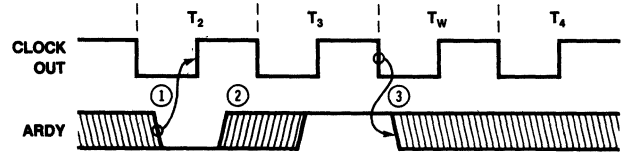
flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (see Appendix B for 80186 synchronizer information).

With this scheme, notice that only the active going edge of the ARDY signal is synchronized. Once the synchronization flip-flop has sampled high, the ARDY input directly drives the RDY flip-flop. Since inputs to this RDY flip-flop must satisfy certain setup and hold times, it is important that these setup and hold times (t_{ARYLCL} = 35ns and t_{CHARYX} = 15 ns respectively) be satisfied by any inactive going transition of the ARDY line. The reason ARDY is implemented in this manner is to allow a slow device the greatest amount of time to respond with a not ready after it has been selected. In a normally ready system, a slow device must respond with a not ready quickly after it has been selected to prevent the processor from continuing and accessing invalid data from the slow device. By implementing ARDY in the above fashion, the slow device has an additional clock phase to respond with a not ready.

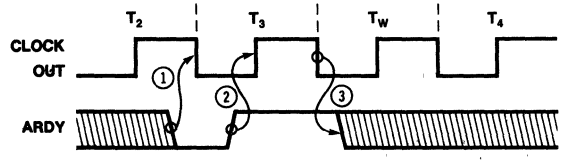
If RDY is sampled active into the RDY flip-flop at the beginning of T_3 or T_w (meaning that ARDY was sam-



1. No set-up or hold times required
2. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns min



1. t_{ARYHCH}: ARDY valid until clock high (ARDY inactive set-up time to clock high) = 20 ns min
2. No set-up or hold time required ONLY if ① is guaranteed
3. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns min



1. t_{ARYLCL}: ARDY low to clock low (ARDY inactive set-up time to clock low) = 35 ns min
   must be satisfied since synchronizing FLIP-FLOP has sampled active
2. t_{ARYHCH}: ARDY high to clock high (ARDY active set-up time) = 20 ns min
   must be satisfied ONLY to guarantee recognition at the next clock (i.e. to guarantee synchronizing FLIP-FLOP will sample ARDY active)
3. t_{CLARYX}: Clock low to ARDY inactive (ARDY active hold time) = 15 ns

Figure 18A. Valid ARDY Transitions

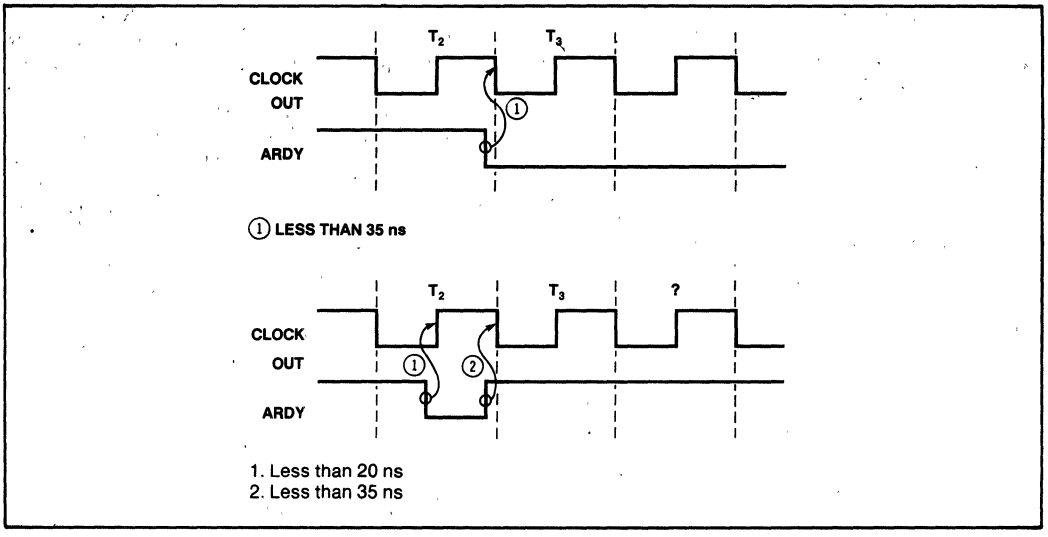


Figure 18B. Invalid ARDY Transitions

pled high into the synchronization flip-flop in the middle of a T state, and has remained high until the beginning of the next T state), that T state will be immediately followed by T_4. If RDY is sampled low into the RDY flip-flop at the beginning of T_3 or T_w (meaning that either ARDY was sampled low into the synchronization flip-flop OR that ARDY was sampled high into the synchronization flip-flop, but has subsequently changed to low before the ARDY setup time) that T state will be immediately followed by a wait state (T_w). Any asynchronous transition on the ARDY line not occurring during the above times, that is, when the processor is not "looking at" the ready lines, will not cause CPU malfunction.

Again, for ARDY to force wait states to be inserted, SRDY must be driven low, since they are internally ORed together to form the processor RDY signal.

The synchronous ready (SRDY) line requires that ALL transitions on this line during T_2, T_3 or T_w satisfy a certain setup and hold time (t_{SRYCL} = 35 ns and t_{CLSRY} = 15 ns respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line, and subsequent wait state insertion is shown in Figure 19. The processor looks at this line at the beginning of each T_3 and T_w. If the line is sampled active at the beginning of either of these two cycles, that cycle will

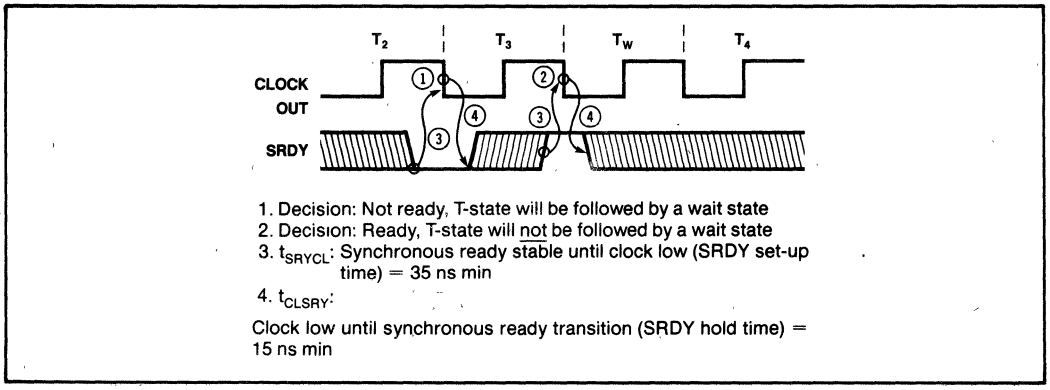


Figure 19. Valid SRDY transitions on the 80186

be immediately followed by T_4. On the other hand, if the line is sampled inactive at the beginning of either of these two cycles, that cycle will be followed by a T_w. Any asynchronous transition on the SRDY line not occurring at the beginning of T_3 or T_w, that is, when the processor is not "looking at" the ready lines will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. This occurs whenever the 80186 internal queue is full and no read/write cycles are being requested by the execution unit or integrated DMA unit. The reader should recall that a separate unit, the bus interface unit, fetches opcodes (including immediate data) from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions or data move instructions, the execution will be bus limited. Here, the execution unit will be required to wait often for an instruction to be fetched before it continues its operation. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

All instruction fetches are word (16-bit) fetches from even addresses unless the fetch occurs as a result of a jump to an odd location. This maximizes the utilization of each bus cycle used for instruction fetching, since each fetch will access two bytes of information. It is also good programming practice to locate all word data at even locations, so that both bytes of the word may be accessed in a single bus cycle (see discussion on data bus interfacing for further information, section 3.1.3 of this note).

Although the amount of bus utilization, i.e., the percentage of bus time used by the 80186 for instruction fetching and execution required for top performance will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. This is caused by the higher performance execution unit requiring instructions from the prefetch queue at a greater rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest of these is the example EPROM interface shown in Figure 20.

The addresses are latched using the address generation circuit shown earlier. Note that the A0 line of each EPROM is connected to the A1 address line from the 80186, NOT the A0 line. Remember, A0 only signals a data transfer on the lower 8 bits of the 16-bit data bus! The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 $\overline{RD}$ signal is used as the $\overline{OE}$ for the EPROMs.

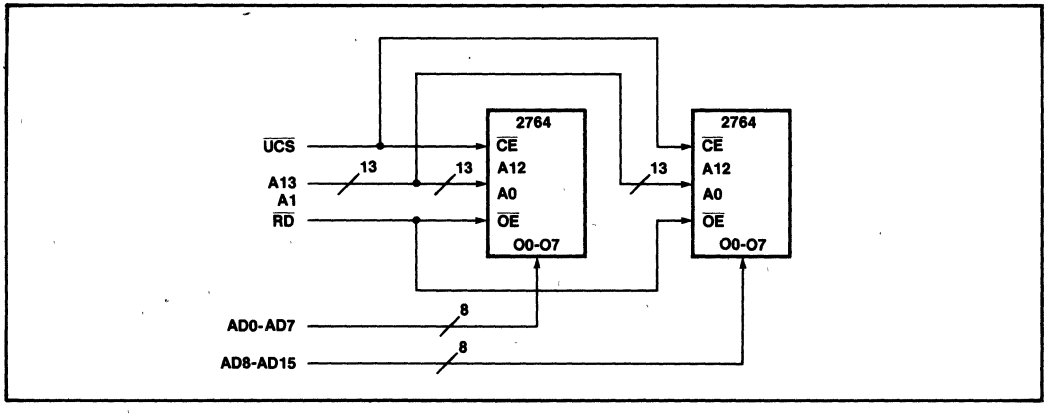


Figure 20. Example 2764/80186 Interface

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

time from
address: $(3 + N)*t_{CLCL} - t_{CLAV} - t_{IVOV}(8282) - t_{DVCL}$
$= 375 + (N * 125) - 44 - 30 - 20$
$= 281 + (N * 125)$ ns

time from
chip select: $(3 + N)*t_{CLCL} - t_{CLCSV} - t_{DVCL}$
$= 375 + (N * 125) - 66 - 20$
$= 289 + (N * 125)$ ns

time from
$\overline{RD}$ ($\overline{OE}$): $(2 + N)t_{CLCL} - t_{CLRL} - t_{DVCL}$
$= 250 + (N * 125) - 70 - 20$
$= 160 + (N * 125)$ ns

where:

t_{CLAV} = time from clock low in T_1 until addresses are valid

t_{CLCL} = clock period of processor

t_{IVOV} = time from input valid of 8282 until output valid of 8282

t_{DVCL} = 186 data valid input setup time until clock low time of T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{RD}$ goes low

N = number of wait states inserted

Thus, for 0 wait state operation, 250ns EPROMs must be used. The only significant parameter not included above is t_{RHAV}, the time from $\overline{RD}$ inactive (high) until the 80186 begins driving address information. This parameter is 85ns, which meets the 2764-25 (250ns speed selection) output float time of 85ns. If slower EPROMs are used, a discrete data buffer *MUST* be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 2186 INTERFACE

An example interface between the 80186 and 2186 iRAMs is shown in Figure 21. This memory component is almost an ideal match with the 80186, because of its large integration, and its not requiring address latching.

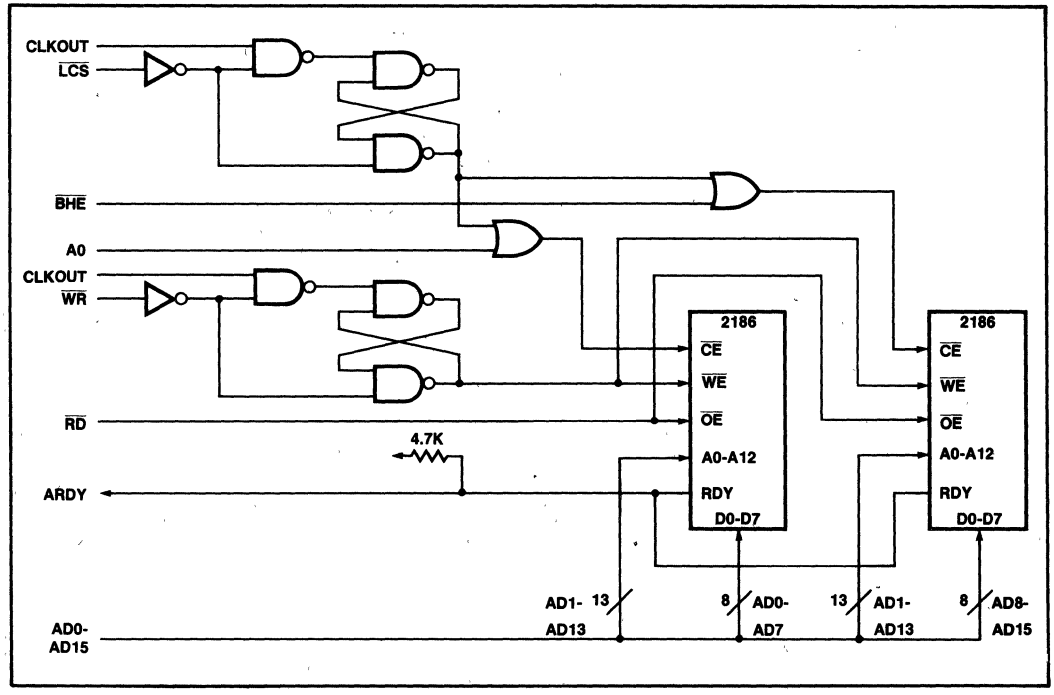


Figure 21. Example 2186/80186 Interface

The 2186 internally is a dynamic RAM integrated with refresh and control circuitry. It operates in two modes, pulse mode and late cycle mode. Pulse mode is entered if the $\overline{CE}$ signal is low to the device a maximum of 130ns, and requires the command input ($\overline{RD}$ or $\overline{WE}$) to go active within 90ns after $\overline{CE}$. Because of these requirements, interfacing the 80186 to the 2186 in pulse mode would be difficult. Instead, the late cycle mode is used. This affords a much simpler interface with no loss of performance. The iRAM automatically selects between these modes by the nature of the control signals.

The 2186 is a leading edge triggered device. This means that address and data information are strobed into the device on the active going (high to low) transition of the command signal. This requires both $\overline{CE}$ and $\overline{WR}$ be delayed until the address and data driven by the 80186 are guaranteed stable. Figure 21 shows a simple circuit which can be used to perform this function. Note that ALE *CANNOT* be used to delay $\overline{CE}$ if addresses are not latched externally, because this would violate the address hold time required by the 2186 (30ns).

Because the 2186s are RAMs, data bus enables ($\overline{BHE}$ and A0, see previous section) MUST be used to factor either the chip enables or write enables of the lower and upper bytes of the 16-bit RAM memory system. If this is not done, all memory writes, including single byte writes, will write to both the upper and lower bytes of the memory system. The example system shown uses $\overline{BHE}$ and A0 as factors to the 2186 $\overline{CE}$. This may be done, because both of these signals (A0 and $\overline{BHE}$) are valid when the address information is valid from the 80186.

The 2186 requires a certain amount of recovery time between its chip enable going inactive and its chip enable going active insure proper operation. For a "normal" cycle (a read or write), this time is t_{EHEL} = 40ns. This means that the 80186 chip select lines will go inactive soon enough at the end of a bus cycle to provide the required recovery time even if two consecutive accesses are made to the iRAMs. If the 2186 $\overline{CE}$ is asserted without a command signal ($\overline{WE}$ or $\overline{OE}$), a "False Memory Cycle" (FMC) will be generated. Whenever a FMC is generated, the recovery time is much longer; another memory cycle must not be initiated for 200ns. As a result, if the memory system will generate FMCs, $\overline{CE}$ must be taken away in the middle of the T state (T_3 or T_w) immediately preceding T_4 to insure two consecutive cycles to the iRAM will not violate this parameter. Status going passive (all high) can be used for this purpose. These lines will all go high during the first phase of the next to last T state (either T_3 or T_w) of a bus cycle (see section 3.1.5).

Finally, since it is a dynamic device, the 2186 requires refresh cycles to maintain data integrity. The circuitry to generate these refresh cycles is integrated within the 2186. Because of this, the 2186 has a ready line which is used to suspend processor operation if a processor RAM access coincides with an internally generated refresh cycle. This is an open collector output, allowing many of them to be wire-OR'ed together, since more than one device may be accessed at at time. These lines are also normally ready, which means that they will be high whenever the 2186 is not being accessed, i.e., they will only be driven low if a processor request coincides with an internal refresh cycle. Thus, the ready lines from the iRAM must be factored into the 80186 RDY circuit only during accesses to the iRAM itself. Since the 2186 refresh logic operates asynchronously to the 80186, this RDY line must be synchronized for proper operation with the 80186, either by the integrated ready synchronizer or by an external circuit. The example circuit uses the integrated synchronizer associated with the ARDY processor input.

The ready lines of the 2186 are active unless a processor access coincides with an internal refresh cycle. These lines must go inactive soon enough after a cycle is requested to insert wait states into the data cycle. The 2186 will drive this line low within 50ns after $\overline{CE}$ is received, which is early enough to force the 80186 to insert wait states if they are required. The primary concern here is that the ARDY line be driven not active before its setup time in the middle of T_2. This is required by the nature of the asynchronous ready synchronization circuitry of the 80186. Since the RDY pulse of the 2186 may be as narrow as 50ns, if ready was returned after the first stage of the synchronizer, and subsequently changed state within the ready setup and hold time of the high to low going edge of the CPU clock at the end of T_2, improper operation may occur (see section 3.1.6).

The example interface shown has a zero wait state RAM read access time from $\overline{CE}$ of:

$$3 * t_{CLCL} - t_{CLCSV} - (\text{TTL delay}) - t_{DVCL}$$
$$= 375 - 66 - 30 - 20 \text{ ns}$$
$$= 259 \text{ ns}$$

where:

t_{CLCL} = CPU clock cycle time

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{DVCL} = 80186 data in setup time before clock low in T_4

The data valid delay from $\overline{OE}$ active is less than 100ns, and is therefore not an access time limiter in this interface. Additionally, the 2186 data float time from $\overline{RD}$ inactive is less than the 85ns 80186 imposed maximum. The $\overline{CE}$ generation circuit shown in Figure 21 provides an address setup time of at least 11ns, and an address hold time of at least 35ns (assuming a maximum two level TTL delay of less than 30ns).

Write cycle address setup and hold times are identical to the read cycle times. The circuit shown provides at least 11ns write data setup and 100ns data hold time from $\overline{WE}$, easily meeting the 0ns setup and 40ns hold times required by the 2186.

For more information concerning 2186 timing and interfacing, please consult the 2186 data sheet, or the application note AP-132, "Designing Memory Systems with the 8Kx8 iRAM" by John Fallin and William Righter (June 1982).

3.2.3 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 22. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface with 64K DRAMs.

All 8203 cycles are generated off control signals ($\overline{RD}$ and $\overline{WR}$) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatibility. Even if a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150ns DRAMs with an 8 MHz 80186, three wait states if 200ns DRAMs are used (see timing analysis, Figure 23).

The entire RAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also be used to insert the wait states required by the interface.

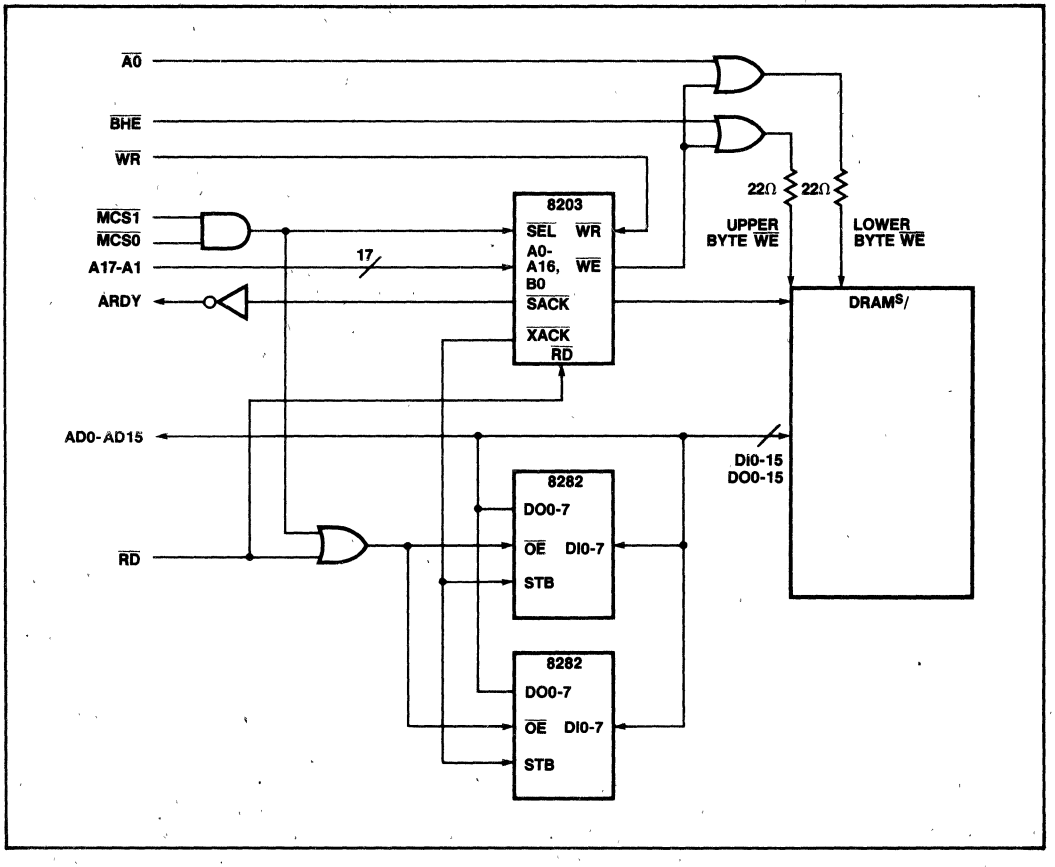


Figure 22. Example 8203/DRAM/80186 Interface

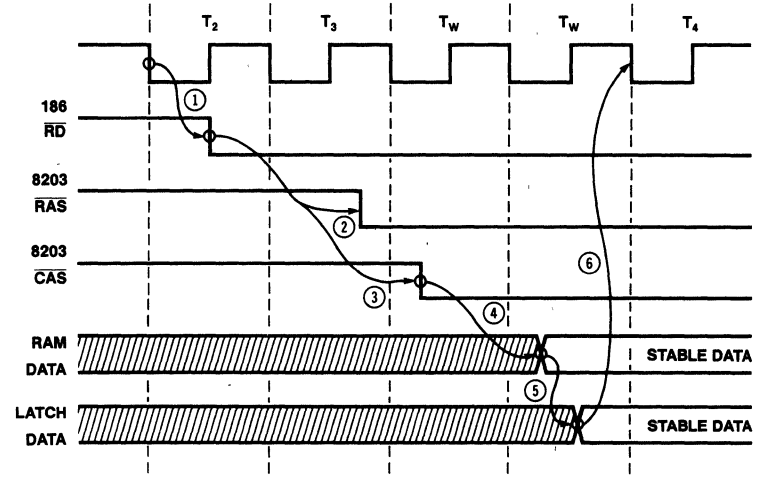


Figure 23. 8203/2164A-15 Access Time Calculation

Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 (used to suspend processor operation when a processor DRAM request coincides with a DRAM refresh cycle) must be synchronized to the 80186. The 80186 ARDY line is used to provide the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. This is fundamentally different than the normally ready mode used by the 2186 iRAMs (see previous section). The 8203 $\overline{\text{SACK}}$ signal is presented to the 80186 only when the DRAM is being accessed. Notice that the $\overline{\text{SACK}}$ output of the 8203 is used, rather than the $\overline{\text{XACK}}$ output. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active, and the time data must be present on the data bus, using the $\overline{\text{XACK}}$ signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

For more information about 8203/DRAM interfacing and timing, please consult the 8203 data sheet, or AP97A, "Interfacing Dynamic RAM to iAPX86/88 Systems Using the Intel 8202A and 8203" by Brad May (April 1982).

3.2.4 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 or 80286 microcomputer systems. This controller provides all address multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 24. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the type of cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLOCKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array driven by the

8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

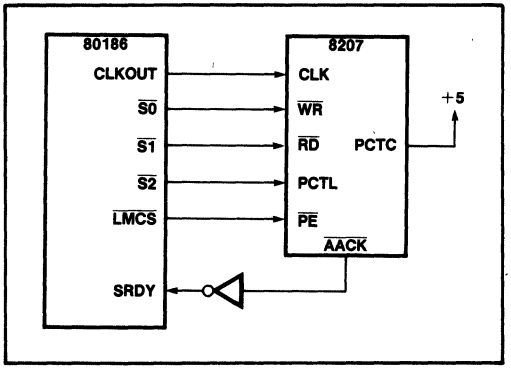


Figure 24. 80186/8207/DRAM Interface

The 8207 AACK signal may be used to generate a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 access cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the AACK line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that only one of SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus master devices (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus to perform bus cycles (memory or I/O reads or writes).

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 25). AD0 - AD15 (address/data 0 - 15) and DEN (data enable) are floated within t_{CLAZ} (35ns) after the same clock edge that HLDA is driven active. A16–A19 (address 16 - 19), RD, WR, BHE (Bus High Enable), DT/R (Data Transmit/Receive) and S0 - S2 (status 0 - 2) are floated within t_{CHCZ} (45ns) after the clock edge immediately before the clock edge on which HLDA comes active.

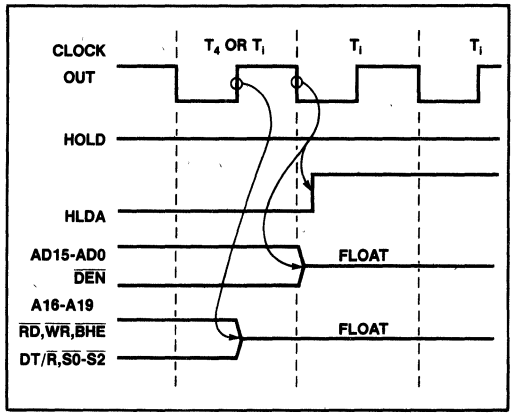


Figure 25. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the chip select lines (UCS, LCS, MCS0-3, and PCS0-6). The designer must be aware that the chip select circuitry does not look at externally generated addresses (see section 10 for a discussion of the chip select logic). Thus, for memory or peripheral devices which are addressed by external bus master devices, discrete chip select and ready generation logic must be used.

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect this latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous signal. To guarantee recognition on a certain clock edge, it must satisfy a certain setup and hold time to the falling

edge of the CPU clock. A full CPU clock cycle is required for this synchronization, that is, the internal HOLD signal is not presented to the internal bus arbitration circuitry until one full clock cycle after it is latched from the HOLD input (see Appendix B for a discussion of 80186 synchronizers). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input; the second signals the internal circuitry to initiate a bus hold. (see Figure 26).

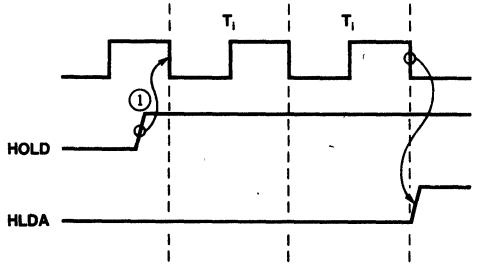
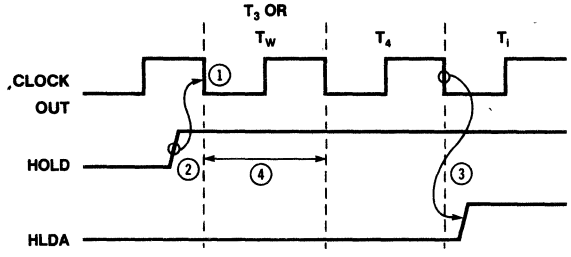


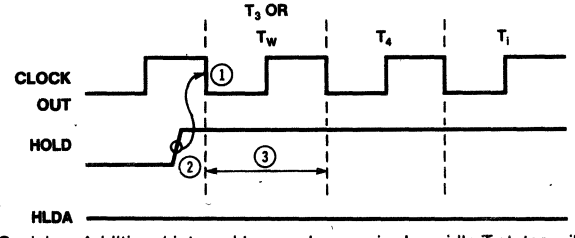
Figure 26. 80186 Idle Bus Hold/HLDA Timing

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These may make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. An idle bus occurs whenever the 80186 is not performing any bus transfers. As stated in section 3.1.1, when the bus is idle, the 80186 generates idle T-states. The bus can become idle only at the end of a bus cycle. Thus, the 80186 can recognize HOLD only after the end of its current bus cycle. The 80186 will normally insert no T_i states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_i states independent of the HOLD input (see section 3.1.7).

When the HOLD request is active, the 80186 will be

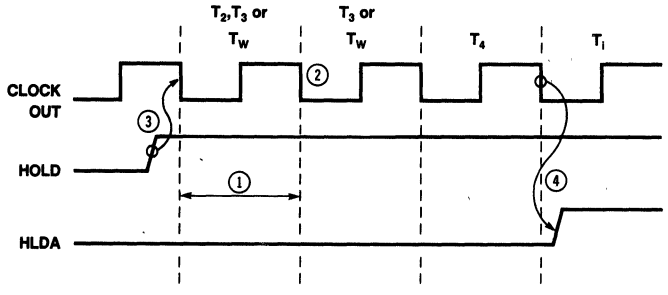


1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than 25 ns (t_{HVCL})
3. Less than 50 ns (t_{CLHAV})
4. HOLD request internally synchronized



1. Decision: Additional internal bus cycles required, no idle T-states will be inserted, Hold not active soon enough to force idle T-states
2. Greater than 25 ns (t_{HVCL}): not required since it will not get recognized anyway
3. HOLD request internally synchronized

Figure 27. HOLD/HLDA Timing in the 80186



1. HOLD request internally synchronized
2. Decision: HOLD request active, idle t-states will be inserted at end of current bus cycle
3. Greater than 25 ns
4. Less than 50 ns

Figure 27A. HOLD/HLDA Timing in the 80186

forced to proceed from T_4 to T_i in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus at the end of the current bus cycle. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 27 graphically portrays the scenarios depicted above.

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required to perform a DMA transfer using the integrated controller. Each of these factors will add additional bus cycle times to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. Any transfers generated by such a prefixed instruction will be locked, and will not be separated by any external bus requesting device. String instructions may be locked. Since string transfers may require thousands of bus cycles, bus latency time will suffer if they are locked.

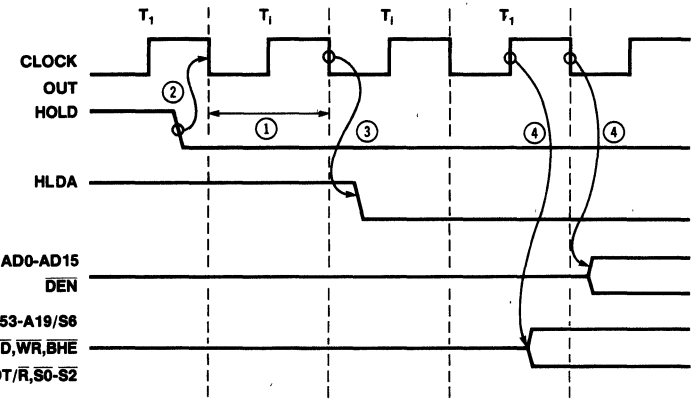
The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in iRMX 86 mode (see section 6.7.4) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by any bus HOLD, either internal or external. See section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

After the 80186 recognizes that the HOLD input has gone inactive, it will drop its HLDA line in a single clock. Figure 28 shows this timing. The 80186 will insert only two T_i after HLDA has gone inactive, assuming that the 80186 has internal bus cycles to run. During the last T_i, status information will go active concerning the bus cycle about to be run (see section 3.1.1). If the 80186 has no pending bus activity, it will maintain all lines floating (high impedance) until the last T_i before it begins its first bus cycle after the HOLD.

3.4 Differences Between the 8086 bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 8288 bus controller and the 8289 bus arbiter) may be used directly with the 80186. There are a few significant differences between the two processors which should be considered.



1. HOLD internally synchronized
2. Greater than 25 ns
3. Less than 50 ns
4. Lines come out of float only if a bus cycle is pending

Figure 28. 80186 Coming out of Hold

- **CPU Duty Cycle and Clock Generator**

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock (1/2 of the time it is high, the other 1/2 of the time it is low). This is different that the 8086, which employs an external clock generator (the 8284A) with a 33% duty cycle CPU clock (1/3 of the time it is high, the other 2/3 of the time, it is low). These differences manifest themselves as follows:

1) No oscillator output is available from the 80186, as it is available from the 8284A clock generator.
2) The 80186 does not provide a PCLK (50% duty cycle, 1/2 CPU clock frequency) output as does the 8284A.
3) The clock low phase of the 80186 is narrower, and the clock high phase is wider than on the same speed 8086.
4) The 80186 does not internally factor AEN with RDY. This means that if both RDY inputs (ARDY and SRDY) are used, external logic must be used to prevent the RDY not connected to a certain device from being driven active during an access to this device (remember, only one RDY input needs to be active to terminate a bus cycle, see section 3.1.6).
5) The 80186 concurrently provides both a single asynchronous ready input and a single synchronous ready input, while the 8284A provides either two synchronous ready inputs or two asynchronous ready inputs as a user strapable option.
6) The CLOCKOUT (CPU clock output signal) drive capacity of the 80186 is less than the CPU clock drive capacity of the 8284A. This means that not as many high speed devices (e.g., Schottky TTL flip-flops) may be connected to this signal as can be used with the 8284A clock output.
7) The crystal or external oscillator used by the 80186 is twice the CPU clock frequency, while the crystal or external oscillator used with the 8284A is three times the CPU clock frequency.

- **Local Bus Controller and Control Signals**

The 80186 simultaneously provides both local bus controller outputs ($\overline{RD}$, $\overline{WR}$, ALE, $\overline{DEN}$ and DT/$\overline{R}$) and status outputs ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$) for use with the 8288 bus controller. This is different from the 8086 where the local bus controller outputs (generated only in min mode) are sacrificed if status outputs (generated only in max mode) are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g.,

a Multibus®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 8288 bus controller to drive the control signals of the system bus.

2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 then on the 8086 or 8288. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.

3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that this queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

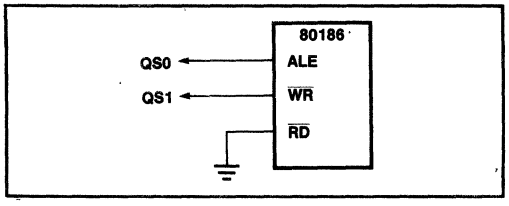


Figure 29. Generating Queue Status Information from the 80186

- **HOLD/HLDA vs. RQ/GT**

As discussed earlier, the 80186 uses a HOLD/HLDA type of protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatiblity with Intel's the new generation of high performance/high integration bus master peripheral devices (for example the 82586 Ethernet* controller or 82730 high performance CRT controller/text coprocessor).

- **Status Information**

The 80186 does not provide S3-S5 status information. On the 8086, S3 and S4 provide information regarding the segment register used to generate the physical address of the currently executing bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status bits are always low on the 80186.

Status signal S6 is used to indicate whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the 80186 integrated DMA unit.

- **Bus Drive**

The 80186 output drivers will drive 200pF loads. This is double that of the 8086 (100pF). This allows larger systems to be constructed without the need for bus buffers. It also means that it is very important to provide good grounds to the 80186, since its large drivers can discharge its outputs very quickly causing large current transients on the 80186 ground pins.

- **Misc.**

The 80186 does not provide early and late write signals, as does the 8288 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 8288. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide differentiated I/O and memory read and write command signals. If these signals are desired, an external 8288 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize differentiated commands (see section 3.1.4).

*Ethernet is a registered trademark of Xerox Corp.

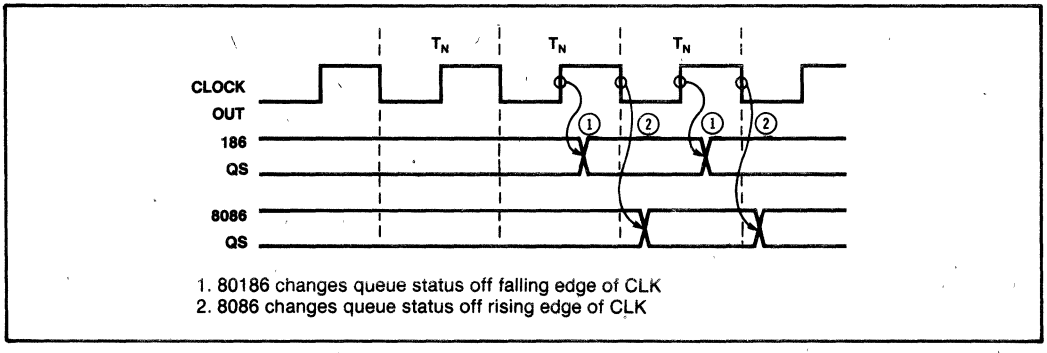


Figure 30. 80186 and 8086 Queue Status Generation

4. DMA UNIT INTERFACING

The 80186 includes a DMA unit which provides two independent high speed DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are exactly the same as bus cycles initiated by the CPU (except that S6 = 1 during all DMA initiated cycles, see section 3.1). Thus interfacing with the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

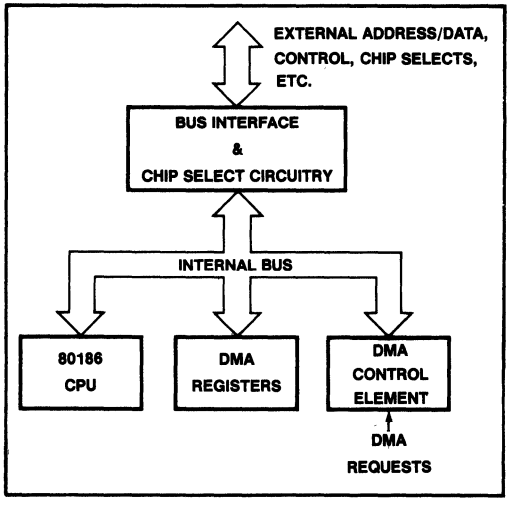


Figure 31. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which are used to access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses
- Programmable generation of DMA requests by:
  1) the source of the data
  2) the destination of the data
  3) timer 2 (see section 5)
  4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers which are used to control channel operation. These registers are included in the 80186 integrated peripheral control block (see appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout and interpretation of the bits in these registers is given in Figure 32.

The 20-bit source and destination pointers allow access to the complete 1 Mbyte address space of the 80186, and that all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA (i.e., the DMA channels address the full 1 Mbyte address space of the 80186 as a flat, linear array without segments). When addressing I/O space, the upper 4 bits of the DMA pointer registers should be programmed to be 0. If they are not programmed 0, then the programmed value (greater than 64K in *I/O space*) will be driven onto the address bus (an area of I/O space not accessable to the CPU). The data transfer will occur correctly, however.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/$\overline{\text{STOP}}$ bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects between byte or word transfers. Two synchronization bits are used to determine the source of the DMA requests (see section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit is used to enable interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

The control register also contains a start/stop (ST/$\overline{\text{STOP}}$) bit. This bit is used to enable DMA transfers. Whenever this bit is set, the channel is

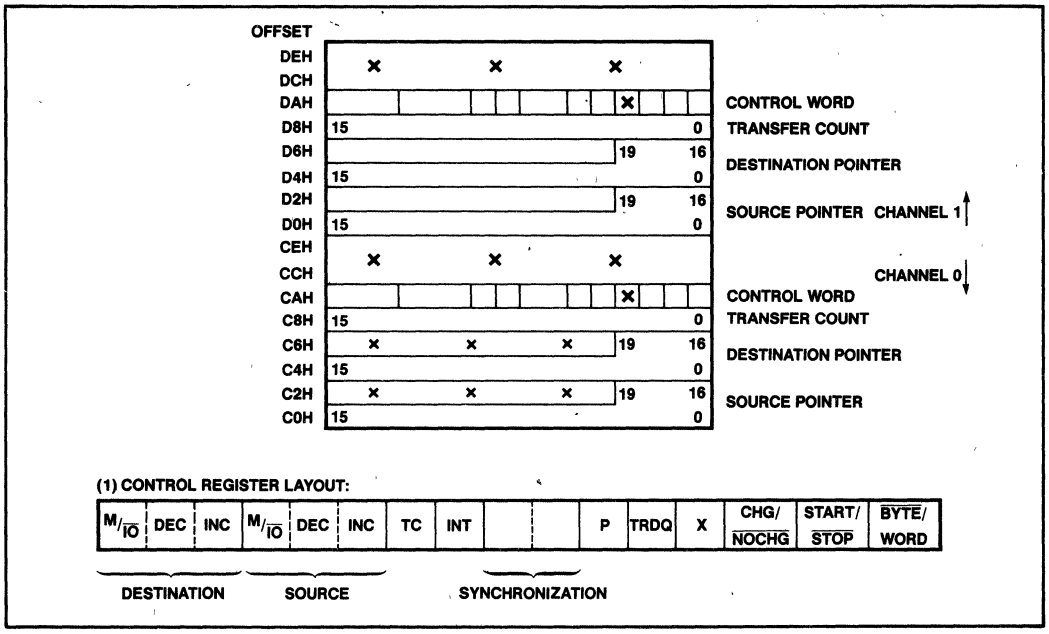


Figure 32. 80186 DMA Register Layout

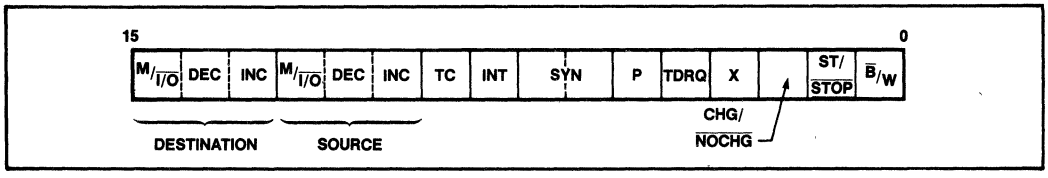
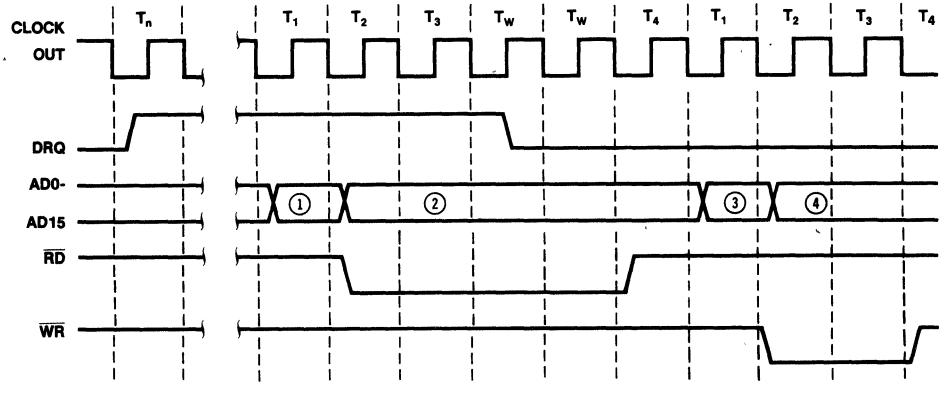


Figure 33. DMA Control Register

"armed," that is, a DMA transfer will occur whenever a DMA request is made to the channel. If this bit is cleared, no DMA transfers will be performed by the channel. A companion bit, the CHG/$\overline{\text{NOCHG}}$ bit, allows the contents of the DMA control register to be changed without modifying the state of the start/stop bit. The ST/$\overline{\text{STOP}}$ bit will only be modified if the CHG/$\overline{\text{NOCHG}}$ bit is also set during the write to the DMA control register. The CHG/$\overline{\text{NOCHG}}$ bit is write only. It will always be read back as a 1. Because DMA transfers could occur immediately after the ST/$\overline{\text{STOP}}$ bit is set, it should only be set only after all other DMA controller registers have been programmed. This bit is automatically cleared when the transfer count register reaches zero and the TC bit in the DMA control register is set, or when the transfer count register reaches zero and unsynchronized DMA transfers are programmed.

All DMA unit programming registers are directly accessable by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/$\overline{\text{STOP}}$ bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only half of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).



1. Source address
2. Source data
3. Destination address
4. Destination data

NOTE: Wait states are inserted by the bus condition during the bus cycle, not by the DMA controller

Figure 34. Example DMA Transfer Cycle on the 80186

4.3 DMA Transfers

Every DMA transfer in the 80186 consists of two independent bus cycles, the fetch cycle and the deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed from memory or I/O space using the address in the source pointer register. The data accessed is placed in an internal temporary register, which is not accessible to the CPU. During the deposit cycle, the byte or word data in this internal register is placed in memory or I/O space using the address in the destination pointer register. These two bus cycles will not be separated by bus HOLD or by the other DMA channel, and one will never be run without the other except when the CPU is RESET. Notice that the bus cycles run by the DMA unit are exactly the same as memory or I/O bus cycles run by the CPU. The only difference between the two is the state of the S6 status line (which is multiplexed on the A19 line): on all CPU initiated bus cycles, this status line will be driven low; on all DMA initiated bus cycles, this status line will be driven high.

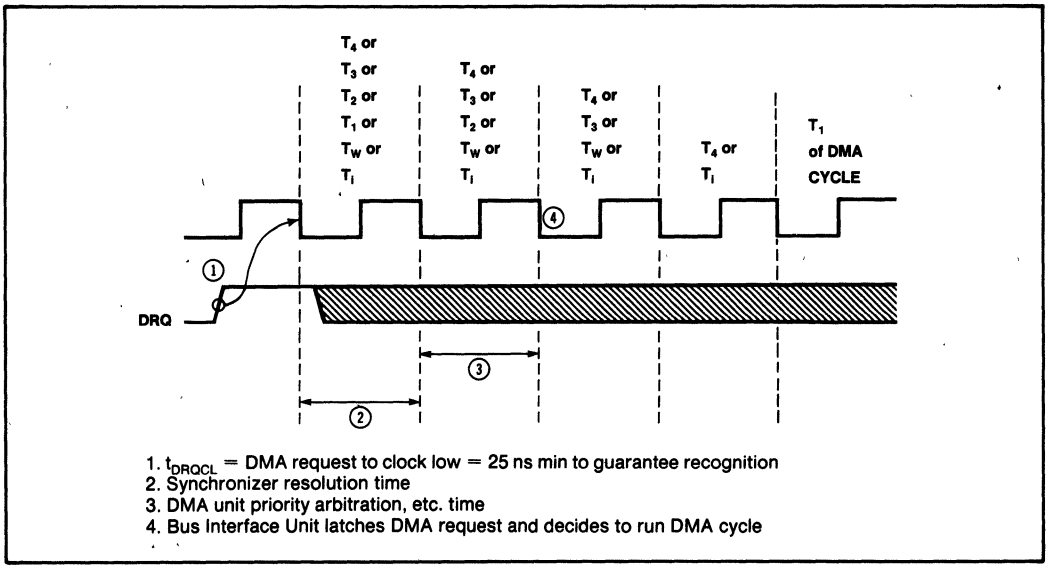
4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source of the DMA data or the destination of the DMA data. All transfer requests on this line are synchronized to the CPU clock before being presented to internal DMA logic. This means that any asynchronous transitions of the DMA request line will not cause the DMA channel to malfunction. In addition to external requests, DMA requests may be generated whenever the internal timer 2 times out, or continuously by programming the synchronization bits in the DMA control register to call for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see section 3.3.2). The only important difference is that external HOLD always has bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA transfer cycles will alternate between the two channels (i.e., the first channel will perform a fetch and



1. t_{DRQCL} = DMA request to clock low = 25 ns min to guarantee recognition
2. Synchronizer resolution time
3. DMA unit priority arbitration, etc. time
4. Bus Interface Unit latches DMA request and decides to run DMA cycle

Figure 35. DMA Request Timing on the 80186 (showing minimum response time to request)

deposit, followed by a fetch and deposit by the second channel, etc).

The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles, that is, a DMA request is sampled 4 clock cycles before the beginning of a bus cycle to determine if any DMA activity will be required. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity (see above).

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor such that any transition on the DMA request line will cause a DMA cycle eventually. Quite the contrary, DMA request must be active at a certain time before the end of a bus cycle for the DMA request to be recognized by the processor. If the DMA request line goes inactive before that window, then no DMA cycles will be run.

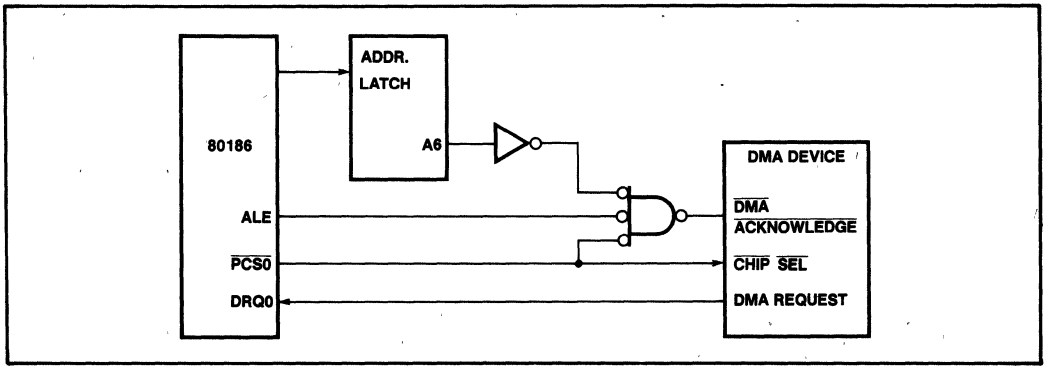


Figure 36. DMA Acknowledge Synthesis from the 80186

4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the $\overline{PCS}$ lines (see Figure 36). Note ALE must be used to factor the DACK because addresses are not guaranteed stable when chip selects go active. This is required because if the address is not stable when the $\overline{PCS}$ goes active, glitches can occur at the output of the DACK generation circuitry as the address lines change state. Once ALE has gone low, the addresses are guaranteed to have been stable for at least t_{AVAL} (30ns).

4.6 Internally Generated DMA Requests

There are two types in internally synchronized DMA transfers, that is, transfer initiated by a unit integrated in the 80186. These two types are transfers in which the DMA request is generated by timer 2, or where DMA request is generated by the DMA channel itself.

The DMA channel can be programmed such that whenever timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, timer 2 must be initiated and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost, i.e., only a single DMA transfer will occur, even though the timer has timed out twice.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, one after the other until the preprogrammed number of DMA transfers (in the DMA transfer count register) have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

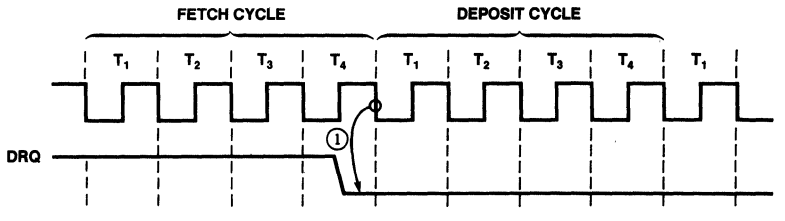
There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes are selected by programming the synchronization bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two source synchronized DMA transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

In a source synchronized DMA transfer, the source of the DMA data requests the DMA cycle. An example of this would be a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at the beginning of the deposit cycle of a DMA transfer. This allows over 3 CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

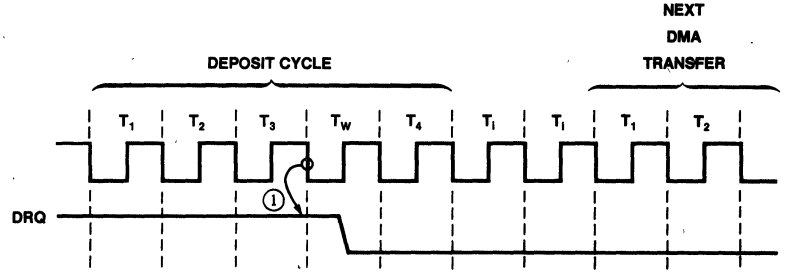
4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the deposit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being



80186 DECISION:

1. Current DMA source synchronized transfer will not be immediately followed by another DMA transfer



80186 Decision:

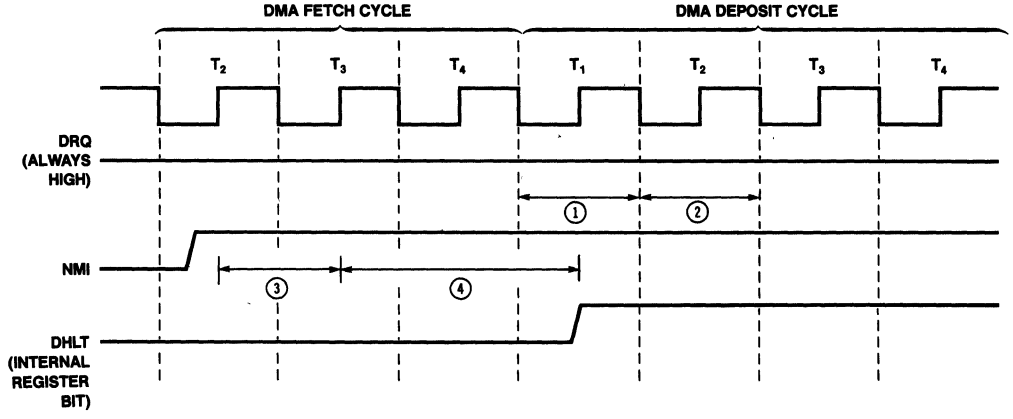
1. Current DMA destination synchronized transfer will be followed immediately by another DMA transfer

Figure 37. Source & Destination Synchronized DMA Request Timing

inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine whether another DMA cycle should be run immediately following the current DMA transfer. To get around this problem, the DMA unit will relinquish the CPU bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the DMA requesting device time to drop its DMA request if it does not immediately desire another immediate DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation (e.g., instruction fetching, memory or I/O reads or writes, etc.) Thus, typically, a CPU initiated bus cycle will be inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however (and none can be guaranteed), the DMA unit will insert only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the DMA destination requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted into the bus cycle. Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted in the deposit cycle of the DMA transfer will lengthen the amount of time from the beginning of the deposit cycle to the time DMA will be sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its DMA request after receiving a DMA acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (100 ns), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its DMA request after receiving the DMA acknowledge. Table 4 shows the amount of time between the beginning of T_2 and the time DMA request is sampled as wait states are inserted in the DMA deposit cycle.

Table 4. DMA Request Inactive Timing

Number of Wait States	Max Time(ns) For DRQ Inactive From Start of T_2
0	100
1	225
2	350
3	475



1. DMA request synchronization
2. Decision: Will DMA cycle be run?
   Answer: No DMA request is active but DHLT is set (from NMI request)
3. NMI synchronization time
4. Logic delay time from synchronized NMI until DHLT set (note: DHLT is in the interrupt control status register)

Figure 38. NMI and DMA Interaction

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended after the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see section 7.3.1). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit should be cleared by the programmer, and DMA activity will resume exactly where it left off, i.e., none of the DMA registers will have been modified. The DMA Halt bit is not automatically reset after the NMI has been serviced. It is automatically reset by the IRET instruction. This DMA halt bit may also be set by the programmer to prevent DMA activity during any critical section of code.

4.9 Example DMA Interfaces

4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA Interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0-AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 driven address information after a read from the 8272 (see section 3.1.3).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to $\overline{PCS2}$. If $\overline{PCS2}$ is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0510H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active. If ALE were not used, the DACK generation circuitry could glitch as address output changed state while the chip select was active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to out-

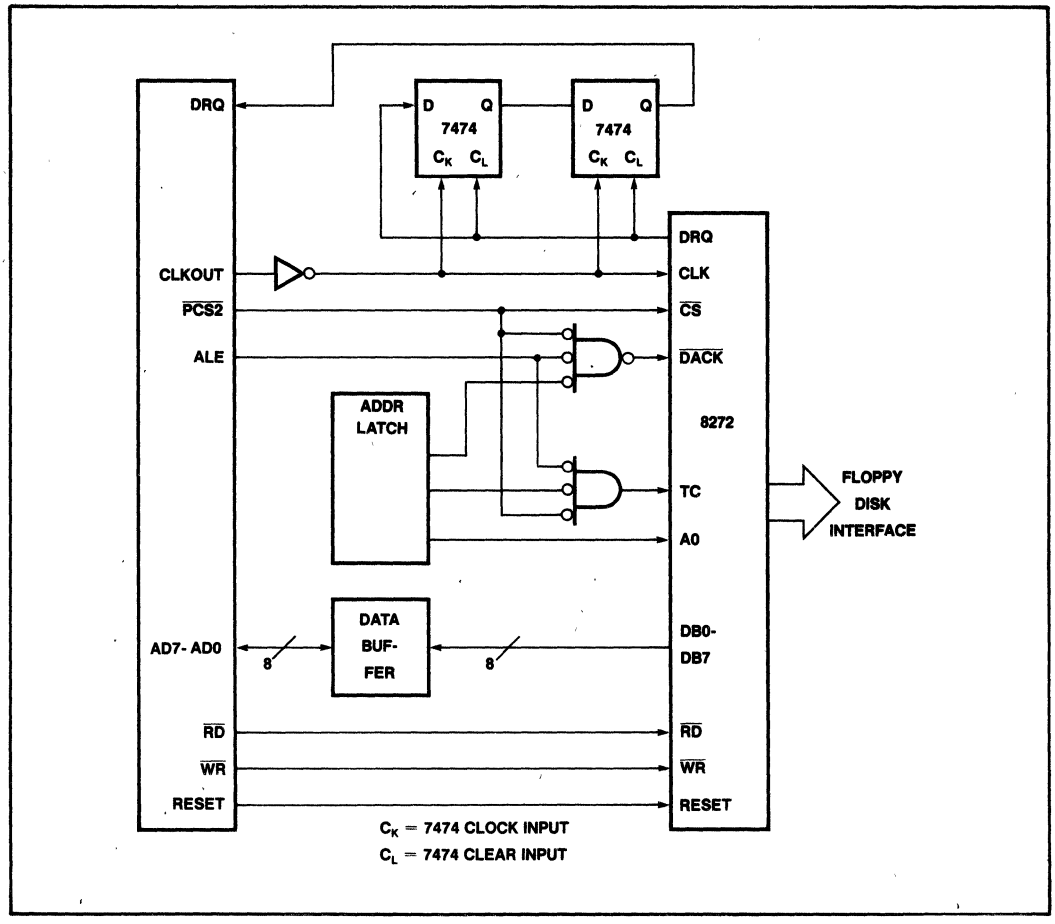


Figure 39. Example 8272/80186 DMA Interface

put a pulse to the 8272 after a certain number of DMA cycles have been run (see next section for 80186 timer information).

The above discussion assumed that a single 80186 $\overline{PCS}$ line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated $\overline{PCS}$ lines could be used for each function. For example, $\overline{PCS2}$ could be used to select the 8272, $\overline{PCS3}$ could be used to drive the DACK line of the 8272, etc.

DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{RD}$ going active) spec of 800ns min. This requires 6.4 80186 CPU clock cycles (at 8 MHz), well beyond the 5 minimum provided by the 80186 (4 clock cycles to the beginning of the DMA bus cycle, 5 to the beginning of T_2 of the DMA bus cycle where $\overline{RD}$ will go active). The two flip-flops add two complete CPU clock cycles to this response time.

DMA request will go away 200ns after DACK is presented to the 8272. During a DMA write cycle (i.e., a destination synchronized transfer), this is not soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 40. The 8274 interface is even simpler than the 8272 interface, since it does not require the generation of a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.

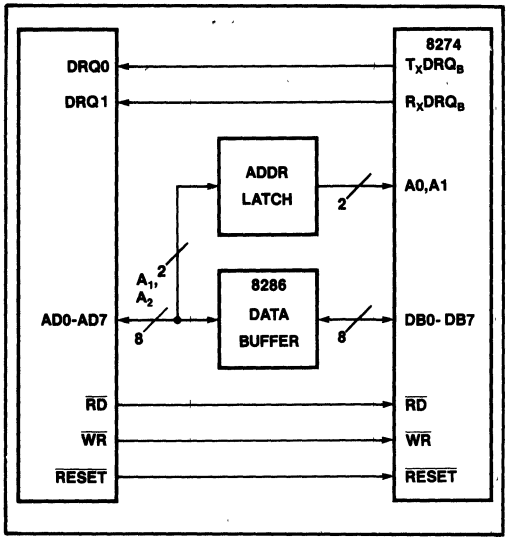


Figure 40. Example 8274/80186 DMA Interface

The data lines of the 8274 are connected through buffers to the 80186 AD0-AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 will not float its drivers before the 80186 will begin driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read from or write to the data register of the 8274 after the 8274 generates the DMA request signal will clear the DMA request. The time between when the control signal ($\overline{RD}$ or $\overline{WR}$) becomes active and when the 8274 will drop its DMA request during a DMA write is 150ns, which will require at least one wait state be inserted into the DMA write cycle for proper operation of the interface.

5. TIMER UNIT INTERFACING

The 80186 includes a timer unit which provides three independent 16-bit timers. These timers operate independently of the CPU. Two of these have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third timer can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 could be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. If the CPU modifies one of the timer registers, this change will affect the counter element the next time that register is presented to the counter element. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Registers

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter element from accessing the timer registers.

Each timer has a 16-bit count register. This register is incremented for each timer event. A timer event can be a low-to-high transition on the external pin (for timers 0 and 1), a CPU clock transition (divided by 4 because of the counter element multiplexing), or a time out of timer 2 (for timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted by a single timer/counter. This register can be both read or written whether the timer is or is not operating.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register will be reset to zero, that is, the maximum count value will never be stored in the count register. This maximum count value may be written while the timer is operating. A maximum count

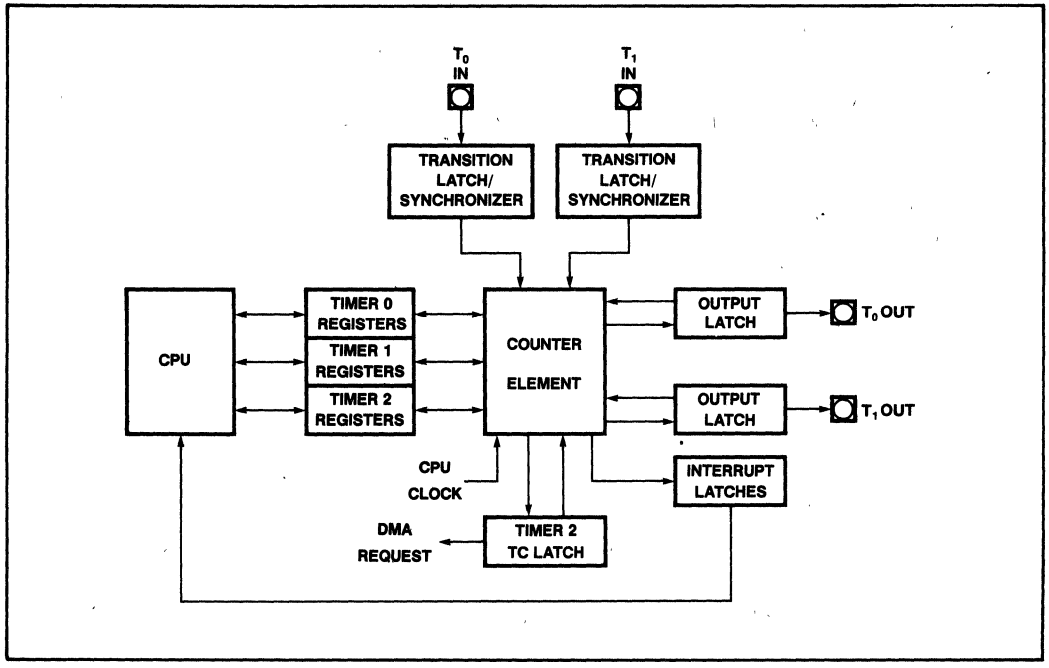


Figure 41. 80186 Timer Model

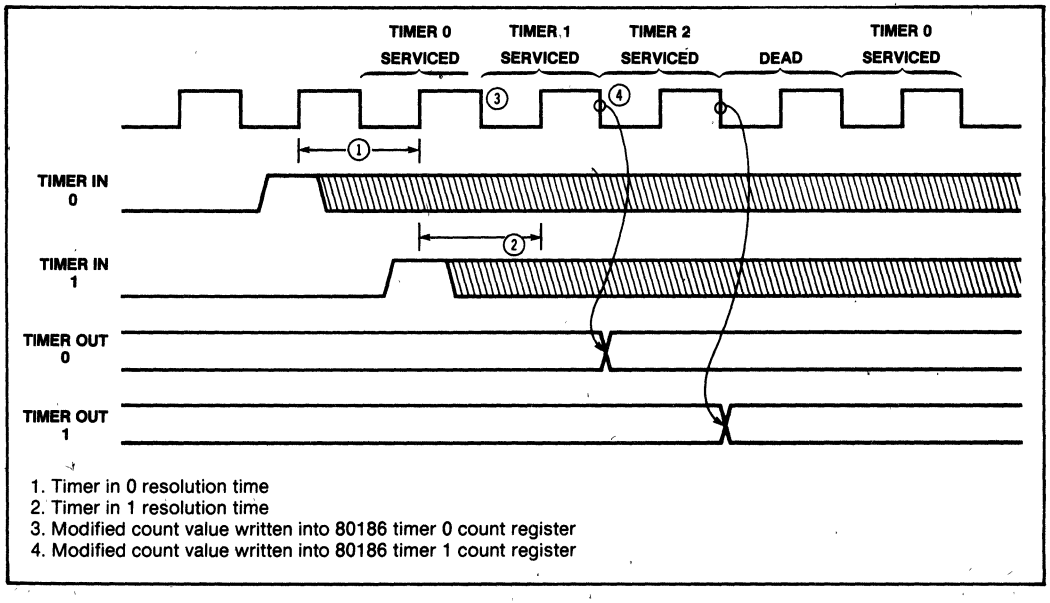


Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

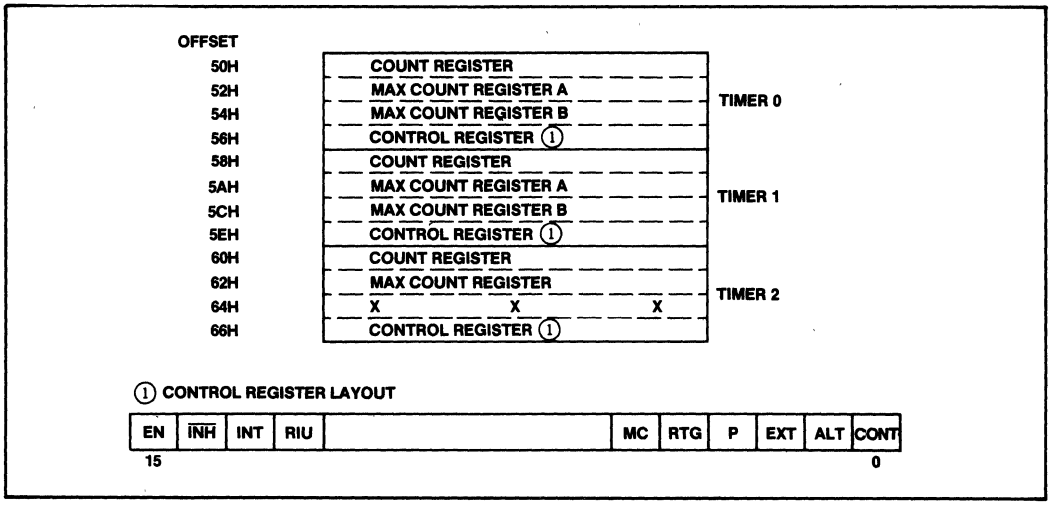


Figure 43. 80186 Timer Register Layout

value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. The user should be aware that only equivalence between the count value and the maximum count register value is checked, that is, the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This could only occur by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If this is programmed, the timer will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTernate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both maximum count register A and maximum count register B are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. This RIU bit is read only. It is unaffected by any write to the timer control register. It will always be read 0 in single maximum count register mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. That is, an interrupt can be generated whenever the value in maximum count register A is reached, and whenever the value in maximum count register B is reached. In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required to clear this bit. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. This bit is used to enable the timer to count. The timer will count timer events only when this bit is set. Any timer events occurring when this bit is reset are ignored. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The timer ENable bit will not be modified by a write to the timer control register if the INHibit bit is not set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a 1.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a

timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit in the timer control register will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register A, reset to zero, ad infinitum. In dual maximum count register mode, the timer will count up the the value in maximum count register A, reset to zero, count up the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts timer events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For timer 2, this is the only timer event which can be used. For timers 0 and 1, this event is selected by clearing the EXTernal and Prescaler bits in the timer control register.

Timers 0 and 1 can use timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTernal bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever timer 2 resets to zero having reached its own maximum count. Note that timer 2 must be initialized and running for the other timer's value to be incremented.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, and may, therefore, be asynchronous (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer to increment. Any transition on this line is latched. If a transition occurs when a timer is not being serviced by the counter element, the transition on the input line will be remembered so that when the timer does get serviced, the input transition will be counted. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate (2 MHz with an 8 MHz CPU clock).

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTernal and RTG (retrigger) bits in the timer control register. If the EXTernal bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTernal bit is cleared, all timer increments are caused by either the CPU clock or by timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

If the EXTernal bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

If the EXTernal bit is low and the RTG bit is high, the timer will act as a digital one-shot. In this mode, every low-to-high transition on the timer input pin will cause the timer to reset to zero. If the timer is enabled (i.e., the ENable bit in the timer control register is set) timer operation will begin (the timer will count CPU clock transitions or timer 2 timeouts). Timer operation will cease at the end of a timer cycle, that is, when the value in the maximum count register A is reached and the timer count value resets to zero (in single maximum count register mode, remember that the maximum count value is never stored in the timer count register) or when the value in maximum count register B is reached and the timer count value resets to zero (in dual maximum count register mode). If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timing cycle again regardless of the state of the CONTinuous bit in the timer control register the RIU bit will not be changed by the input transition. If the CONTinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared at the end of the timer cycle. This means that any additional transitions on the input pin will be ignored by the timer. If the CONTinuous bit in the timer control register is set, the timer will reset to zero and begin another timing cycle for every low-to-high transition on the input pin, regardless of whether the timer had reached the end of a timer cycle, because the timer ENable bit would not have been cleared at the end of the timing cycle. The timer will also continue counting at the end of a timer cycle, whether or not another transition has occurred on the input pin. An example use of the timer in this mode is an alarm clock time out signal or interrupt.

5.5 Timer Output Pin Operation

Timers 0 and 1 each contain a single timer output pin. This pin can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer out pin will vary depending on when the input transition occurs relative to the timer's being serviced by the counter element.

When the timer is in single maximum count register mode (the ALTernate bit in the timer control register is cleared) the timer output pin will go low for a single CPU clock the clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as a baud rate generator.

When the timer is programmed in dual maximum count register mode (the ALTernate bit in the timer control register is set), the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used for the current count, high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTinuous bit in the timer control register is set), this pin could generate a waveform of any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

5.6 Sample 80186 Timer Applications

The 80186 timers can be used for almost any application for which a discrete timer circuit would be used. These include real time clocks, baud rate generators, or event counters.

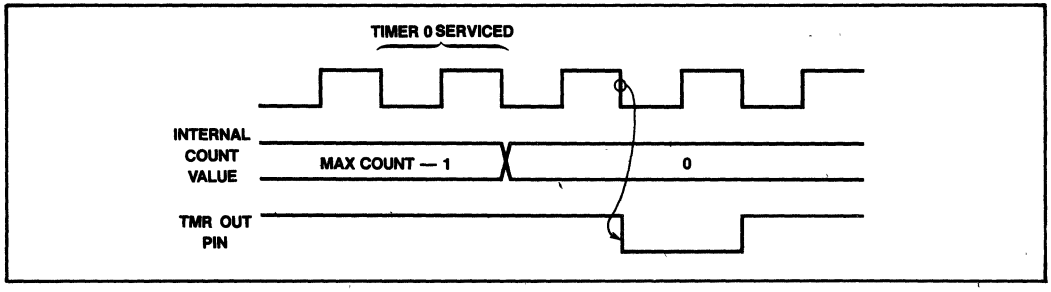


Figure 44. 80186 Timer Out Signal

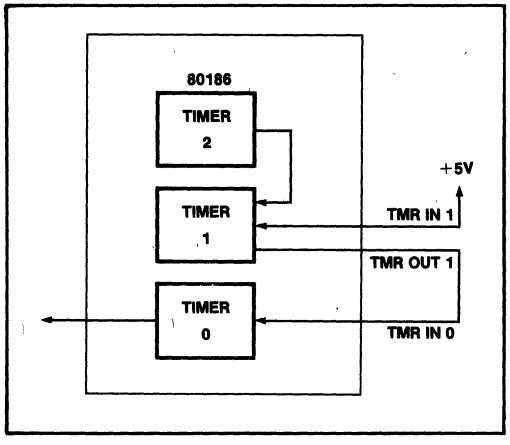


Figure 45. 80186 Real Time Clock

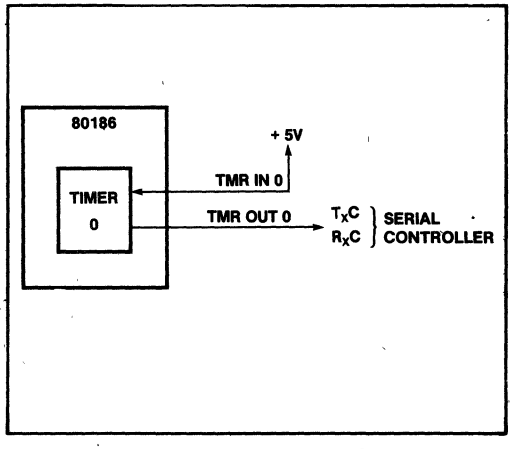


Figure 46. 80186 Baud Rate Generator

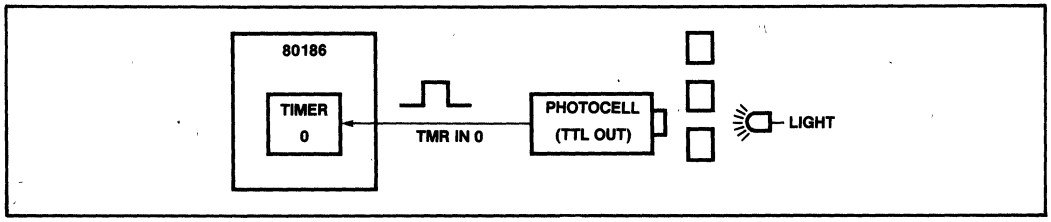


Figure 47.

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the code to program the timer as a baud rate generator is included in appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in appendix D.

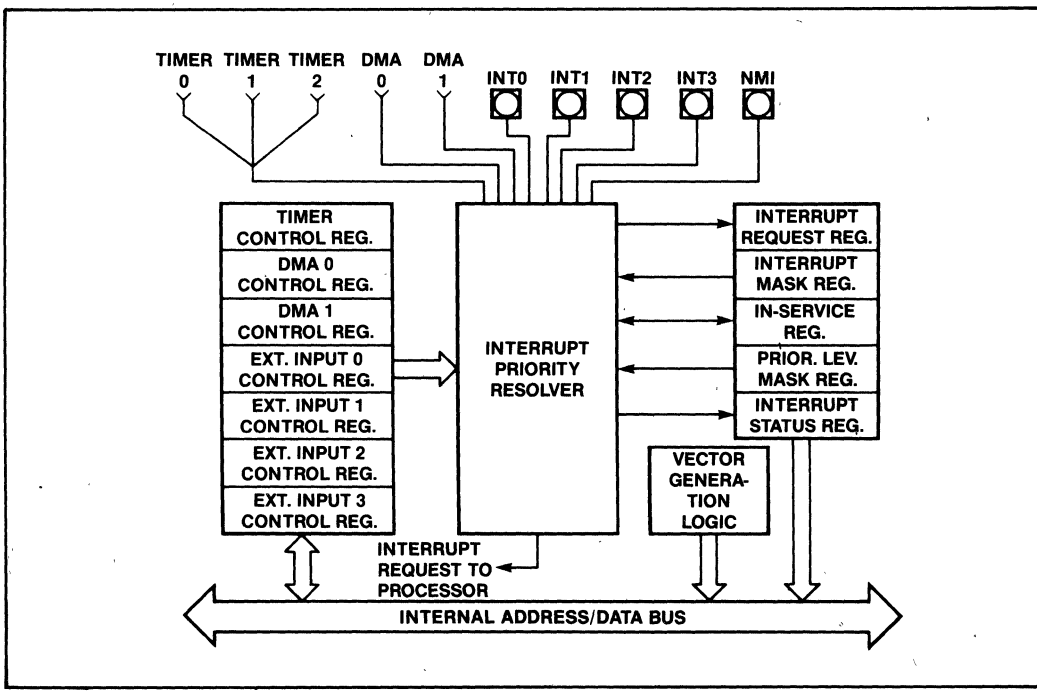


Figure 48. 80186 Interrupt Controller Block Diagram

6. 80186 INTERRUPT CONTROLLER INTERFACING

The 80186 contains an integrated interrupt controller. This unit performs tasks of the interrupt controller in a typical system. These include synchronization of interrupt requests, priortization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external interrupt controller to allow compatibility with the iRMX 86 operating system, and the 80130/80150 operating system firmware chips.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the programmed mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, non-iRMX 86 mode (referred to henceforth as **master** mode), and **iRMX 86** mode. In master mode the integrated controller acts as the master interrupt controller for the system, while in iRMX 86 mode the controller operates as a slave to an external interrupt controller which operates as the master interrupt controller for the system. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes, but the basic charter and function of the interrupt controller remains fundamentally the same. The difference is when in master mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in iRMX 86 mode the interrupt controller presents its interrupt input to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in iRMX 86 mode is done by setting the iRMX mode bit in the peripheral control block pointer (see appendix A).

6.3 Interrupt Controller Registers

The interrupt controller has a number of registers which are used to control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller (master and iRMX 86 mode). The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two basic modes of operation of the interrupt controller. The method of interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers contain

MASTER MODE	OFFSET ADDRESS	iRMX86™ Mode
INT3 CONTROL REGISTER	3EH	(1)
INT2 CONTROL REGISTER	3CH	(1)
INT1 CONTROL REGISTER	3AH	TIMER 2 CONTROL REGISTER
INT0 CONTROL REGISTER	38H	TIMER 1 CONTROL REGISTER
DMA1 CONTROL REGISTER	36H	DMA1 CONTROL REGISTER
DMA0 CONTROL REGISTER	34H	DMA0 CONTROL REGISTER
TIMER CONTROL REGISTER	32H	TIMER 0 CONTROL REGISTER
INTERRUPT CONTROLLER STATUS REGISTER	30H	INTERRUPT CONTROLLER STATUS REGISTER
INTERRUPT REQUEST REGISTER	2EH	INTERRUPT REQUEST REGISTER
IN-SERVICE REGISTER	2CH	IN SERVICE REGISTER
PRIORITY MASK REGISTER	2AH	PRIORITY MASK REGISTER
MASK REGISTER	28H	MASK REGISTER
POLL STATUS REGISTER	26H	(1)
POLL REGISTER	24H	(1)
EOI REGISTER	22H	SPECIFIC EOI REGISTER
(1)	20H	INTERRUPT VECTOR REGISTER

1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers

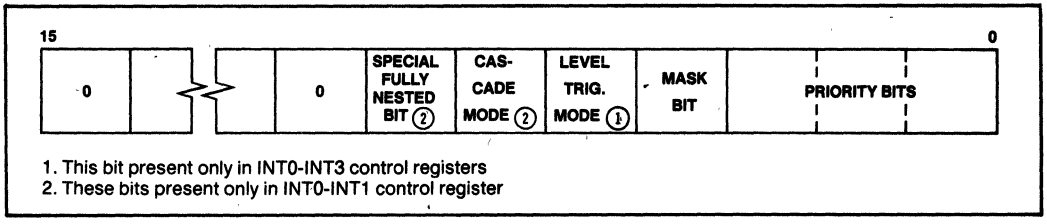


Figure 50. Interrupt Controller Control Register

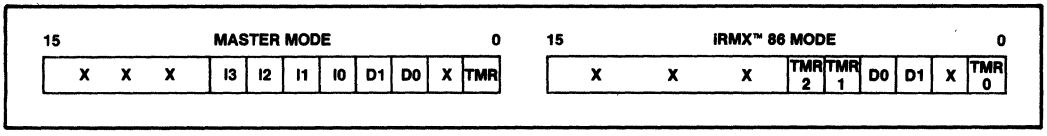


Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

three bits which select one of eight different interrupt priority levels for the interrupt device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is low, the interrupt is enabled, when it is high, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In master mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In iRMX 86 mode, the external interrupt inputs are not used, so each timer can have its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for each interrupt control register. Whenever an interrupt request is made by the interrupt source associated with a specific control register, the bit in interrupt request register is set, regardless if the interrupt is enabled, or if it is of sufficient priority to cause a processor interrupt. The bits in this register which are associated with integrated peripheral devices (the DMA and timer units) can be read or written, while the bits in this register which are associated with the external interrupt pins can only be read (values written to them are not stored). These interrupt request bits are automatically cleared when the interrupt is acknowledged.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller contains a mask register (see Figure 51). This register contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is identically the same bit as is provided in the interrupt control register: modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt controller also contains a priority mask register (see Figure 52). This register contains three bits which indicate the priority of the current interrupt being serviced. When an interrupt is acknowledged (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register, see below), these bits are set to the priority of the device whose interrupt is being acknowledged (which will never be lower than the previous priority programmed into these bits). They prevent any interrupt of lower priority (as set by the priority bits in the interrupt control registers for interrupt sources) from interrupting the processor. These bits are automatically set to the priority of the next lowest interrupt when the End Of Interrupt is issued by the CPU to the interrupt controller (or all 1's if there is no interrupt pending, meaning that interrupts of all priority levels are enabled). This register may be read or written.

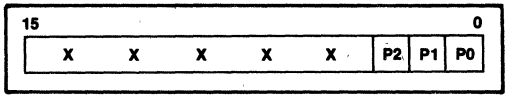


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the con-

trol register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

6.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). Both of these registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If (and only if) an interrupt is pending, they also contain information as to the interrupt type of the highest priority interrupt pending.

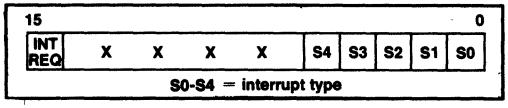


Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. Only the interrupt request, in-service and priority mask registers in the interrupt controller are set appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers. These registers are read only; data written to them is not stored. These registers are not supported in iRMX 86 mode. The state of the bits in these registers in iRMX 86 mode is not defined. However, accessing the poll register location when in iRMX 86 mode will cause the interrupt controller to "acknowledge" the interrupt (i.e., the in-service bit and priority level mask register bits will be set).

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt to the controller by writing to this register. After receiving the End Of Interrupt, the interrupt controller automatically resets the in-service bit for the interrupt and the priority mask register bits. The value of the word written to this register determines whether the End Of Interrupt is specific or non-specific. A non-specific End Of Interrupt is specified by setting the non-specific bit in the word written to the End Of Interrupt register. In a non-specific End Of Interrupt, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific End Of Interrupt allows the in-service bit cleared to be explicitly specified. The in-service bit is reset whether the bit was set by an interrupt acknowledge or if it was set by the CPU writing the

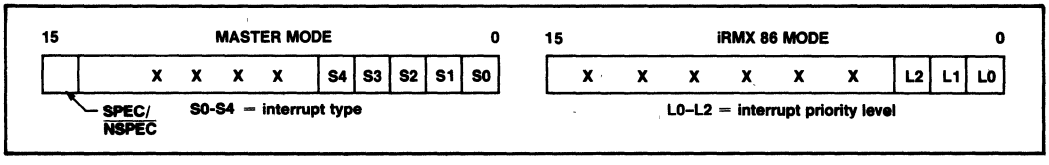


Figure 54. 80186 End of Interrupt Register Format

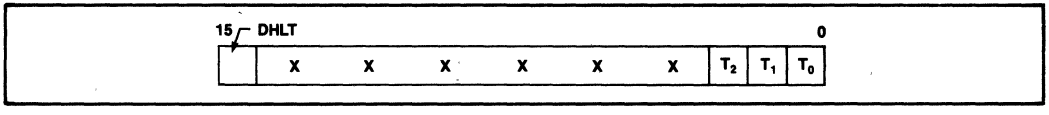


Figure 55. 80186 Interrupt Status Register Format

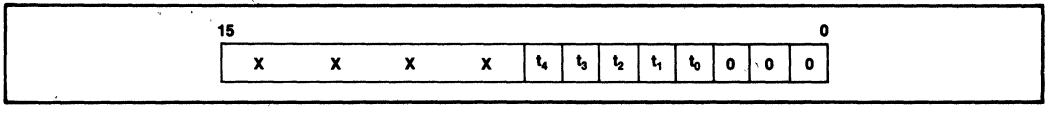


Figure 56. 80186 Interrupt Vector Register Format (iRMX 86 mode only)

bit directly to the in-service register. If the highest priority interrupt is reset, the priority mask register bits will change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority mask register bits will not be modified (because the highest priority interrupt being serviced has not changed). Only the specific EOI is supported in iRMX 86 mode. This register is write only: data written is not stored and cannot be read back.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains four significant bits. There are three bits used to show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit in the interrupt status register is the DMA halt bit. When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set explicitly by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All significant bits in this register are read/write.

6.3.8 INTERRUPT VECTOR REGISTER

Finally, in iRMX 86 mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register is used to specify the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in iRMX 86 mode).

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Each interrupt source may be programmed to be a different priority level in the interrupt controller. An interrupt request generation flow chart is shown in Figure 57. Such a flowchart would be followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller, so that if the condition causing the interrupt is cleared in the originating integrated peripheral device, the interrupt request will remain pending in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register of the interrupt controller. For all internal interrupts, the latched interrupt request can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request in this mode. Each timer has a unique interrupt vector (see section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level as referenced to all other interrupt devices. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lowest priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in master mode. In this mode, the external pins associated with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition each of these control registers contains a bit which selects either edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is "re-armed." In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt request (and also the bit in the interrupt request register) will also go inactive (low). Also, if the interrupt input is in edge triggered mode, a low-to-high transition on the input pin must occur before the interrupt request bit will be set in the interrupt request register.

If the C (Cascade) bit of the INT0 or INT1 control registers are set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the

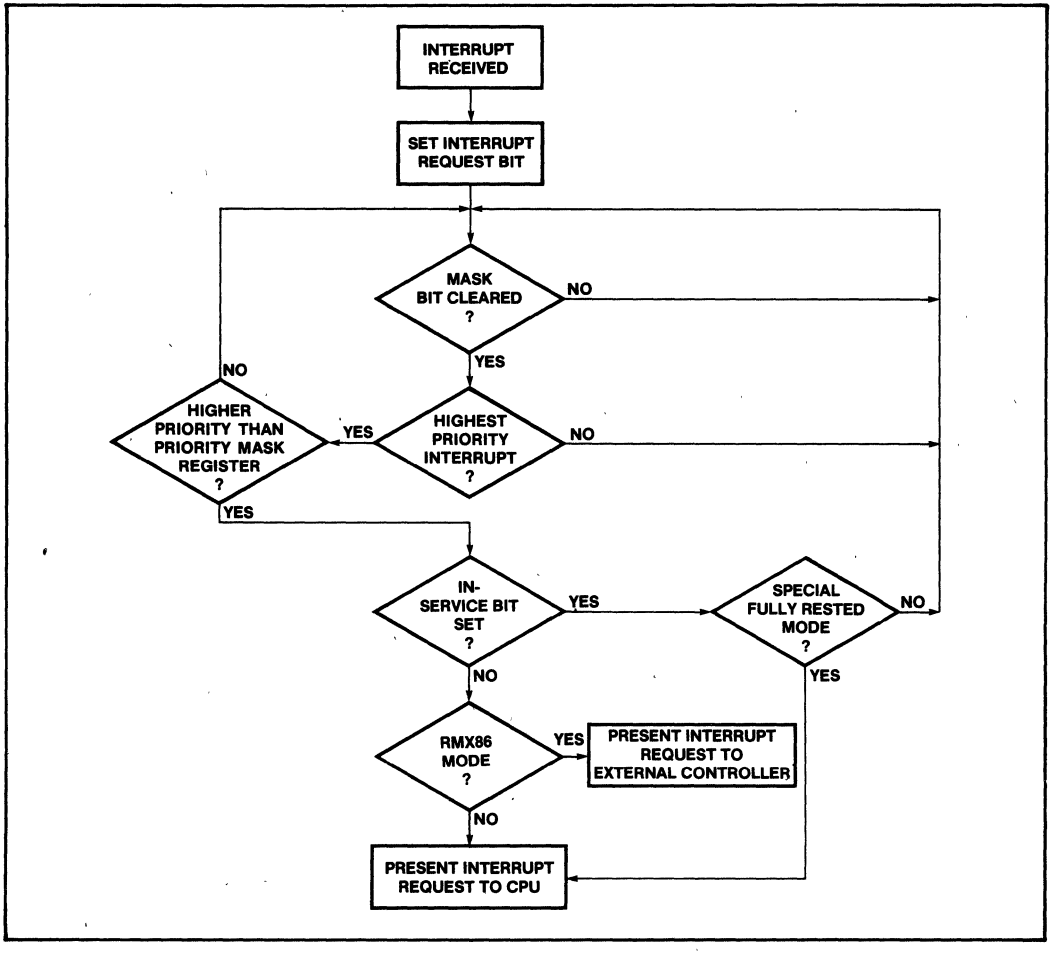


Figure 57. 80186 Interrupt Request Sequencing

interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into cascade mode. This allows 128 individually vectored interrupt sources if two banks of 9 external interrupt controllers each are used.

6.4.3 iRMX™ 86 MODE INTERRUPT SOURCES

When the interrupt controller is configured in iRMX 86 mode, the integrated interrupt controller accepts interrupt requests only from the integrated peripherals. Any external interrupt requests must go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line on the 80186, In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in iRMX 86 mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see section 6.7.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is pro-

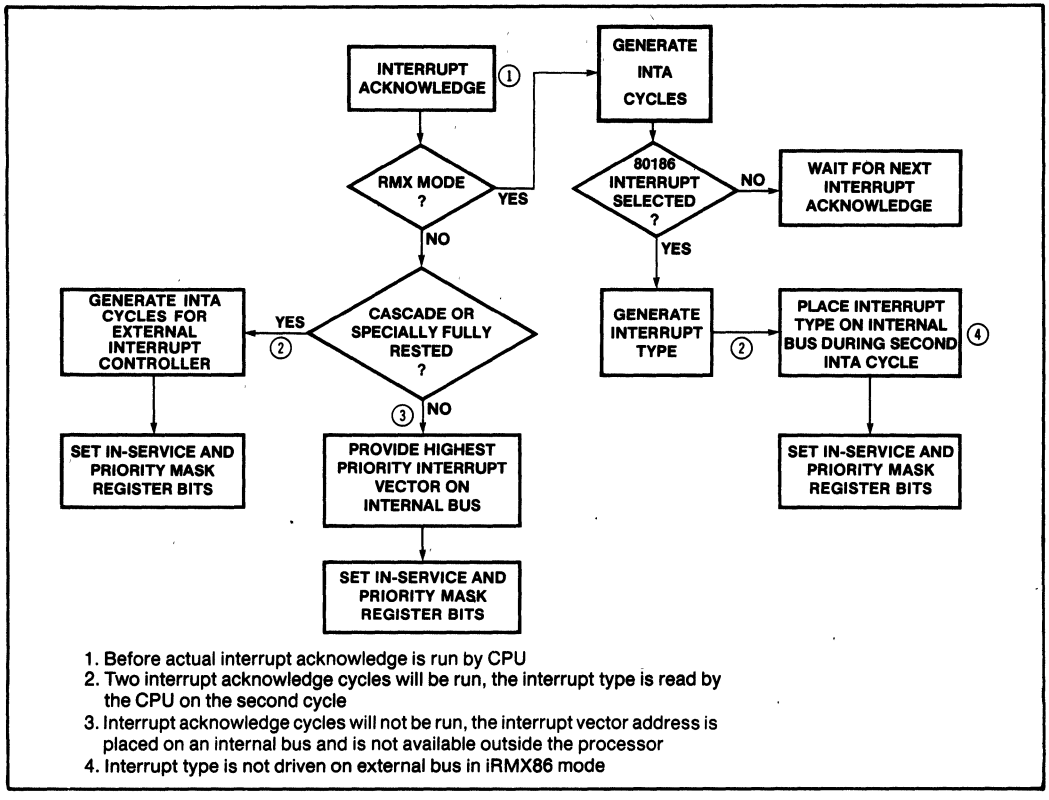


Figure 58. 80186 Interrupt Acknowledge Sequencing

viding the interrupt vector information with the controller in master mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in iRMX 86 mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and the priority mask bits and reset the interrupt request bit in the integrated controller. The priority mask bits will prevent the controller from generating any further interrupts to the CPU from sources of lower priority until the higher priority interrupt service routine has run. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In master mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4. This speeds interrupt response.

In master mode, the integrated interrupt controller is the master interrupt controller of the system. As a result, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in low memory.

Table 5. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Default Priority
timer 0	8	0a
timer 1	18	0b
timer 2	19	0c
DMA 0	10	2
DMA 1	11	3
INT 0	12	4
INT 1	13	5
INT 2	14	6
INT 3	15	7

Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need be be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster then the interrupt response when external vectoring is required, or if the interrupt controller is run in iRMX 86 mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in table 5) is used.

6.5.2 INTERNAL VECTORING, iRMX™ 86 MODE

In iRMX 86 mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In iRMX 86 mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will be allowed to place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowledge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186 samples the

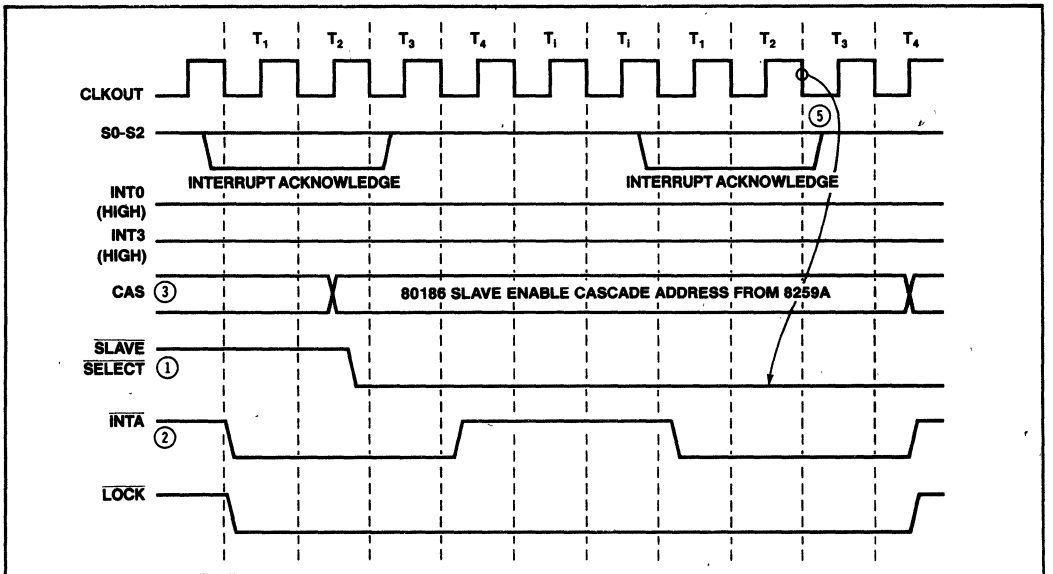


Figure 59. 80186 iRMX-86 Mode Interrupt Acknowledge Timing

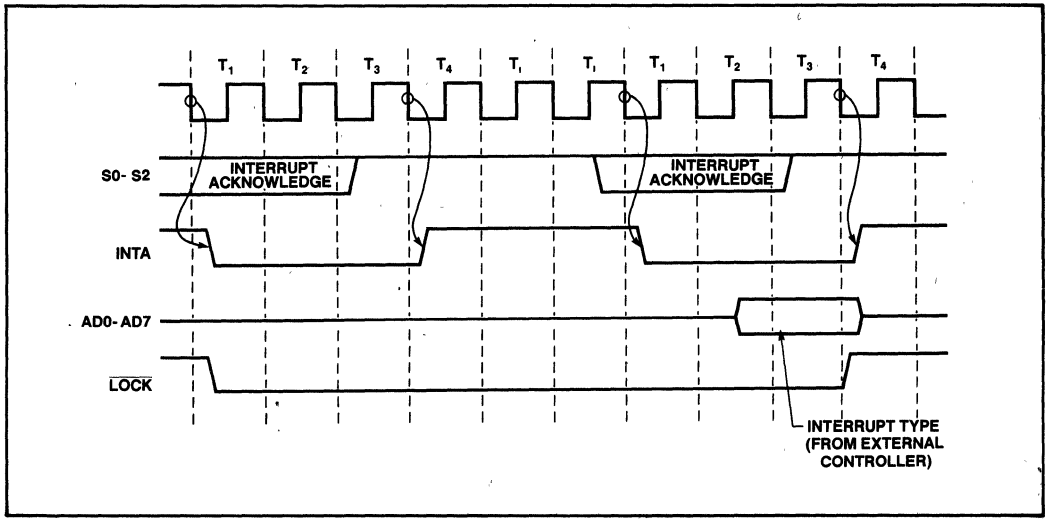


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

SLAVE SELECT line during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable 20ns before and 10ns after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the LOCK output active (meaning that DMA requests and HOLD requests will not be honored until both cycles have been run). Note that the two interrupt acknowledge cycles will always be separated by two idle T states, and that wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with the timing requirements of an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in iRMX 86 mode, even for internally generated vectors, and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time here is the same as if an externally vectored interrupt was required, namely 55 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in cascade mode, special fully nested mode, or iRMX 86 mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 60). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines (S0-S2) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.6 Interrupt Controller External Connections

The four interrupt signals can be programmably configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or iRMX 86 mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

6.6.1 DIRECT INPUT MODE

When the Cascade mode bits are cleared, the interrupt input lines are configured as direct interrupt input lines (see Figure 61). In this mode an interrupt source (e.g., an 8272 floppy disk controller) may be directly connected to the interrupt input line. Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any interrupt acknowledge cycles.

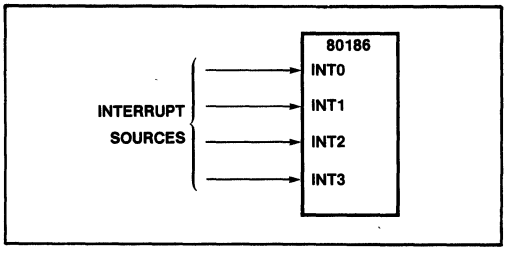


Figure 61. 80186 Non-Cascaded Interrupt Connection

These lines can be individually programmed in either edge or level triggered mode using their respective control registers. In edge triggered mode, a low-to-high transition must occur before the interrupt will be generated to the CPU, while in level triggered mode, only a high level must be present on the input for an interrupt to be generated. In edge trigger mode, the interrupt input must also be low for at least 1 CPU clock cycle to insure recognition. In both modes, the interrupt input must remain active until acknowledged.

6.6.2 CASCADE MODE

When the Cascade mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in cascade mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/$\overline{\text{INTA0}}$ and INT3/$\overline{\text{INTA1}}$ lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/$\overline{\text{INTA0}}$ provides the interrupt acknowledge for an INT0 input, and INT3/$\overline{\text{INTA1}}$ provides the interrupt acknowledge for an INT1 input. Figure 62 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/$\overline{\text{INTA0}}$ line, and will also be reflected by interrupt acknowledge status being generated on the $\overline{\text{S0}}$-$\overline{\text{S2}}$ status lines. On the second pulse, the interrupt type will be read in. The 80186 externally vectored interrupt response is covered in more detail in section 6.5.

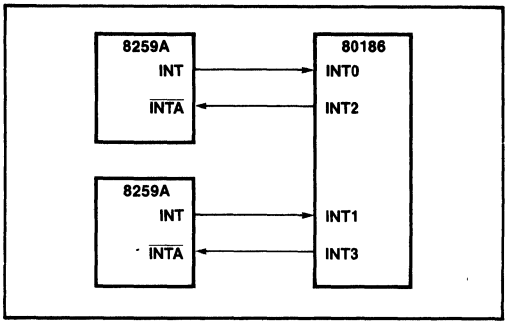


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

INT0/INT2/$\overline{\text{INTA0}}$ and INT1/INT3/$\overline{\text{INTA1}}$ may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/$\overline{\text{INTA0}}$ may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/$\overline{\text{INTA1}}$ each provide separate internally vectored interrupt inputs.

When an interrupt is received on a cascaded interrupt, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's mask and priority mask registers. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186 interrupt request line, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a special fully nested

mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike cascade mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM mode bit is set and the Cascade mode bit is not also set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt.

6.6.4 iRMX™ 86 MODE

When the RMX bit in the peripheral relocation register is set, the interrupt controller is set into iRMX 86 mode. In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 63 shows the hardware configuration of the 80186 interrupt lines with an external controller in iRMX 86 mode.

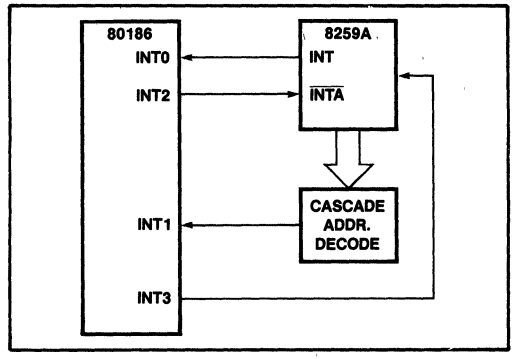


Figure 63. 80186 iRMX86 Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

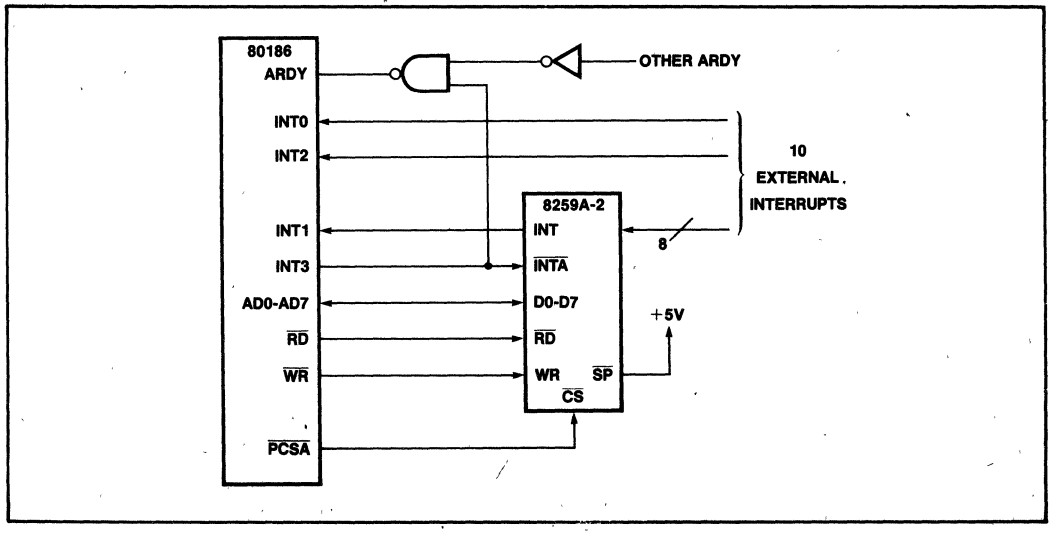


Figure 64. 80186/8259A Interrupt Cascading

6.7 Example 8259A/Cascade Mode Interface

Figure 64 shows the 80186 and 8259A in cascade interrupt mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an "interrupt ready" signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in slave mode).

6.8 Example 80130 iRMX™ 86 Mode Interface

Figure 65 shows the 80186 and 80130 connected in iRMX 86 mode. In this mode, the 80130 interrupt controller is the master interrupt controller of the system. The 80186 generates an interrupt request to the 80130 interrupt controller when one of the 80186 integrated peripherals has created an interrupt condition, and that condition is sufficient to generate an interrupt from the 80186 integrated interrupt controller. Note that the 80130 decodes the interrupt acknowledge status directly from the 80186 status lines; thus, the INT2/$\overline{\text{INTA0}}$ line of the 80186 need not be connected to the 80130. Figure 65 uses this interrupt acknowledge signal to enable the cascade address decoder. The 80130 drives the cascade address on AD8-AD10 during T_1 of the second interrupt acknowledge cycle. This cascade address is latched into the system address latches, and if the proper cascade address is decoded by the 8205 decoder, the 80186 INT1/SLAVE SELECT line will be driven active, enabling the 80186 integrated interrupt controller to place its interrupt vector on the internal bus. The code to configure the 80186 into iRMX 86 mode is presented in appendix E.

6.9 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response

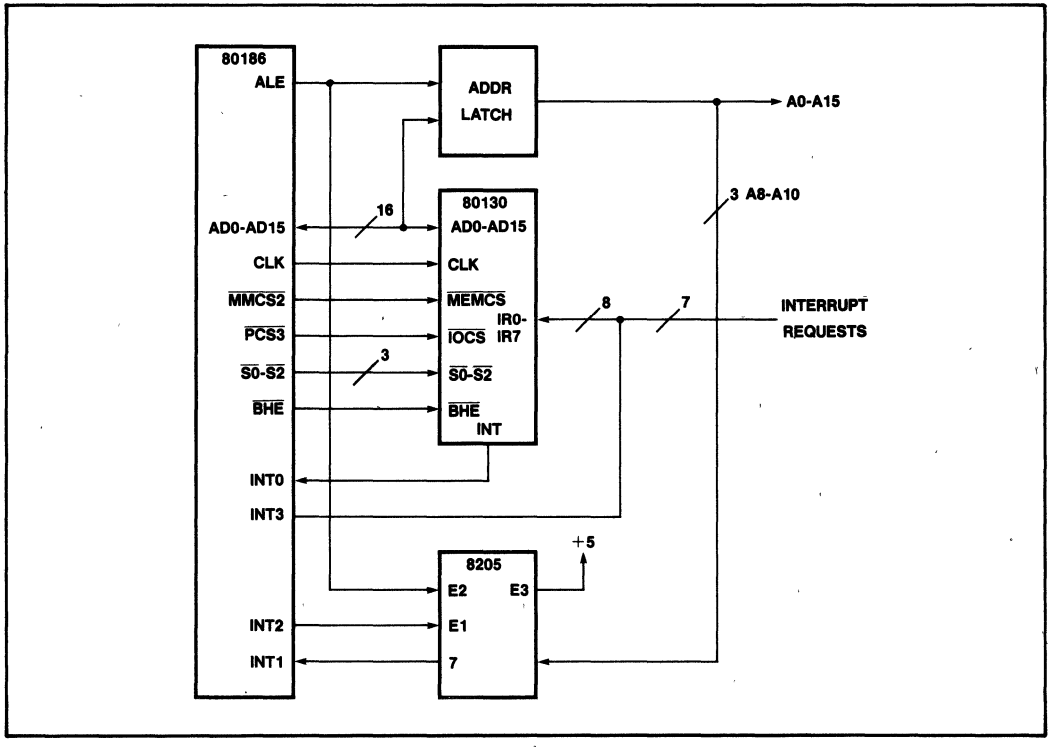


Figure 65. 80186/80130 iRMX86 Mode Interface

time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the instruction being executed and the state of the interrupt enable flip-flop.

Interrupts will be acknowledged only if the interrupt enable flip-flop in the CPU is set. Thus, interrupt latency will be very long indeed if interrupts are never enabled by the processor!

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). Finally, an interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the $\overline{\text{TEST}}$ input is active. If the WAIT sees the $\overline{\text{TEST}}$ input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. This is required, since the WAIT is used to prevent execution by the 80186 of an 8087 instruction while the 8087 is busy.

7. CLOCK GENERATOR

The 80186 includes a clock generator which generates the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

The 80186 crystal oscillator is a parallel resonant, Pierce oscillator. It was designed to be used as shown in Figure 67. The capacitor values shown are approximate. As the crystal frequency drops, they should be increased, so that at the 4 MHz minimum crystal frequency supported by the 80186 they take on a value of 30pF. The output of this oscillator is not directly available outside the 80186.

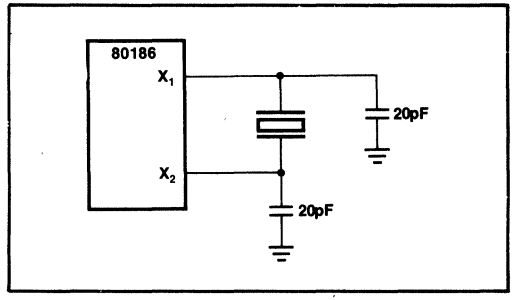


Figure 67. 80186 Crystal Connection

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left open. This oscillator input is used to drive an internal di-

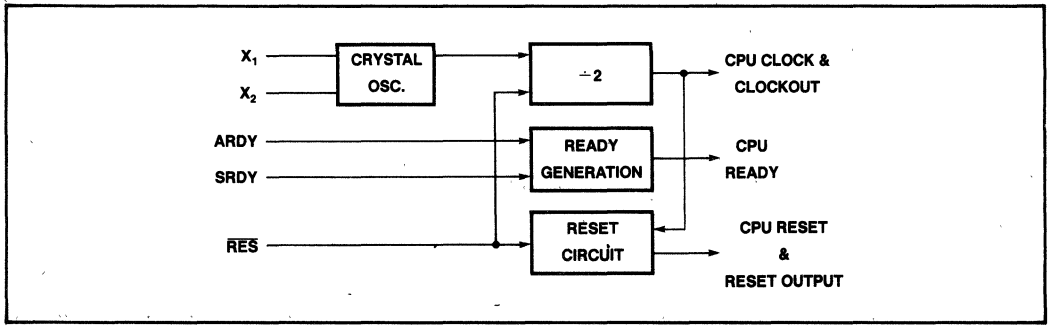


Figure 66. 80186 Clock Generator Block Diagram

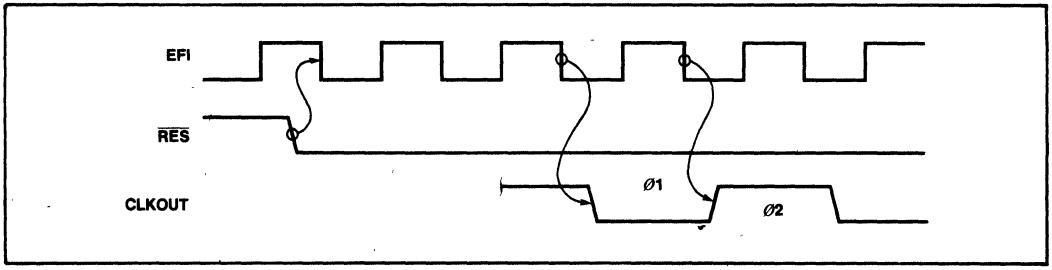


Figure 68. 80186 Clock Generator Reset

vide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input ($\overline{RES}$) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input signal also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the $\overline{RES}$ input signal first goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after $\overline{RES}$ goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next EFI cycle, the $\overline{RES}$ input signal must satisfy a 25ns setup time to the high-to-low EFI input signal (see Figure 68). During this clear, clockout will be high. On the next high-to-low transition of X1, clockout will go low, and will change state on every subsequent high-to-low transition of EFI.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains active as long as the $\overline{RES}$ input also remains active. After the $\overline{RES}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To insure that the RESET output will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{RES}$ input must satisfy certain hold and setup times to the low-to-high edge of the clockout signal of the 80186 (see Figure 69).

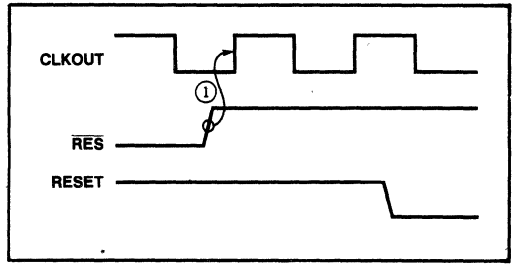


Figure 69. 80186 Coming out of Reset

8. CHIP SELECTS

The 80186 includes a chip select unit which generates hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can be used to fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 70).

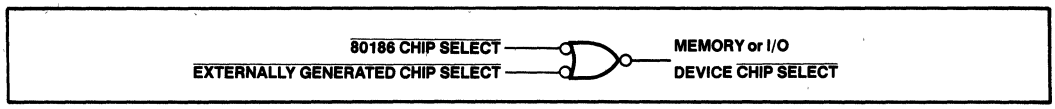


Figure 70. 80186/External Chip Select/Device Chip Select Generation

8.1 Memory Chip Selects

The 80186 provides six discrete chip select lines which are meant to be connected to memory components in an 80186 system. These signals are named UCS, LCS, and MCS0-3 for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Selects 0-3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 71).

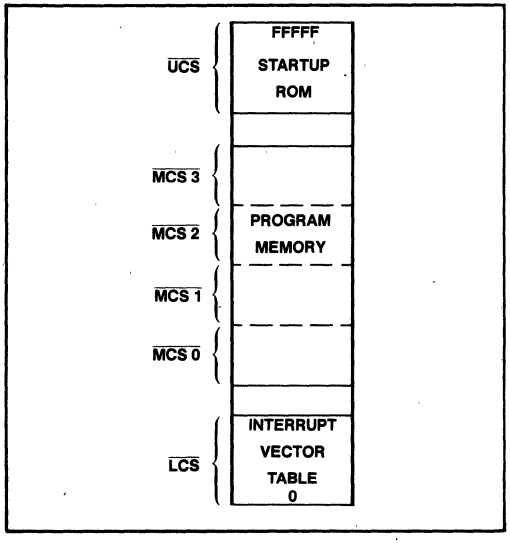


Figure 71. 80186 Memory Areas & Chip Selects

As could be guessed by their names, upper memory, lower memory, and mid-range memory chip selects are designed to address upper, lower, and middle areas of memory in an 80186 system. The upper limit of UCS and the lower limit of LCS are fixed at FFFFFH and 00000H in memory space, respectively. The other limit of these is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 72). These include 1 each for UCS and LCS, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

On reset, only UCS is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see section 8.3 for more information on internal ready generation). All other chip select registers assume indeterminate states after reset, but none of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. These signals are named PCS0-6. Each of these lines is active for one of seven contiguous 128 byte areas in memory or I/O space above a programmed base address.

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 72). These registers allow the base address of the peripherals to be set, and allow the peripherals to be mapped into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows PCS5 and PCS6 to become latched A1 and A2 outputs. When this option is selected, PCS5 and PCS6 will reflect the state of A1 and A2 throughout a bus cycle. These are provided to allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high. They will only reflect A1 and A2 after both PACS and MPCS have been accessed (and are programmed to provide A1 and A2!).

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

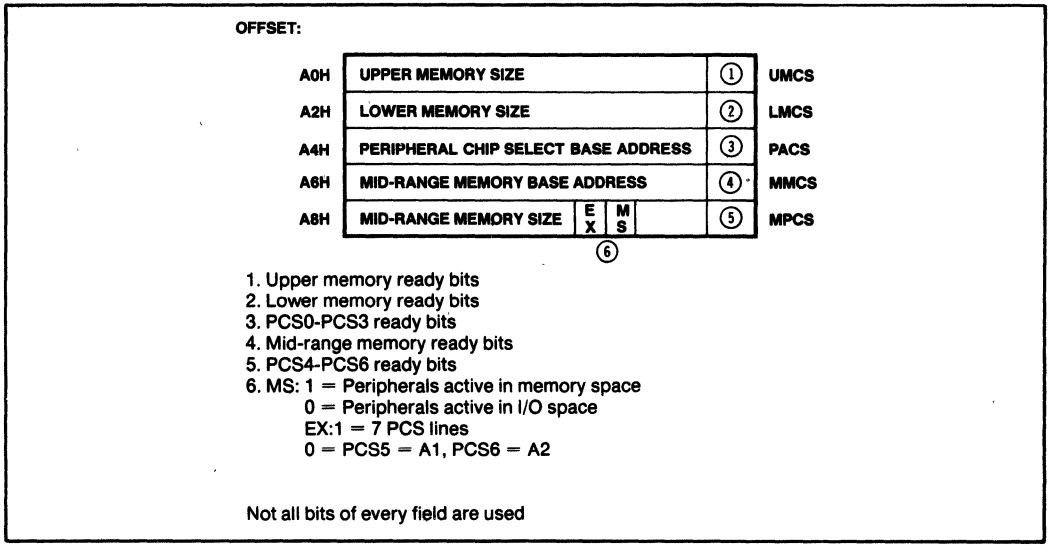


Figure 72. 80186 Chip Select Control Registers

For each ready generation area, 0-3 wait states may be inserted by the internal unit. Table 6 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore the state of the external ready (i.e., only the internal ready circuit will be used) or to factor the state of the external ready (i.e., a ready will be returned to the processor only after both the internal ready circuit has gone ready and the external ready has gone ready). Some kind of circuit must be included to generate an external ready, however, since upon reset the ready generator is programmed to factor external ready to all accesses to the top 1K byte memory block. If a ready was not returned on one of the external ready lines (ARDY or SRDY) the processor would wait forever to fetch its first instruction.

Table 6. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples of the use of the chip select lines are given in the bus interface section of this note (section 3.2). These examples show how simple it is to use the chip select function provided by the 80186. The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186 CPU or DMA units. When another master has the bus, it must generate its own chip select function. In addition, whenever the bus is given by the 80186 to an external master (through the HOLD/ HLDA arrangement) the 80186 does NOT float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any of the locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas *must* be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate.

If any of the chip select areas overlap the integrated 256-byte control register block, the timing on the

chip select line is altered. As always, any values returned on the external bus from this access are ignored.

9. SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

9.1 System Initialization in an 80186 System

Most programmable components of a computer system must be initialized before they are used. This is also true for the 80186. The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals are used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs, the microcomputer system will cease to operate (since the processor will fetch garbage from the data bus). Appendix F shows a typical initialization sequence for the 80186 chip select unit.

Once the chip selects have been properly initialized, the rest of the 80186 system may be initialized much like an 8086 system. For example, the interrupt vector table might get set up, the interrupt controller initialized, a serial I/O channel initialized, and the main program begun. Note that the integrated peripherals included in the 80186 do not share the same programming model as the standard Intel peripherals used to implement these functions in a typical 8086 system, i.e., different values must be programmed into different registers to achieve the same function using the integrated peripherals. Appendix F shows a typical initialization sequence for an interrupt driven system using the 80186 interrupt controller.

9.2 Initialization for iRMX™ 86 System

Using the iRMX 86 operating system with the 80186 requires an external 8259A and an external 8253/4 or alternatively an external 80130 OSF component. These are required because the operating system is interrupt driven, and expects the interrupt controller and timers to have the register model of these external devices. This model is not the same as is implemented by the 80186. Because of this, the 80186 interrupt controller must be placed in iRMX 86 mode after reset. This initialization can be done at any time after reset before jump to the root task of iRMX 86 System is actually performed. If need be, a small section of code which initializes both the 80186 chip selects and the 80186 interrupt controller can be inserted between the reset vector location and the beginning of iRMX 86 System (see Figure 73). In this case, upon reset, the processor would jump to the 80186 initialization code, and when this has been completed, would jump to the iRMX 86 initialization code (in the root task). It is important that the 80186 hardware be initialized before iRMX 86 operation is begun, since some of the resources addressed by the 80186 system may not be initialized properly by iRMX 86 System if the initialization is done in the reverse manner.

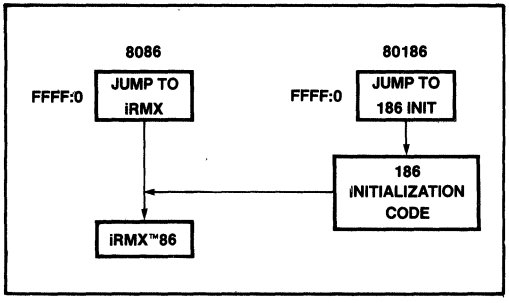


Figure 73. iRMX-86 Initialization with 8086 & 80186

9.3 Instruction Execution Differences Between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

Undefined Opcodes:

When the opcodes 63H,64H,65H,66H,67H,F1H, FEH XX111XXXB and FFH XX111XXXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH opcode:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will execute an illegal instruction exception, interrupt type 6.

Word Write at Offset FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset

FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

Shift/Rotate by Value Greater Then 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK prefix:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

Interrupted String Move Instructions:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be re-executed upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

Conditions causing divide error with an integer divide:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater then 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling −32768 and −128 in decimal, respectively).

ESC Opcode:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for co-processors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register!) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in appendix H of this note.

10. CONCLUSIONS

The 80186 is a glittering example of state-of-the art integrated circuit technology applied to make the job of the microprocessor system designer simpler and faster. Because many of the required peripherals and their interfaces have been cast in silicon, and because of the timing and drive latitudes provided by the part, the designer is free to concentrate on other issues of system design. As a result, systems designed around the 80186 allow applications where no other processor has been able to provide the necessary performance at a comparable size or cost.

APPENDIX A: PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers fills 256 contiguous bytes and can be located beginning on any 256 byte boundary of the 80186 memory or I/O space. A map of these registers is shown in Figure A-1.

A.1 Setting the Base Location of the Peripheral Control Block

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation register will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into iRMX86 compatibility mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

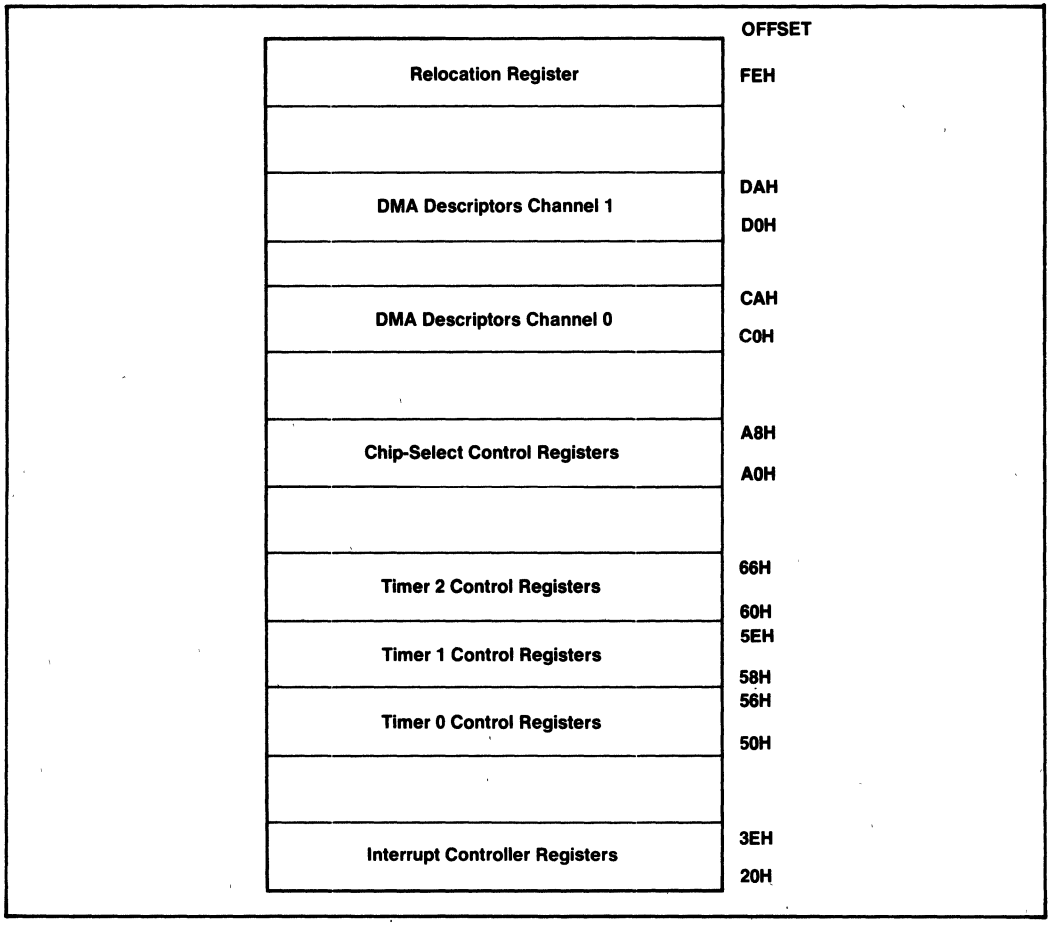


Figure A-1. 80186 Integrated Peripheral Control Block

AFN-210973

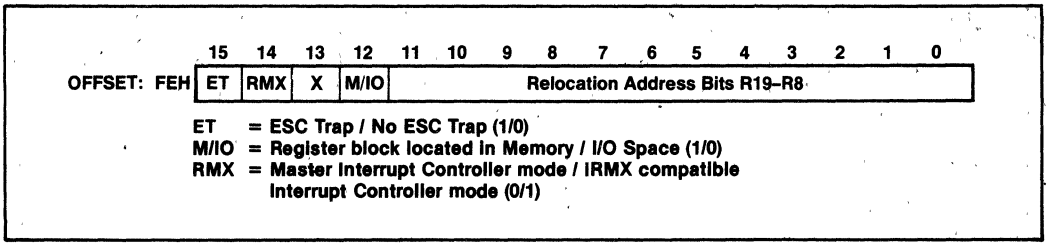


Figure A-2. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEH in I/O space.

If the user wished to locate the peripheral control block starting at memory location 10000H he would program the peripheral control register with the value 1100H. By doing this, he would move all registers within the integrated peripheral control block to memory locations 10000H to 100FFH. Note that since the relocation register is contained within the peripheral control block, it too would move to word location 100FEH in memory space.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the integrated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle, with only the lower 8 bits of data being driven by the write cycle (since the upper 8 bits of data are nonexistant on the external data bus!)

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored whenever an access is made to any location within the integrated peripheral register control block. This ready will also be returned if an access is made to a location within the 256 byte area of the periperal control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. ANY access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block must be WORD accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 AND the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16 bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle with only the lower 8 bits of data being driven on the external data bus (on a write).

APPENDIX B: 80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 Why Synchronizers Are Required

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch, wait a certain amount of time, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have. Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling

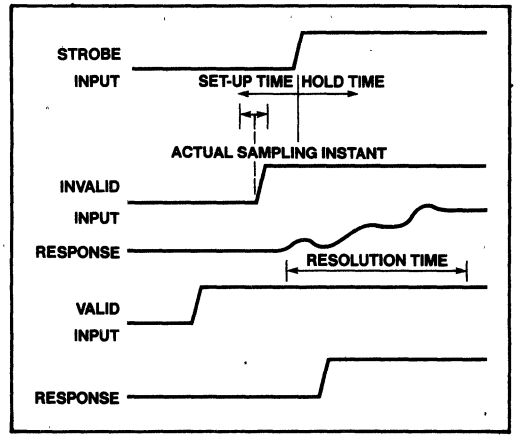


Figure B-1. Valid & Invalid Latch Input Transitions & Responses

window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

B.2 80186 Synchronizers

The 80186 contains synchronizers on the $\overline{\text{RES}}$, $\overline{\text{TEST}}$, TmrIn0-1, DRQ0-1, NMI, INT0-3, ARDY, and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C: 80186 EXAMPLE DMA INTERFACE CODE

```
$mod186
name                                assembly_example_80186_DMA_support
;
;  This file contains an example procedure which initializes the 80186 DMA
;          controller to perform the DMA transfers between the 80186 system the the
;          8272 Floppy Disk Controller (FDC). It assumes that the 80186
;          peripheral control block has not been moved from its reset location.
;
arg1                        equ     word ptr [BP + 4]
arg2                        equ     word ptr [BP + 6]
arg3                        equ     word ptr [BP + 8]
DMA_FROM_LOWER              equ     0FFC0h              ;  DMA register locations
DMA_FROM_UPPER              equ     0FFC2h
DMA_TO_LOWER                equ     0FFC4h
DMA_TO_UPPER                equ     0FFC6h
DMA_COUNT                   equ     0FFC8h
DMA_CONTROL                 equ     0FFCAh
DMA_TO_DISK_CONTROL         equ     01486h              ;  destination synchronization
                                                        ;  source to memory, incremented
                                                        ;  destination to I/O
                                                        ;  no terminal count
                                                        ;  byte transfers
DMA_FROM_DISK_CONTROL       equ     0A046h              ;  source synchronization
                                                        ;  source to I/O
                                                        ;  destination to memory, incr
                                                        ;  no terminal count
                                                        ;  byte transfers
FDC_DMA                     equ     6B8h                ;  FDC DMA address
FDC_DATA                    equ     688h                ;  FDC data register
FDC_STATUS                  equ     680h                ;  FDC status register

cgroup                      group   code
code                        segment                     public 'code'
                            public  set_dma_
                            assume  cs:cgroup
;
;  set_dma (offset,to) programs the DMA channel to point one side to the
;          disk DMA address, and the other to memory pointed to by ds:offset. If
;          'to' = 0 then will be a transfer from disk to memory; if
;          'to' = 1 then will be a transfer from memory to disk. The parameters to
;          the routine are passed on the stack.
;
set_dma_                    proc    near
                            enter   0,0                 ;  set stack addressability
                            push    AX                  ;  save registers used
                            push    BX
                            push    DX
                            test    arg2,1              ;  check to see direction of
                                                        ;  transfer
                            jz      from_disk
;  performing a transfer from memory to the disk controller
;
                            mov     AX,DS               ;  get the segment value
                            rol     AX,4                ;  gen the upper 4 bits of the
                                                        ;  physical address in the lower 4
                                                        ;  bits of the register
```

```
                        mov     BX,AX                   ; save the result...
                        mov     DX,DMA_FROM_UPPER       ; prgm the upper 4 bits of the
                        out     DX,AX                   ; DMA source register
                        and     AX,0FFF0h               ; form the lower 16 bits of the
                                                        ; physical address
                        add     AX,arg1                 ; add the offset
                        mov     DX,DMA_FROM_LOWER       ; prgm the lower 16 bits of the
                        out     DX,AX                   ; DMA source register
                        jnc     no_carry_from           ; check for carry out of addition
                        inc     BX                      ; if carry out, then need to adj
                        mov     AX,BX                   ; the upper 4 bits of the pointer
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
no_carry_from:
                        mov     AX,FDC_DMA              ; prgm the low 16 bits of the DMA
                        mov     DX,DMA_TO_LOWER         ; destination register
                        out     DX,AX
                        xor     AX,AX                   ; zero the up 4 bits of the DMA
                        mov     DX,DMA_TO_UPPER         ; destination register
                        out     DX,AX
                        mov     AX,DMA_TO_DISK_CONTROL; prgm the DMA ctl reg
                        mov     DX,DMA_CONTROL          ; note: DMA may begin immediatly
                        out     DX,AX                   ; after this word is output
                        pop     DX
                        pop     BX
                        pop     AX
                        leave
                        ret
from_disk:
;
;   performing a transfer from the disk to memory
;
                        mov     AX,DS
                        rol     AX,4
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
                        mov     BX,AX
                        and     AX,0FFF0h
                        add     AX,arg1
                        mov     DX,DMA_TO_LOWER
                        out     DX,AX
                        jnc     no_carry_to
                        inc     BX
                        mov     AX,BX
                        mov     DX,DMA_TO_UPPER
                        out     DX,AX
no_carry_to:
                        mov     AX,FDC_DMA

                        mov     DX,DMA_FROM_LOWER
                        out     DX,AX
                        xor     AX,AX
                        mov     DX,DMA_FROM_UPPER
                        out     DX,AX
                        mov     AX,DMA_FROM_DISK_CONTROL
                        mov     DX,DMA_CONTROL
```

```
                    out     DX,AX
                    pop     DX
                    pop     BX
                    pop     AX
                    leave
                    ret
set_dma_            endp

code                ends
                    end
```

APPENDIX D: 80186 EXAMPLE TIMER INTERFACE CODE

```
$mod186
name                                    example_80186_timer_code
;
;  this file contains example 80186 timer routines. The first routine
;          sets up the timer and interrupt controller to cause the timer
;          to generate an interrupt every 10 milliseconds, and to service
;          interrupt to implement a real time clock. Timer 2 is used in
;          this example because no input or output signals are required.
;          The code example assumes that the peripheral control block has
;          not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1                            equ     word ptr [BP + 4]
arg2                            equ     word ptr [BP + 6]
arg3                            equ     word ptr [BP + 8]
timer_2int                      equ     19                      ;  timer 2 has vector type 19
timer_2control                  equ     0FF66h
timer_2max_ctl                  equ     0FF62h
timer_int_ctl                   equ     0FF32h                  ;  interrupt controller regs
eoi_register                    equ     0FF22h
interrupt_stat                  equ     0FF30h

data                            segment                         public 'data'
                                public  hour_,minute_,second_,msec_
msec_                           db      ?
hour_                           db      ?
minute_                         db      ?
second_                         db      ?
data                            ends

cgroup                          group   code
dgroup                          group   data

code                            segment                         public 'code'
                                public  set_time_
                                assume  cs:code,ds:dgroup
;
;  set_time(hour,minute,second) sets the time variables, initializes the
;          80186 timer2 to provide interrupts every 10 milliseconds, and
;          programs the interrupt vector for timer 2
;
set_time_                       proc    near
                                enter   0,0                     ;  set stack addressability
                                push    AX                      ;  save registers used
                                push    DX
                                push    SI
                                push    DS

                                xor     AX,AX                   ;  set the interrupt vector
                                                                ;  the timers have unique
                                                                ;  interrupt
                                                                ;  vectors even though they share
                                                                ;  the same control register
                                mov     DS,AX

                                mov     SI,4 * timer2_int
```

```
                        mov     DS:[SI],offset timer_2_interrupt_routine
                        inc     SI
                        inc     SI
                        mov     DS:[SI],CS
                        pop     DS

                        mov     AX,arg1                 ; set the time values
                        mov     hour_,AL
                        mov     AX,arg2
                        mov     minute_,AL
                        mov     AX,arg3
                        mov     second_,AL
                        mov     msec_,0

                        mov     DX,timer2_max_ctl       ; set the max count value
                        mov     AX,20000                ; 10 ms / 500 ns (timer 2 counts
                                                        ; at 1/4 the CPU clock rate)
                        out     DX,AX
                        mov     DX,timer2_control       ; set the control word
                        mov     AX,1110000000000001b    ; enable counting
                                                        ; generate interrupts on TC
                                                        ; continuous counting
                        out     DX,AX

                        mov     DX,timer_int_ctl        ; set up the interrupt controller
                        mov     AX,0000b                ; unmask interrupts
                                                        ; highest priority interrupt
                        out     DX,AX
                        sti                             ; enable processor interrupts

                        pop     SI
                        pop     DX
                        pop     AX
                        leave
                        ret
set_time_               endp

timer2_interrupt_routine proc   far
                        push    AX
                        push    DX

                        cmp     msec_,99                ; see if one second has passed
                        jae     bump_second             ; if above or equal...
                        inc     msec_
                        jmp     reset_int_ctl
bump_second:
                        mov     msec_,0                 ; reset millisecond
                        cmp     second_,59              ; see if one minute has passed
                        jae     bump_minute
                        inc     second_
                        jmp     reset_int_ctl
bump_minute:
                        mov     second_,0
                        cmp     minute_,59              ; see if one hour has passed
                        jae     bump_hour
                        inc     minute_
                        jmp     reset_int_ctl
```

```
bump_hour:
                        mov     minute_,0
                        cmp     hour_,12                ;  see if 12 hours have passed
                        jae     reset_hour
                        inc     hour_
                        jmp     reset_int_ctl
reset_hour:
                        mov     hour_,1
reset_int_ctl:

                        mov     DX,eoi_register
                        mov     AX,8000h                ;  non-specific end of interrupt
                        out     DX,AX

                        pop     DX
                        pop     AX
                        iret
timer2_interrupt_routine endp
code                    ends
                        end
$mod186
name                    example_80186_baud_code
;
;  this file contains example 80186 timer routines. The second routine
;       sets up the timer as a baud rate generator. In this mode,
;       Timer 1 is used to continually output pulses with a period of
;       6.5 usec for use with a serial controller at 9600 baud
;       programmed in divide by 16 mode (the actual period required
;       for 9600 baud is 6.51 usec). This assumes that the 80186 is
;       running at 8 MHz. The code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah

code                    segment                         public 'code'
                        assume  cs:code
;
;  set_baud() initializes the 80186 timer1 as a baud rate generator for
;       a serial port running at 9600 baud
;
set_baud_               proc    near
                        push    AX                      ;  save registers used
                        push    DX

                        mov     DX,timer1_max_cnt       ;  set the max count value
                        mov     AX,13                   ;  500ns * 13 = 6.5 usec
                        out     DX,AX
                        mov     DX,timer1_control       ;  set the control word
                        mov     AX,1100000000000001b    ;  enable counting
                                                        ;  no interrupt on TC
                                                        ;  continuous counting
                                                        ;  single max count register
                        out     DX,AX

                        pop     DX
                        pop     AX
```

```
                        ret
set_baud_               endp
code                    ends
                        end
$mod186
name                    example_80186_count_code
;
;  this file contains example 80186 timer routines. The third routine
;       sets up the timer as an external event counter. In this mode,
;       Timer 1 is used to count transitions on its input pin. After
;       the timer has been set up by the routine, the number of
;       events counted can be directly read from the timer count
;       register at location FF58H in I/O space. The timer will
;       count a maximum of 65535 timer events before wrapping
;       around to zero. This code example also assumes that the
;       peripheral control block has not been moved from its reset
;       location (FF00-FFFF in I/O space).
;
timer1_control          equ     0FF5Eh
timer1_max_cnt          equ     0FF5Ah
timer1_cnt_reg          equ     0FF58H

code                    segment                         public 'code'
                        assume  cs:code
;
;  set_count() initializes the 80186 timer1 as an event counter
;
set_count_              proc    near
                        push    AX              ;  save registers used
                        push    DX

                        mov     DX,timer1_max_cnt   ;  set the max count value
                        mov     AX,0            ;  allows the timer to count
                                                ;  all the way to FFFFH
                        out     DX,AX
                        mov     DX,timer1_control   ;  set the control word
                        mov     AX,1100000000000101b    ;  enable counting
                                                ;  no interrupt on TC
                                                ;  continuous counting
                                                ;  single max count register
                                                ;  external clocking
                        out     DX,AX

                        xor     AX,AX           ;  zero AX
                        mov     DX,timer1_cnt_reg   ;  and zero the count in the timer
                        out     DX,AX           ;  count register

                        pop     DX
                        pop     AX
                        ret

set_count_              endp
code                    ends
                        end
```

APPENDIX E: 80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```
$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller to provide
;       two cascaded interrupt inputs (through an external 8259A
;       interrupt controller on pins INT0/INT2) and two direct
;       interrupt inputs (on pins INT1 and INT3). The default priority
;       levels are used. Because of this, the priority level programmed
;       into the control register is set the 111, the level all
;       interrupts are programmed to at reset.
;
int0_control            equ     0FF38H
int_mask                equ     0FF28H
;
code                    segment                         public 'code'
                        assume  CS:code
set_int_                proc    near
                        push    DX
                        push    AX

                        mov     AX,0100111B             ;  cascade mode
                                                        ;  interrupt unmasked
                        mov     DX,int0_control
                        out     DX,AX

                        mov     AX,01001101B            ;  now unmask the other external
                                                        ;  interrupts
                        mov     DX,int_mask
                        out     DX,AX
                        pop     AX
                        pop     DX
                        ret
set_int_                endp
code                    ends
                        end
$mod186
name                    example_80186_interrupt_code
;
;  This routine configures the 80186 interrupt controller into iRMX 86
;       mode. This code does not initialize any of the 80186
;       integrated peripheral control registers, nor does it initialize
;       the external 8259A or 80130 interrupt controller.
;
relocation_reg          equ     0FFFEH
;
code                    segment                         public 'code'
                        assume  CS:code
set_rmx_                proc    near
                        push    DX
                        push    AX

                        mov     DX,relocation_reg
                        in      AX,DX                   ;  read old contents of register
                        or      AX,0000000000000000B    ;  set the RMX mode bit
                        out     DX,AX
```

```
                    pop     AX
                    pop     DX
                    ret
set_rmx_            endp
code                ends
                    end
```

APPENDIX F: 80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```
name                    example_80186_system_init
;
;  This file contains a system initialization routine for the 80186
;       or the 8086. The code determines whether it is running on
;       an 80186 or an 8086, and if it is running on an 80186, it
;       initializes the integrated chip select registers.
;
restart                 segment  at                    0FFFFh
;
;  This is the processor reset address at 0FFFF0H
;
                        org      0
                        jmp      far ptr initialize
restart                 ends
;
                        extrn    monitor:far
init_hw                 segment  at                    0FFF0h
                        assume   CS:init_hw
;
;  This segment initializes the chip selects. It must be located in the
;       top 1K to insure that the ROM remains selected in the 80186
;       system until the proper size of the select area can be programmed.
;
UMCS_reg                equ      0FFA0H        ;  chip select register locations
LMCS_reg                equ      0FFA2H
PACS_reg                equ      0FFA4H
MPCS_reg                equ      0FFA8H
UMCS_value              equ      0F800H        ;  64K, no wait states
LMCS_value              equ      07F8H         ;  32K, no wait states
PACS_value              equ      72H           ;  peripheral base at 400H, 2 ws
MPCS_value              equ      0BAH          ;  PCS5 and 6 supplies,
                                               ;  peripherals in I/O space
initialize              proc     far
                        mov      AX,2          ;  determine if this is an
                        mov      CL,33         ;  8086 or an 80186 (checks
                        shr      AX,CL         ;  to see if the multiple bit
                        test     AX,1          ;  shift value was ANDed)
                        jz       not_80186

                        mov      DX,UMCS_reg   ;  program the UMCS register
                        mov      AX,UMCS_value
                        out      DX,AX

                        mov      DX,LMCS_reg   ;  program the LMCS register
                        mov      AX,LMCS_value
                        out      DX,AX

                        mov      DX,PACS_reg   ;  set up the peripheral chip
                                               ;  selects (note the mid-range
                                               ;  memory chip selects are not
                                               ;  needed in this system, and
                                               ;  are thus not initialized
                        mov      AX,PACS_value
                        out      DX,AX
```

```
                        mov     DX,MPCS_reg
                        mov     AX,MPCS_value
                        out     DX,AX
;
;   Now that the chip selects are all set up, the main program of the
;           computer may be executed.
;
not_80186:
                        jmp     far ptr monitor
initialize              endp
init_hw                 ends
                        end
```

APPENDIX G: 80186 WAIT STATE PERFORMANCE

Because the 80186 contains seperate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performace degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not effect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution times calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot access the opcode bytes immediatly upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1 Exec Time (μsec)	Program 1 Perf Degr	Program 2 Exec Time (μsec)	Program 2 Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

```
$mod186
name                    example_wait_state_performance
;
;  This file contains two programs which demonstrate the 80186 performance
;       degradation as wait states are inserted. Program 1 performs a
;       transformation between two types of characters sets, then copies
;       the transformed characters back to the original buffer (which is 64
;       bytes long. Program 2 performs the same type of transformation, however
;       instead of performing a table lookup, it multiplies each number in the
;       original 32 word buffer by a constant (3, note the use of the integer
;       immediate multiply instruction). Program "nothing" is used to measure
;       the call and return times from the driver program only.
;
cgroup                  group   code
dgroup                  group   data
data                    segment                 public 'data'
```

```
t_table         db      256 dup (?)
t_string        db      64 dup (?)
m_array         dw      32 dup (?)
data            ends

code            segment                         public 'code'
                assume  CS:cgroup,DS:dgroup
                public  bench_1,bench_2,nothing_,wait_state_,set_timer_
bench_1         proc    near
                push    SI                      ; save registers used
                push    CX
                push    BX
                push    AX

                mov     CX,64                   ; translate 64 bytes
                mov     SI,0
                mov     BH,0

loop_back:
                mov     BL,t_string[SI]         ; get the byte
                mov     AL,t_table[BX]          ; translate byte
                mov     t_string[SI],AL         ; and store it
                inc     SI                      ; increment index
                loop    loop_back               ; do the next byte

                pop     AX
                pop     BX
                pop     CX
                pop     SI
                ret
bench_1         endp

bench_2         proc    near
                push    AX                      ; save registers used
                push    SI
                push    CX

                mov     CX,32                   ; multiply 32 numbers
                mov     SI,offset m_array

loop_back_2:
                imul    AX,word ptr [SI],3      ; immediate multiply
                mov     word ptr [SI],AX
                inc     SI
                inc     SI
                loop    loop_back_2

                pop     CX
                pop     SI
                pop     AX
                ret
bench_2_        endp
```

```
nothing_        proc    near
                ret
nothing_        endp
;
;  wait_state(n) sets the 80186 LMCS register to the number of wait states
;          (0 to 3) indicated by the parameter n (which is passed on the stack).
;          No other bits of the LMCS register are modified.
;
wait_state_     proc    near
                enter   0,0                     ;  set up stack frame
                push    AX                      ;  save registers used
                push    BX
                push    DX

                mov     BX,word ptr [BP + 4]    ;  get argument
                mov     DX,0FFA2h               ;  get current LMCS register
contents
                in      AX,DX

                and     AX,0FFFCh               ;  and off existing ready bits
                and     BX,3                    ;  insure ws count is good
                or      AX,BX                   ;  adjust the ready bits
                out     DX,AX                   ;  and write to LMCS

                pop     DX
                pop     BX
                pop     AX
                leave                           ;  tear down stack frame
                ret
wait_state_     endp
;
;  set_timer() initializes the 80186 timers to count microseconds. Timer 2
;       is set up as a prescaler to timer 0, the microsecond count can be read
;          directly out of the timer 0 count register at location FF50H in I/O
;          space.
;
set_timer_      proc    near
                push    AX
                push    DX

                mov     DX,0ff66h               ;  stop timer 2
                mov     AX,4000h
                out     DX,AX

                mov     DX,0ff50h               ;  clear timer 0 count
                mov     AX,0
                out     DX,AX

                mov     DX,0ff52h               ;  timer 0 counts up to 65535
                mov     AX,0
                out     DX,AX
```

```
                    mov     DX,0ff56h       ; enable timer 0
                    mov     AX,0c009h
                    out     DX,AX

                    mov     DX,0ff60h       ; clear timer 2 count
                    mov     AX,0
                    out     DX,AX

                    mov     DX,0ff62h       ; set maximum count of timer 2
                    mov     AX,2
                    out     DX,AX

                    mov     DX,0ff66h       ; re-enable timer 2
                    mov     AX,0c001h
                    out     DX,AX

                    pop     DX
                    pop     AX
                    ret
set_timer_          endp
code                ends
                    end
```

APPENDIX H: 80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "$mod186" switch must be given to the assembler. This can be done by placing the line: "$mod186" at the beginning of the assembly language file.

- **PUSH immediate**

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

- **PUSHA, POPA**

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DS, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

- **IMUL by an immediate value**

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

- **shifts/rotates by an immediate value**

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e. the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

- **block input/output**

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceeded by a REPeat prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceeded by a REPeat prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

```
INSB          ; perform byte input
REP OUTSW     ; perform word block output
```

- **BOUND**

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediatly the memory location containing the lower bound of the array.

- **ENTER and LEAVE**

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP                 /* save the previous frame
                           pointer */
if level = 0 then
  BP := SP;
else    temp1 := SP; /* save current frame pointer
                        */
  temp2 := level - 1;
  do while temp2 > 0 /* copy down previous level
                        frame */
    BP := BP - 2;     /* pointers */
    PUSH [BP];
  BP := temp1;
  PUSH BP;          /* put current level frame
                       pointer */

/* in the save area */
SP := SP - disp;    /* create space on the stack
                       for */

/* local variables */
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction. As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

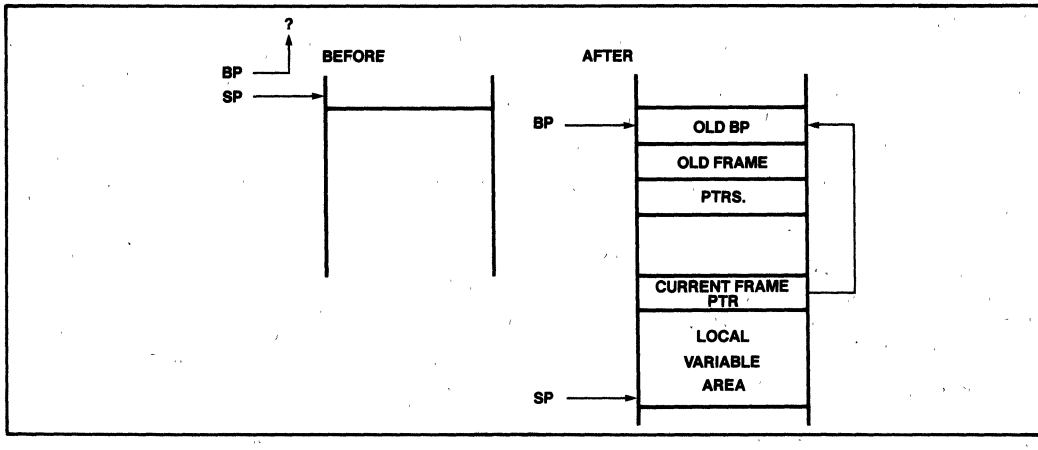


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I: 80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8-AD15 on the 80186 are transformed to A8-A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{BHE}$/S7 is always defined HIGH by the 80188, since the upper half of the data bus is non-existant.
- The DMA controller of the 80188 only performs byte transfers. The $\overline{B}$/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and the 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, NOT in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When an access is made to the internal registers, only a single bus cycle will be run, and only the lower 8-bits of the written data will be driven on the external bus. All accesses to registers within the integrated peripheral block must be WORD accesses.

iAPX 86/10
16-BIT HMOS MICROPROCESSOR

8086/8086-2/8086-1

- **Direct Addressing Capability 1 MByte of Memory**
- **Architecture Designed for Powerful Assembly Language and Efficient High Level Languages.**
- **14 Word, by 16-Bit Register Set with Symmetrical Operations**
- **24 Operand Addressing Modes**
- **Bit, Byte, Word, and Block Operations**
- **8 and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal Including Multiply and Divide**
- **Range of Clock Rates: 5 MHz for 8086, 8 MHz for 8086-2, 10 MHz for 8086-1**
- **MULTIBUS™ System Compatible Interface**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel iAPX 86/10 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The iAPX 86/10 operates in both single processor and multiple processor configurations to achieve high performance levels.

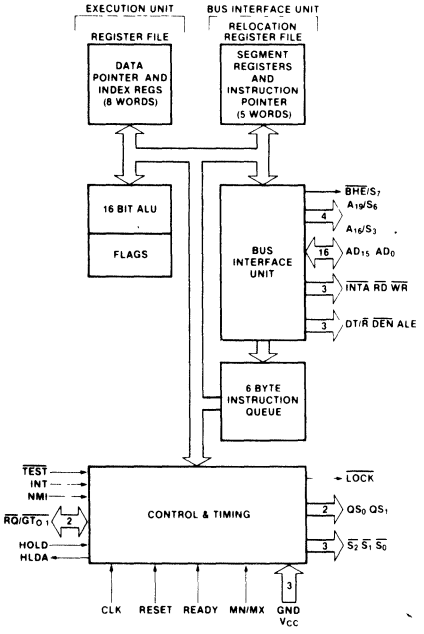
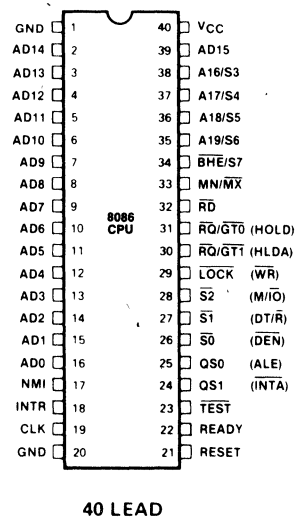


Figure 1. iAPX 86/10 CPU Block Diagram



40 LEAD

Figure 2. iAPX 86/10 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for iAPX 86 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD_{15}-AD_0	2-16, 39	I/O	**Address Data Bus:** These lines constitute the time multiplexed memory/IO address (T_1) and data (T_2, T_3, T_W, T_4) bus. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7-D_0. It is LOW during T_1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A_0 to condition chip select functions. (See $\overline{BHE}$.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge."
A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3	35-38	O	**Address/Status:** During T_1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T_2, T_3, T_W, and T_4. The status of the interrupt enable FLAG bit (S_5) is updated at the beginning of each CLK cycle. A_{17}/S_4 and A_{16}/S_3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."
$\overline{BHE}/S_7$	34	O	**Bus High Enable/Status:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}-D_8. Eight-bit oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T_2, T_3, and T_4. The signal is active LOW, and floats to 3-state OFF in "hold." It is LOW during T_1 for the first interrupt acknowledge cycle.
$\overline{RD}$	32	O	**Read:** Read strobe indicates that the processor is performing a memory of I/O read cycle, depending on the state of the S_2 pin. This signal is used to read devices which reside on the 8086 local bus. $\overline{RD}$ is active LOW during T_2, T_3 and T_W of any read cycle, and is guaranteed to remain HIGH in T_2 until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge."
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.
INTR	18	I	**Interrupt Request:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{TEST}$	23	I	**$\overline{TEST}$:** input is examined by the "Wait" instruction. If the $\overline{TEST}$ input is LOW execution continues, otherwise the processor waits in an "Idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

A_{17}/S_4	A_{16}/S_3	Characteristics
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
S_6 is 0 (LOW)		

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/ to odd address
1	0	Lower byte from/ to even address
1	1	None

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
NMI	17	I	**Non-maskable interrupt:** an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.
RESET	21	I	**Reset:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**Clock:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** +5V power supply pin.
GND	1, 20		**Ground**
MN/$\overline{MX}$	33	I	**Minimum/Maximum:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., MN/$\overline{MX}$ = V_{SS}). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$	26-28	O	**Status:** active during T_4, T_1, and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals float to 3-state OFF in "hold acknowledge." These status lines are encoded as shown.
$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$	30, 31	I/O	**Request/Grant:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT_0}$ having higher priority than $\overline{RQ}/\overline{GT_1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows (see Figure 9): 1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1). 2. During a T_4 or T_I clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T_4 of the cycle when all the following conditions are met: 1. Request occurs on or before T_2. 2. Current cycle is not the low byte of a word (on an odd address). 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristics
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1 (HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
			If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge."
QS_1, QS_0	24, 25	O	**Queue Status:** The queue status is valid during the CLK cycle after which the queue operation is performed. QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue. QS_1 / QS_0 / CHARACTERISTICS 0 (LOW) / 0 / No Operation 0 / 1 / First Byte of Op Code from Queue 1 (HIGH) / 0 / Empty the Queue 1 / 1 / Subsequent Byte from Queue

The following pin function descriptions are for the 8086 in minimum mode (i.e., MN/$\overline{MX}$ = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

M/$\overline{IO}$	28	O	**Status line:** logically equivalent to S_2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/$\overline{IO}$ becomes valid in the T_4 preceding a bus cycle and remains valid until the final T_4 of the cycle (M = HIGH, IO = LOW). M/$\overline{IO}$ floats to 3-state OFF in local bus "hold acknowledge."
$\overline{WR}$	29	O	**Write:** indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/$\overline{IO}$ signal. $\overline{WR}$ is active for T_2, T_3 and T_W of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge."
$\overline{INTA}$	24	O	**$\overline{INTA}$** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T_2, T_3 and T_W of each interrupt acknowledge cycle.
ALE	25	O	**Address Latch Enable:** provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T_1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**Data Transmit/Receive:** needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/$\overline{R}$ is equivalent to $\overline{S_1}$ in the maximum mode, and its timing is the same as for M/$\overline{IO}$. (T = HIGH, R = LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge."
$\overline{DEN}$	26	O	**Data Enable:** provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access and for INTA cycles. For a read or $\overline{INTA}$ cycle it is active from the middle of T_2 until the middle of T_4, while for a write cycle it is active from the beginning of T_2 until the middle of T_4. $\overline{DEN}$ floats to 3-state OFF in local bus "hold acknowledge."
HOLD, HLDA	31, 30	I/O	**HOLD:** indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement in the middle of a T_1 clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. The same rules as for $\overline{RQ}$/$\overline{GT}$ apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setup time.

FUNCTIONAL DESCRIPTION

GENERAL OPERATION

The internal functions of the iAPX 86/10 processor are partitioned logically into two processing units. The first is the Bus Interface Unit (BIU) and the second is the Execution Unit (EU) as shown in the block diagram of Figure 1.

These units can interact directly but for the most part perform as separate asynchronous operational processors. The bus interface unit provides the functions related to instruction fetching and queuing, operand fetch and store, and address relocation. This unit also provides the basic bus control. The overlap of instruction pre-fetching provided by this unit serves to increase processor performance through improved bus bandwidth utilization. Up to 6 bytes of the instruction stream can be queued while waiting for decoding and execution.

The instruction stream queuing mechanism allows the BIU to keep the memory utilized very efficiently. Whenever there is space for at least 2 bytes in the queue, the BIU will attempt a word fetch memory cycle. This greatly reduces "dead time" on the memory bus. The queue acts as a First-In-First-Out (FIFO) buffer, from which the EU extracts instruction bytes as required. If the queue is empty (following a branch instruction, for example), the first byte into the queue immediately becomes available to the EU.

The execution unit receives pre-fetched instructions from the BIU queue and provides un-relocated operand addresses to the BIU. Memory operands are passed through the BIU for processing by the EU, which passes results to the BIU for storage. See the Instruction Set description for further register set and architectural descriptions.

MEMORY ORGANIZATION

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3a.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries and are thus not constrained to even boundaries as is the case in many 16-bit computers. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU automatically performs the proper number of memory accesses, one if the word operand is on an even byte boundary and two if it is on an odd byte boundary. Except for the performance penalty, this double access is transparent to the software. This performance penalty does not occur for instruction fetches, only word operands.

Physically, the memory is organized as a high bank (D_{15}-D_8) and a low bank (D_7-D_0) of 512K 8-bit bytes addressed in parallel by the processor's address lines

A_{19} - A_1. Byte data with even addresses is transferred on the D_7-D_0 bus lines while odd addressed byte data (A_0 HIGH) is transferred on the D_{15}-D_8 bus lines. The processor provides two enable signals, $\overline{BHE}$ and A_0, to selectively allow reading from or writing into either an odd byte location, even byte location, or both. The instruction stream is fetched from memory as words and is addressed internally by the processor to the byte level as necessary.

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

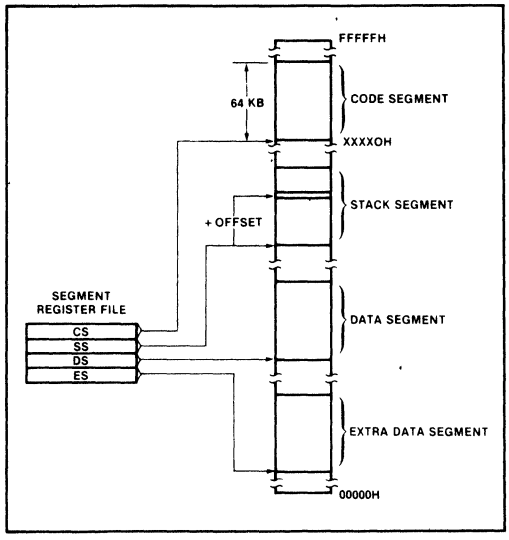


Figure 3a. Memory Organization

In referencing word data the BIU requires one or two memory cycles depending on whether or not the starting byte of the word is on an even or odd address, respectively. Consequently, in referencing word operands performance can be optimized by locating data on even address boundaries. This is an especially useful technique for using the stack, since odd address references to the stack may adversely affect the context switching time for interrupt processing or task multiplexing.

Certain locations in memory are reserved for specific CPU operations (see Figure 3b.) Locations from address FFFF0H through FFFFFH are reserved for operations including a jump to the initial program loading routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be. Locations 00000H through 003FFH are reserved for interrupt operations. Each of the 256 possible interrupt types has its service routine pointed to by a 4-byte pointer element consisting of a 16-bit segment address and a 16-bit offset address. The pointer elements are assumed to have been stored at the respective places in reserved memory prior to occurrence of interrupts.

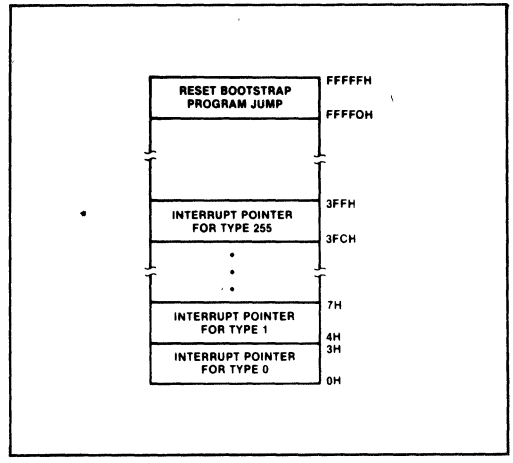


Figure 3b. Reserved Memory Locations

MINIMUM AND MAXIMUM MODES

The requirements for supporting minimum and maximum iAPX 86/10 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8086 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes dependent on the condition of the strap pin. When MN/$\overline{MX}$ pin is strapped to GND, the 8086 treats pins 24 through 31 in maximum mode. An 8288 bus controller interprets status information coded into $\overline{S}_0$,$\overline{S}_1$,$\overline{S}_2$ to generate bus timing and control signals compatible with the MULTIBUS™ architecture. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 8086 generates bus control signals itself on pins 24 through 31, as shown in parentheses in Figure 2. Examples of minimum mode and maximum mode systems are shown in Figure 4.

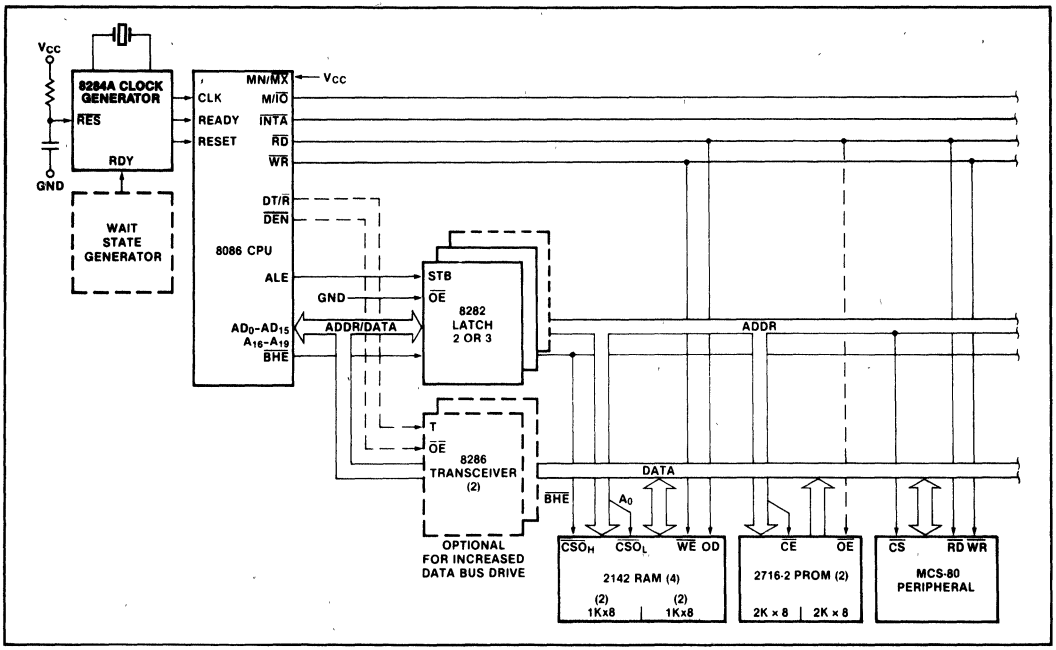


Figure 4a. Minimum Mode iAPX 86/10 Typical Configuration

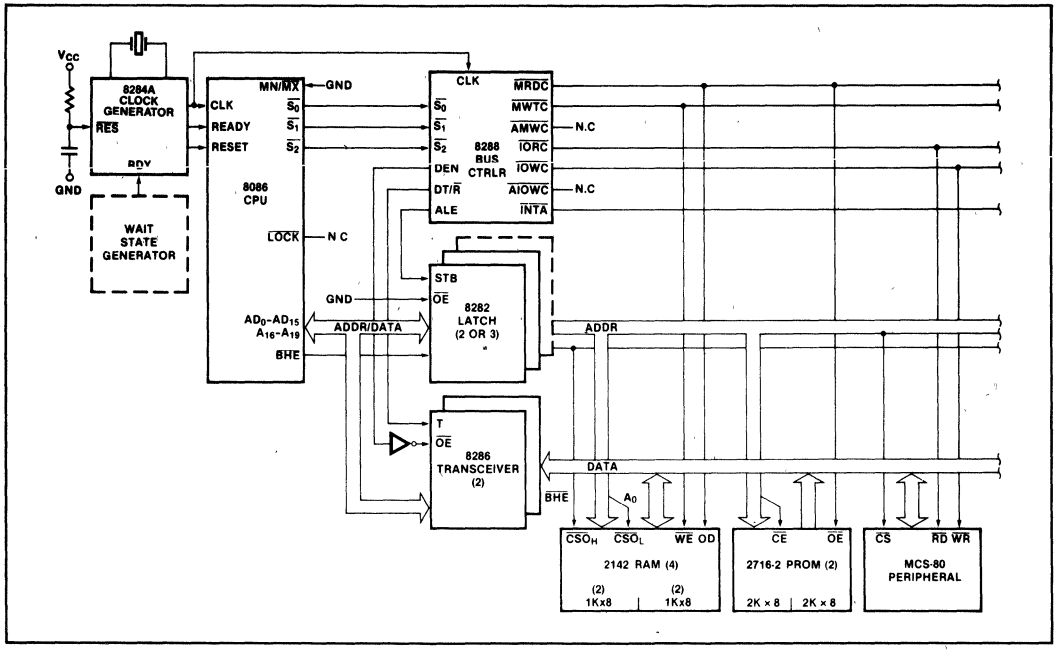


Figure 4b. Maximum Mode iAPX 86/10 Typical Configuration

BUS OPERATION

The 86/10 has a combined address and data bus commonly referred to as a time multiplexed bus. This technique provides the most efficient use of pins on the processor while permitting the use of a standard 40-lead package. This "local bus" can be buffered directly and used throughout the system with address latching provided on memory and I/O modules. In addition, the bus can also be demultiplexed at the processor with a single set of address latches if a standard non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T_1, T_2, T_3 and T_4 (see Figure 5). The address is emitted from the processor during T_1 and data transfer occurs on the bus during T_3 and T_4. T_2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device, "Wait" states (T_W) are inserted between T_3 and T_4. Each inserted "Wait" state is of the same duration as a CLK cycle. Periods can occur between 8086 bus cycles. These are referred to as "Idle" states (T_I) or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T_1 of any bus cycle the ALE (Address Latch Enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S_0}$, $\overline{S_1}$, and $\overline{S_2}$ are used, in maximum mode, by the bus controller to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S_3 through S_7 are multiplexed with high-order address bits and the $\overline{BHE}$ signal, and are therefore valid during T_2 through T_4. S_3 and S_4 indicate which segment register (see Instruction Set description) was used for this bus cycle in forming the address, according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S_5 is a reflection of the PSW interrupt enable bit. S_6=0 and S_7 is a spare status bit.

I/O ADDRESSING

In the 86/10, I/O operations can address up to a maximum of 64K I/O byte registers or 32K I/O word registers. The I/O address appears in the same format as the memory address on bus lines A_{15}-A_0. The address lines A_{19}-A_{16} are zero in I/O operations. The variable I/O instructions which use register DX as a pointer have full address capability while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space.

I/O ports are addressed in the same manner as memory locations. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on D_{15}-D_8. Care must be taken to assure that each register within an 8-bit peripheral located on the lower portion of the bus be addressed as even.

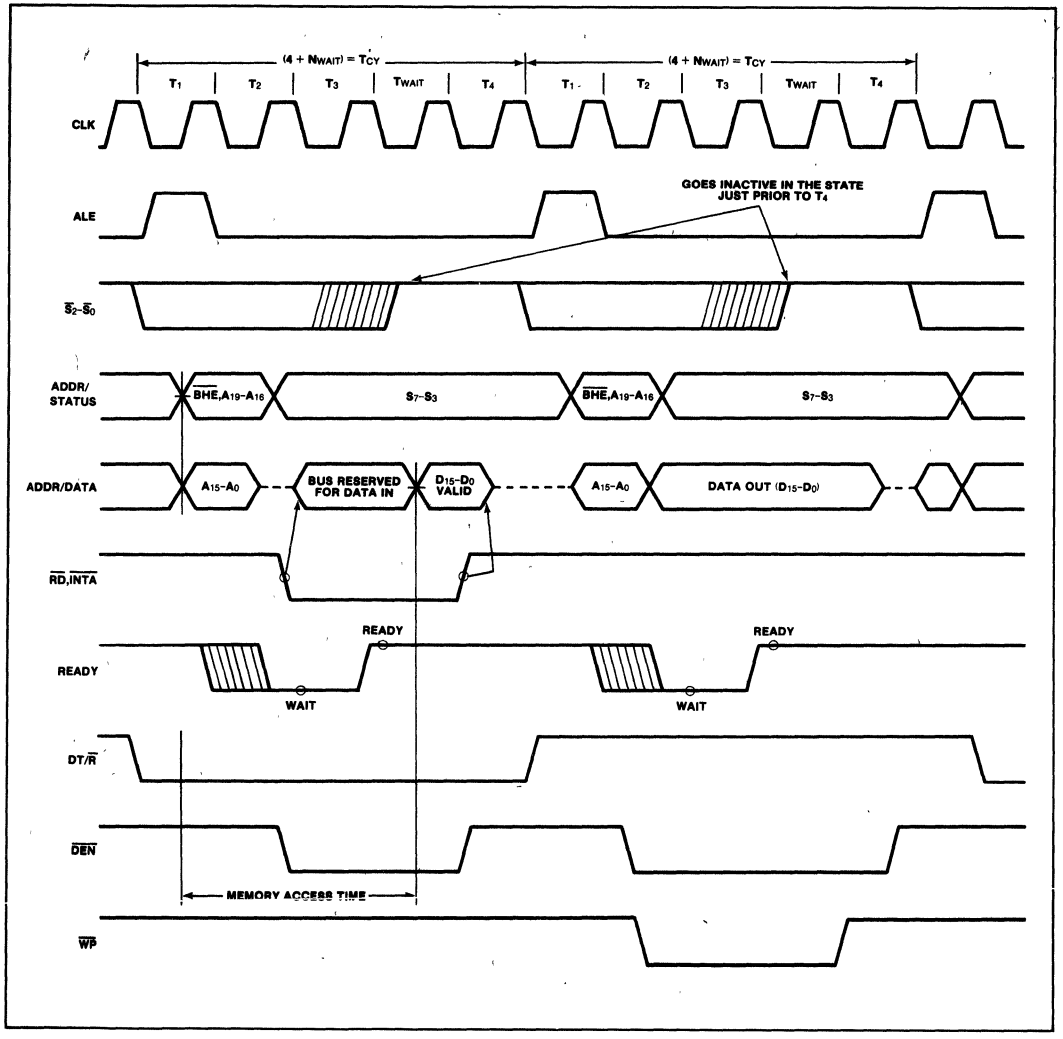


Figure 5. Basic System Timing

EXTERNAL INTERFACE

PROCESSOR RESET AND INITIALIZATION

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8086 RESET is required to be HIGH for greater than 4 CLK cycles. The 8086 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 8086 operates normally beginning with the instruction in absolute location FFFF0H (see Figure 3B). The details of this operation are specified in the Instruction Set description of the MCS-86 Family User's Manual. The RESET input is internally synchronized to the processor clock. At initialization the HIGH-to-LOW transition of RESET must occur no sooner than 50 μs after power-up, to allow complete initialization of the 8086.

NMI may not be asserted prior to the 2nd CLK cycle following the end of RESET.

INTERRUPT OPERATIONS

Interrupt operations fall into two classes; software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the Instruction Set description. Hardware interrupts can be classified as non-maskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256-element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 3b), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type" An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to "vector" through the appropriate element to the new interrupt service program location.

NON-MASKABLE INTERRUPT (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. (See Instruction Set description.)

NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLE INTERRUPT (INTR)

The 86/10 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block-type instruction. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the

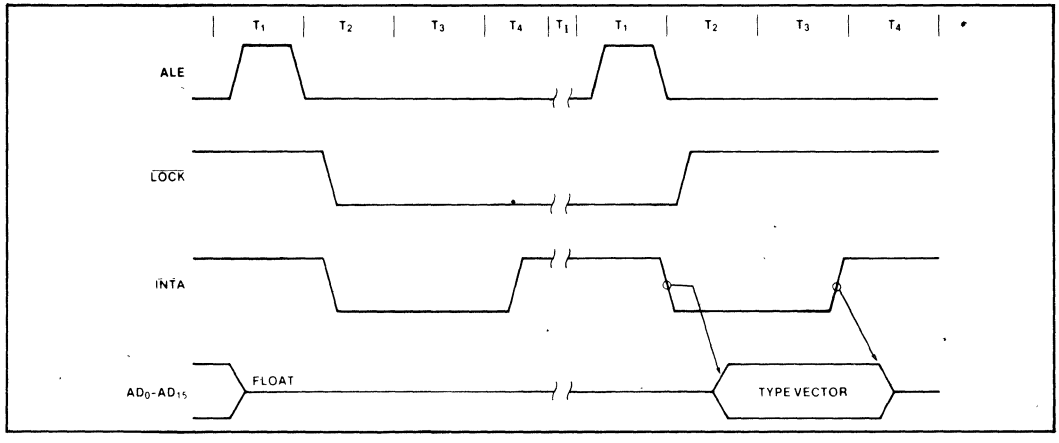


Figure 6. Interrupt Acknowledge Sequence

FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

During the response sequence (figure 6) the processor executes two successive (back-to-back) interrupt acknowledge cycles. The 8086 emits the LOCK signal from T_2 of the first bus cycle until T_2 of the second. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The INTERRUPT RETURN instruction includes a FLAGS pop which returns the status of the original interrupt enable bit when it restores the FLAGS.

HALT

When a software "HALT" instruction is executed the processor indicates that it is entering the "HALT" state in one of two ways depending upon which mode is strapped. In minimum mode, the processor issues one ALE with no qualifying bus control signals. In Maximum Mode, the processor issues appropriate HALT status on $\overline{S}_2\overline{S}_1\overline{S}_0$ and the 8288 bus controller issues one ALE. The 8086 will not leave the "HALT" state when a local bus "hold" is entered while in "HALT". In this case, the processor reissues the HALT indicator. An interrupt request or RESET will force the 8086 out of the "HALT" state.

READ/MODIFY/WRITE (SEMAPHORE) OPERATIONS VIA LOCK

The $\overline{LOCK}$ status information is provided by the processor when directly consecutive bus cycles are required during the execution of an instruction. This provides the processor with the capability of performing read/modify/write operations on memory (via the Exchange Register With Memory instruction, for example) without the possibility of another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (forced LOW) in the clock cycle following the one in which the software "LOCK" prefix instruction is decoded by the EU. It is deactivated at the end of the last bus cycle of the instruction following the "LOCK" prefix instruction. While $\overline{LOCK}$ is active a request on a RQ/GT pin will be recorded and then honored at the end of the LOCK.

EXTERNAL SYNCHRONIZATION VIA TEST

As an alternative to the interrupts and general I/O capabilities, the 8086 provides a single software-testable input known as the $\overline{TEST}$ signal. At any time the program may execute a WAIT instruction. If at that time the $\overline{TEST}$ signal is inactive (HIGH), program execution becomes suspended while the processor waits for $\overline{TEST}$ to become active. It must remain active for at least 5 CLK cycles. The WAIT instruction is re-executed repeatedly until that time. This activity does not consume bus cycles. The processor remains in an idle state while waiting. All 8086 drivers go to 3-state OFF if bus "Hold"is entered. If interrupts are enabled, they may occur while the processor is waiting. When this occurs the processor fetches the WAIT instruction one extra time, processes the interrupt, and then re-fetches and re-executes the WAIT instruction upon returning from the interrupt.

BASIC SYSTEM TIMING

Typical system configurations for the processor operating in minimum mode and in maximum mode are shown in Figures 4a and 4b, respectively. In minimum mode, the MN/$\overline{MX}$ pin is strapped to V_{CC} and the processor emits bus control signals in a manner similar to the 8085. In maximum mode, the MN/$\overline{MX}$ pin is strapped to V_{SS} and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals. Figure 5 illustrates the signal timing relationships.

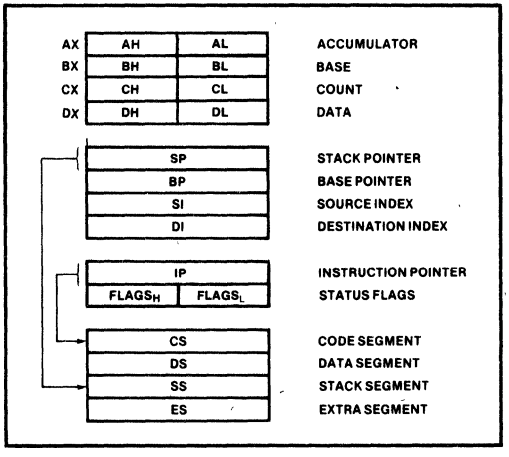


Figure 7. iAPX 86/10 Register Model

SYSTEM TIMING — MINIMUM SYSTEM

The read cycle begins in T_1 with the assertion of the Address Latch Enable (ALE) signal. The trailing (low-going) edge of this signal is used to latch the address information, which is valid on the local bus at this time, into the 8282/8283 latch. The $\overline{BHE}$ and A_0 signals address the low, high, or both bytes. From T_1 to T_4 the M/$\overline{IO}$ signal indicates a memory or I/O operation. At T_2 the address is removed from the local bus and the bus goes to a high impedance state. The read control signal is also asserted at T_2. The read ($\overline{RD}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal

to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (8286/8287) is required to buffer the 8086 local bus, signals DT/$\overline{R}$ and $\overline{DEN}$ are provided by the 8086.

A write cycle also begins with the assertion of ALE and the emission of the address. The M/$\overline{IO}$ signal is again asserted to indicate a memory or I/O write operation. In the T_2 immediately following the address emission the processor emits the data to be written into the addressed location. This data remains valid until the middle of T_4. During T_2, T_3, and T_W the processor asserts the write control signal. The write ($\overline{WR}$) signal becomes active at the beginning of T_2 as opposed to the read which is delayed somewhat into T_2 to provide time for the bus to float.

The $\overline{BHE}$ and A_0 signals are used to select the proper byte(s) of the memory/IO word to be read or written according to the following table:

$\overline{BHE}$	A0	CHARACTERISTICS
0	0	Whole word
0	1	Upper byte from/ to odd address
1	0	Lower byte from/ to even address
1	1	None

I/O ports are addressed in the same manner as memory location. Even addressed bytes are transferred on the D_7-D_0 bus lines and odd addressed bytes on D_{15}-D_8.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge signal ($\overline{INTA}$) is asserted in place of the read ($\overline{RD}$) signal and the address bus is floated. (See Figure 6.) In the second of two successive INTA cycles, a byte of information is read from bus lines D_7-D_0 as supplied by the interrupt system logic (i.e., 8259A Priority Interrupt Controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into an interrupt vector lookup table, as described earlier.

BUS TIMING—MEDIUM SIZE SYSTEMS

For medium size systems the MN/$\overline{MX}$ pin is connected to V_{SS} and the 8288 Bus Controller is added to the system as well as an 8282/8283 latch for latching the system address, and a 8286/8287 transceiver to allow for bus loading greater than the 8086 is capable of handling. Signals ALE, DEN, and DT/$\overline{R}$ are generated by the 8288 instead of the processor in this configuration although their timing remains relatively the same. The 8086 status outputs ($\overline{S}_2$, $\overline{S}_1$, and $\overline{S}_0$) provide type-of-cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence data isn't valid at the leading edge of write. The 8286/8287 transceiver receives the usual T and OE inputs from the 8288's DT/$\overline{R}$ and DEN.

The pointer into the interrupt vector table, which is passed during the second INTA cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8259A Priority Interrupt Controller is positioned on the local bus, a TTL gate is required to disable the 8286/8287 transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to Ground . . . . . −1.0 to +7V
Power Dissipation . . . . . 2.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS

(8086: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL}=2.5 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current: 8086 8086-1 8086-2		340 360 350	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0 − AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0 − AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz

A.C. CHARACTERISTICS (8086: T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)
(8086-1: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)
(8086-2: T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		−10		−8		ns	
THVCH	HOLD Setup Time	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		15		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

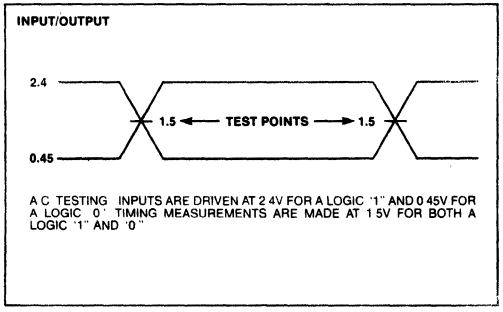
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

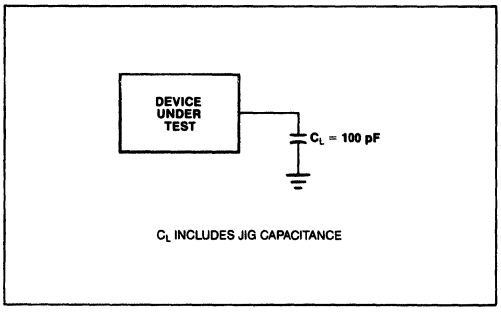
Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH−20		TCLCH−10		TCLCH-10		ns	
TCLLH	ALE Active Delay		80		40		50	ns	
TCHLL	ALE Inactive Delay		85		45		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL−10		TCHCL−10		TCHCL−10		ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	*C_L = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCHDX	Data Hold Time	10		10		10		ns	
TWHDX	Data Hold Time After WR	TCLCH−30		TCLCH−25		TCLCH−30		ns	
TCVCTV	Control Active Delay 1	10	110	10	50	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	45	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	50	10	70	ns	
TAZRL	Address Float to READ Active	0		0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	70	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	60	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL−45		TCLCL−35		TCLCL−40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	60	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL−75		2TCLCL−40		2TCLCL−50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL−60		2TCLCL−35		2TCLCL−40		ns	
TAVAL	Address Valid to ALE Low	TCLCH−60		TCLCH−35		TCLCH−40		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 8284A shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state. (8 ns into T3).

A.C. TESTING INPUT, OUTPUT WAVEFORM

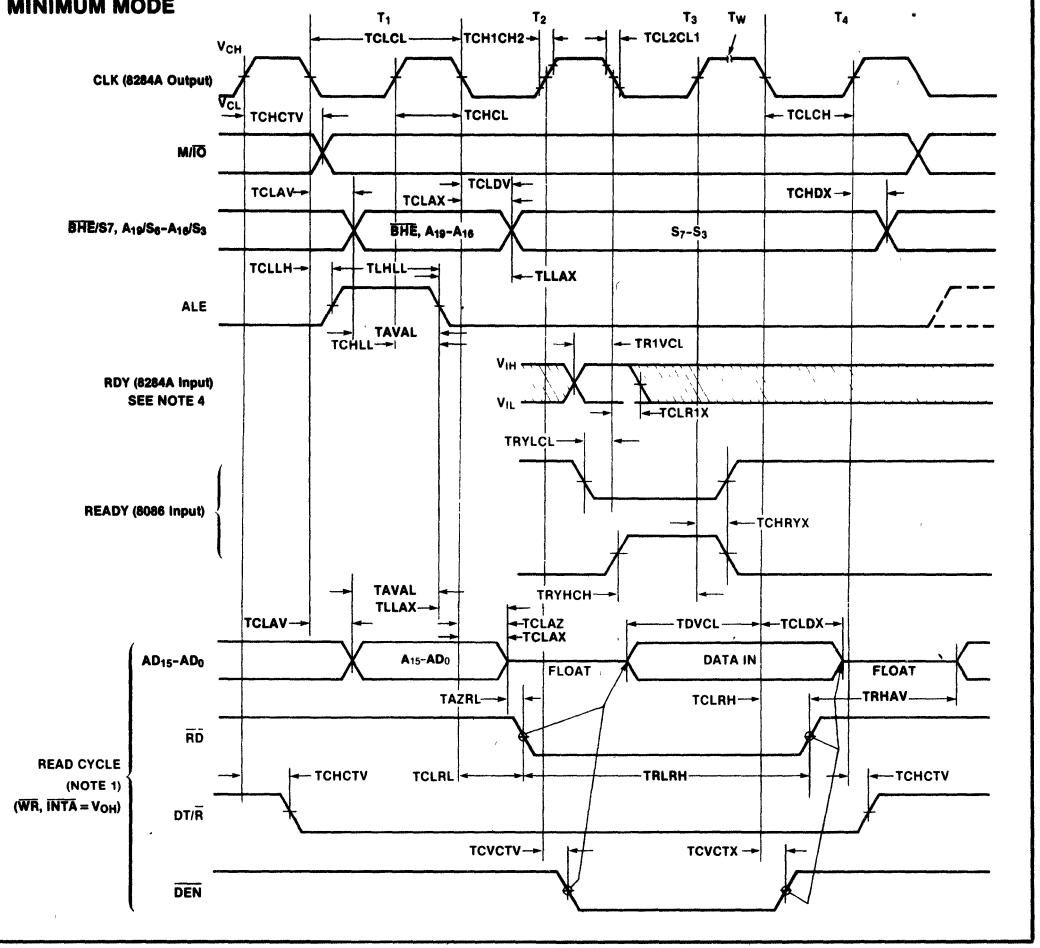


A.C. TESTING LOAD CIRCUIT



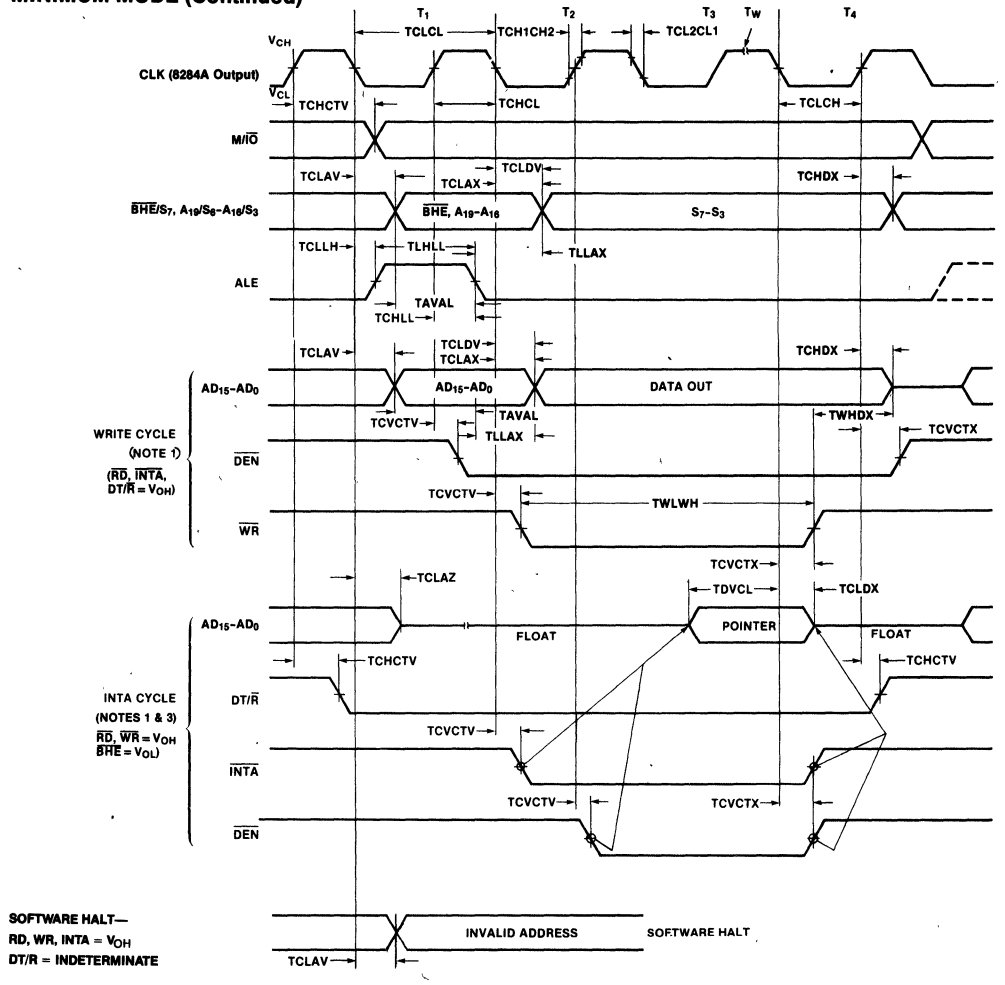
WAVEFORMS

MINIMUM MODE



WAVEFORMS (Continued)

MINIMUM MODE (Continued)



NOTES:

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control signals shown for second INTA cycle.
4. Signals at 8284A are shown for reference only.
5. All timing measurements are made at 1.5V unless otherwise noted.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER) TIMING REQUIREMENTS

Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2 (Preliminary)		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLCL	CLK Cycle Period	200	500	100	500	125	500	ns	
TCLCH	CLK Low Time	118		53		68		ns	
TCHCL	CLK High Time	69		39		44		ns	
TCH1CH2	CLK Rise Time		10		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		5		20		ns	
TCLDX	Data In Hold Time	10		10		10		ns	
TR1VCL	RDY Setup Time into 8284A (See Notes 1, 2)	35		35		35		ns	
TCLR1X	RDY Hold Time into 8284A (See Notes 1, 2)	0		0		0		ns	
TRYHCH	READY Setup Time into 8086	118		53		68		ns	
TCHRYX	READY Hold Time into 8086	30		20		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		−10		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, $\overline{TEST}$) (See Note 2)	30		15		15		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	30		12		15		ns	
TCHGX	$\overline{RQ}$ Hold Time into 8086	40		20		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12		12	ns	From 2.0V to 0.8V

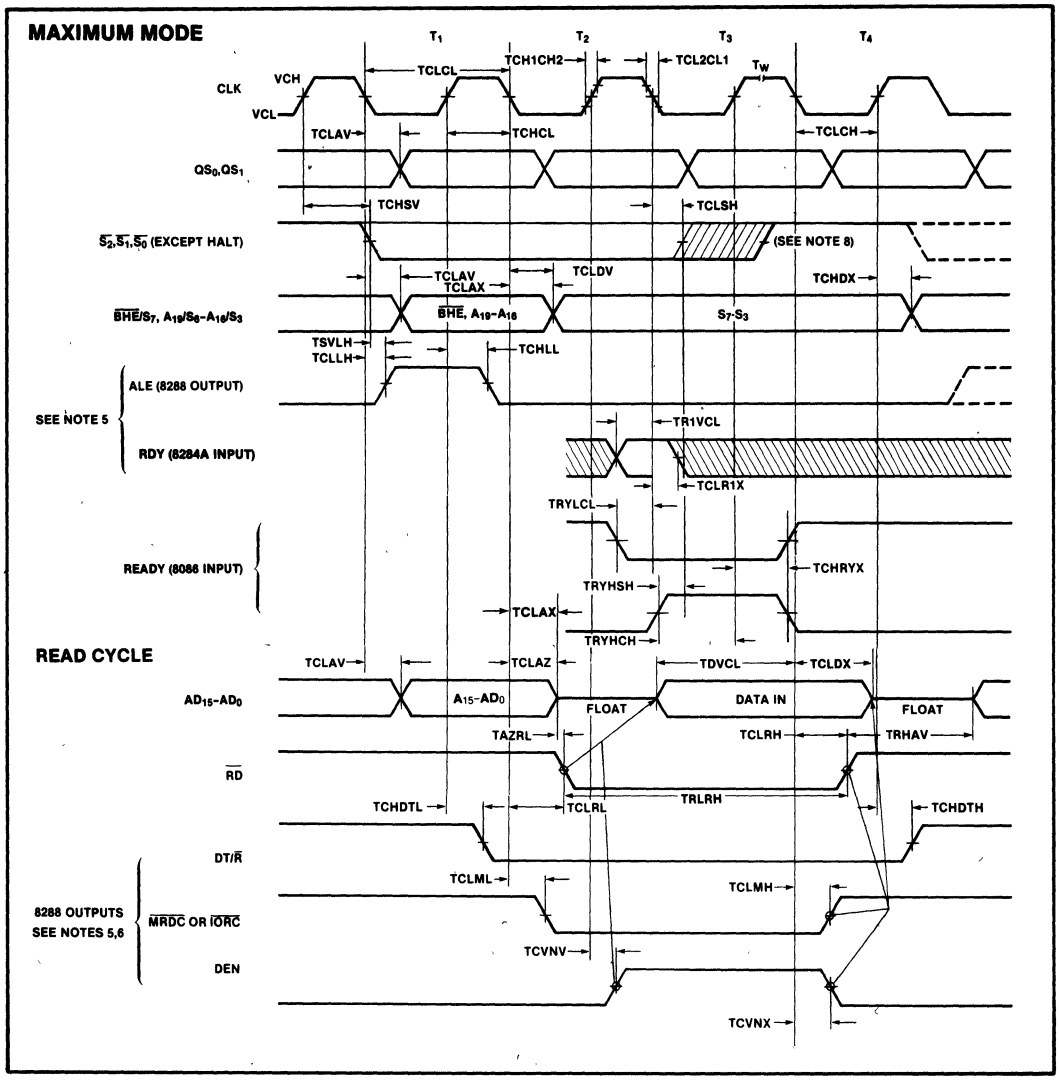
NOTES:
1. Signal at 8284A or 8288 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T3 and wait states.
4. Applies only to T2 state (8 ns into T3).

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

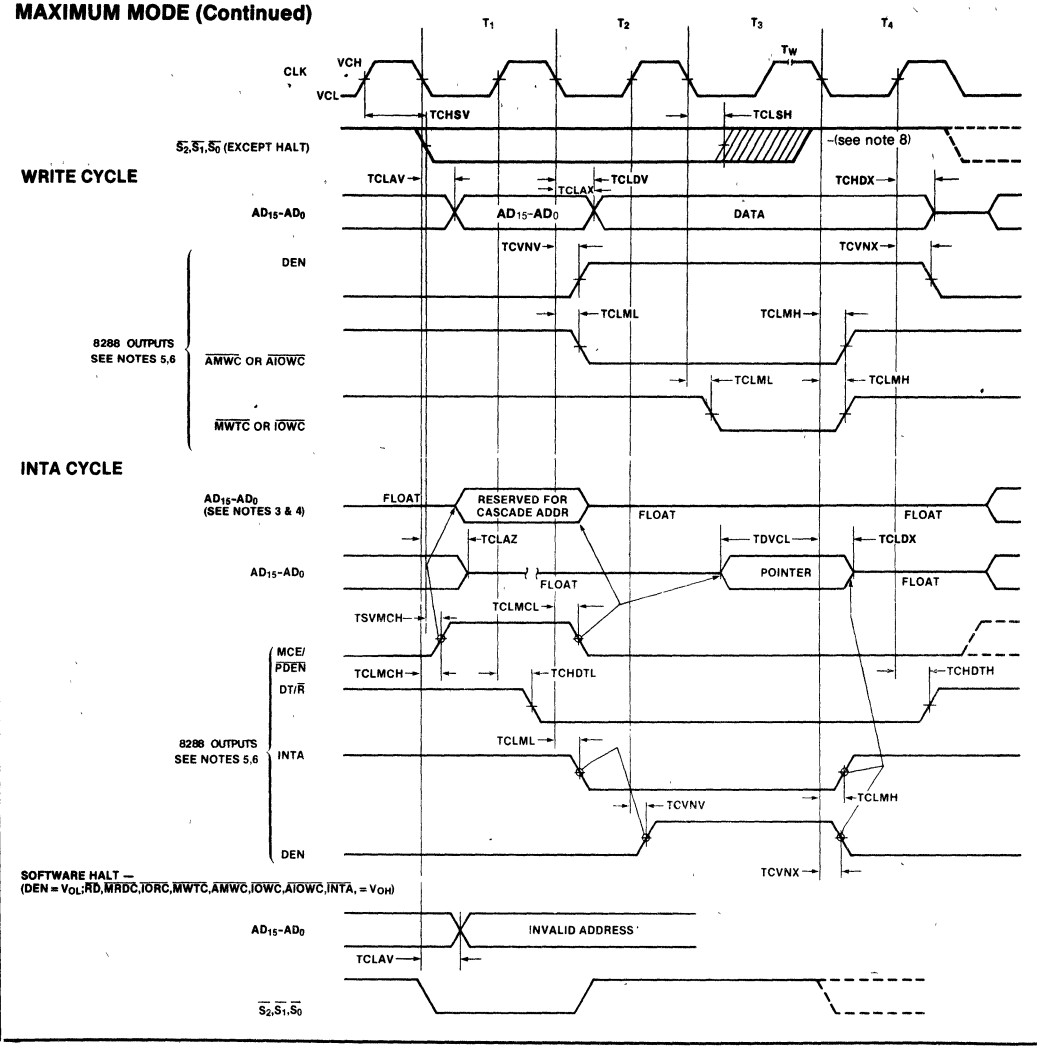
Symbol	Parameter	8086		8086-1 (Preliminary)		8086-2 (Preliminary)		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TCLML	Command Active Delay (See Note 1)	10	35	10	35	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		45		65	ns	
TCHSV	Status Active Delay	10	110	10	45	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	55	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	50	10	60	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	10	40	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15		15	ns	
TSVMCH	Status Valid to MCE High (See Note 1)		15		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15		15	ns	C_L = 20-100 pF for all 8086 Outputs (In addition to 8086 self-load)
TCLMCL	MCE Inactive Delay (See Note 1)		15		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	50	10	60	ns	
TCHDX	Data Hold Time	10		10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		0		ns	
TCLRL	RD Active Delay	10	165	10	70	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	60	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL−45		TCLCL−35		TCLCL−40		ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30		30	ns	
TCLGL	GT Active Delay	0	85	0	45	0	50	ns	
TCLGH	GT Inactive Delay	0	85	0	45	0	50	ns	
TRLRH	RD Width	2TCLCL−75		2TCLCL−40		2TCLCL−50		ns	
TOLOH	Output Rise Time		20		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

WAVEFORMS


MAXIMUM MODE
T1
T2
T3
T4
CLK
VCH
VCL
TCLCL
TCH1CH2
TCL2CL1
TW
TCLAV
TCHCL
TCLCH
QS0,QS1
TCHSV
TCLSH
S2,S1,S0 (EXCEPT HALT)
(SEE NOTE 8)
TCLAV
TCLAX
TCLDV
TCHDX
BHE/S7, A19/S6-A16/S3
BHE, A19-A16
S7-S3
TSVLH
TCLLH
TCHLL
SEE NOTE 5
ALE (8288 OUTPUT)
RDY (8284A INPUT)
TR1VCL
TCLR1X
TRYLCL
READY (8086 INPUT)
TCHRYX
TRYHSH
TCLAX
TRYHCH
READ CYCLE
TCLAV
TCLAZ
TDVCL
TCLDX
AD15-AD0
A15-AD0
FLOAT
DATA IN
FLOAT
TAZRL
TCLRH
TRHAV
RD
TCHDTL
TCLRL
TRLRH
TCHDTH
DT/R
TCLML
TCLMH
8288 OUTPUTS
SEE NOTES 5,6
MRDC OR IORC
DEN
TCVNV
TCVNX

WAVEFORMS (Continued)

MAXIMUM MODE (Continued)

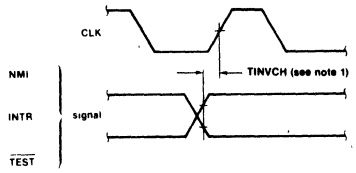


NOTES:

1. All signals switch between V_{OH} and V_{OL} unless otherwise specified.
2. RDY is sampled near the end of T_2, T_3, T_W to determine if T_W machines states are to be inserted.
3. Cascade address is valid between first and second INTA cycle.
4. Two INTA cycles run back-to-back. The 8086 LOCAL ADDR/DATA BUS is floating during both INTA cycles. Control for pointer address is shown for second INTA cycle.
5. Signals at 8284A or 8288 are shown for reference only.
6. The issuance of the 8288 command and control signals ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$ and DEN) lags the active high 8288 CEN.
7. All timing measurements are made at 1.5V unless otherwise noted.
8. Status inactive in state just prior to T_4.

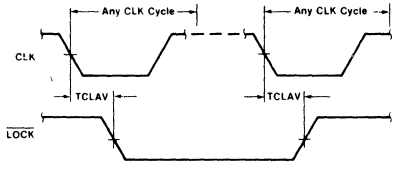
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

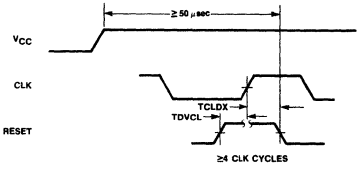


NOTE 1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

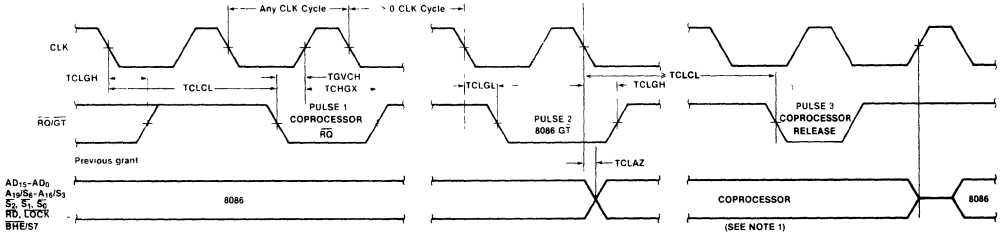
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



RESET TIMING



REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTES 1 THE COPROCESSOR MAY NOT DRIVE THE BUSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)

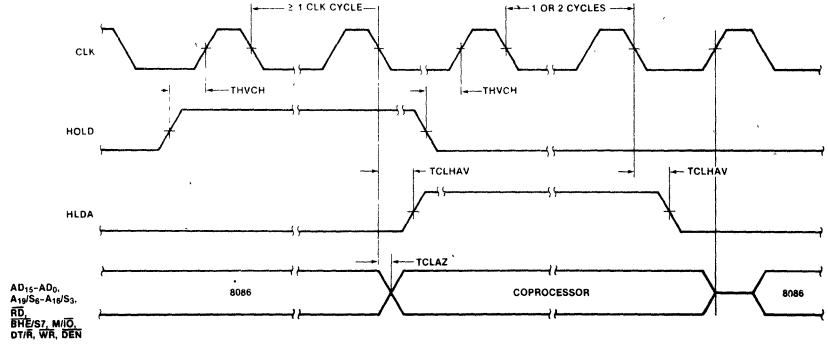


Table 2. Instruction Set Summary

DATA TRANSFER

MOV = Move.	76543210	76543210	76543210	76543210
Register/memory to/from register	100010dw	mod reg r/m		
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w 1
Immediate to register	1011w reg	data	data if w 1	
Memory to accumulator	1010000w	addr low	addr high	
Accumulator to memory	1010001w	addr low	addr high	
Register/memory to segment register	10001110	mod 0 reg r/m		
Segment register to register/memory	10001100	mod 0 reg r/m		

PUSH = Push			
Register/memory	11111111	mod 110 r/m	
Register	01010 reg		
Segment register	000 reg 110		

POP = Pop			
Register/memory	10001111	mod 000 r/m	
Register	01011 reg		
Segment register	000 reg 111		

XCHG = Exchange			
Register/memory with register	1000011w	mod reg r/m	
Register with accumulator	10010 reg		

IN=Input from		
Fixed port	1110010w	port
Variable port	1110110w	

OUT = Output to		
Fixed port	1110011w	port
Variable port	1110111w	

XLAT=Translate byte to AL	11010111		
LEA=Load EA to register	10001101	mod reg r/m	
LDS=Load pointer to DS	11000101	mod reg r/m	
LES=Load pointer to ES	11000100	mod reg r/m	
LAHF=Load AH with flags	10011111		
SAHF=Store AH into flags	10011110		
PUSHF=Push flags	10011100		
POPF=Pop flags	10011101		

ARITHMETIC

ADD = Add				
Reg /memory with register to either	000000dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w 01
Immediate to accumulator	0000010w	data	data if w 1	

ADC = Add with carry				
Reg /memory with register to either	000100dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w=01
Immediate to accumulator	0001010w	data	data if w 1	

INC = Increment			
Register/memory	1111111w	mod 000 r/m	
Register	01000 reg		
AAA=ASCII adjust for add	00110111		
DAA=Decimal adjust for add	00100111		

SUB = Subtract				
Reg /memory and register to either	001010dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w=01
Immediate from accumulator	0010110w	data	data if w 1	

SBB = Subtract with borrow				
Reg /memory and register to either	000110dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w=01
Immediate from accumulator	0001110w	data	data if w=1	

DEC Decrement	76543210	76543210	76543210	76543210
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG=Change sign	1111011w	mod 011 r/m		

CMP Compare				
Register/memory and register	001110dw	mod reg r/m		
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w 01
Immediate with accumulator	0011110w	data	data if w 1	
AAS ASCII adjust for subtract	00111111			
DAS Decimal adjust for subtract	00101111			
MUL Multiply (unsigned)	1111011w	mod 100 r/m		
IMUL Integer multiply (signed)	1111011w	mod 101 r/m		
AAM ASCII adjust for multiply	11010100	00001010		
DIV Divide (unsigned)	1111011w	mod 110 r/m		
IDIV Integer divide (signed)	1111011w	mod 111 r/m		
AAD ASCII adjust for divide	11010101	00001010		
CBW Convert byte to word	10011000			
CWD Convert word to double word	10011001			

LOGIC

NOT Invert	1111011w	mod 010 r/m	
SHL/SAL Shift logical/arithmetic left	110100vw	mod 100 r/m	
SHR Shift logical right	110100vw	mod 101 r/m	
SAR Shift arithmetic right	110100vw	mod 111 r/m	
ROL Rotate left	110100vw	mod 000 r/m	
ROR Rotate right	110100vw	mod 001 r/m	
RCL Rotate through carry flag left	110100vw	mod 010 r/m	
RCR Rotate through carry right	110100vw	mod 011 r/m	

AND And			
Reg /memory and register to either	001000dw	mod reg r/m	
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w 1
Immediate to accumulator	0010010w	data	data if w 1	

TEST And function to flags, no result				
Register/memory and register	1000010w	mod reg r/m		
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w 1
Immediate data and accumulator	1010100w	data	data if w 1	

OR Or				
Reg /memory and register to either	000010dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w 1
Immediate to accumulator	0000110w	data	data if w=1	

XOR Exclusive or				
Reg /memory and register to either	001100dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w 1
Immediate to'accumulator	0011010w	data	data if w 1	

STRING MANIPULATION

REP=Repeat	1111001z
MOVS=Move byte/word	1010010w
CMPS=Compare byte/word	1010011w
SCAS=Scan byte/word	1010111w
LODS=Load byte/wd to AL/AX	1010110w
STOS=Stor byte/wd from AL/A	1010101w

Table 2. Instruction Set Summary (Continued)

	76543210	76543210	76543210
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	11101000	disp-low	disp-high
Indirect within segment	11111111	mod 010 r/m	
Direct intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 011 r/m	
JMP = Unconditional Jump:			
Direct within segment	11101001	disp low	disp-high
Direct within segment-short	11101011	disp	
Indirect within segment	11111111	mod 100 r/m	
Direct intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 101 r/m	
RET = Return from CALL			
Within segment	11000011		
Within seg adding immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment adding immediate to SP	11001010	data-low	data high
JE/JZ=Jump on equal/zero	01110100	disp	
JL/JNGE=Jump on less/not greater or equal	01111100	disp	
JLE/JNG=Jump on less or equal/not greater	01111110	disp	
JB/JNAE=Jump on below/not above or equal	01110010	disp	
JBE/JNA=Jump on below or equal/ not above	01110110	disp	
JP/JPE=Jump on parity/parity even	01111010	disp	
JO=Jump on overflow	01110000	disp	
JS=Jump on sign	01111000	disp	
JNE/JNZ=Jump on not equal/not zero	01110101	disp	
JNL/JGE=Jump on not less/greater or equal	01111101	disp	
JNLE/JG=Jump on not less or equal/ greater	01111111	disp	

	76543210	76543210
JNB/JAE=Jump on not below/above or equal	01110011	disp
JNBE/JA=Jump on not below or equal/above	01110111	disp
JNP/JPO=Jump on not par/par odd	01111011	disp
JNO=Jump on not overflow	01110001	disp
JNS Jump on not sign	01111001	disp
LOOP Loop CX times	11100010	disp *
LOOPZ/LOOPE Loop while zero/equal	11100001	disp
LOOPNZ/LOOPNE Loop while not zero/equal	11100000	disp
JCXZ Jump on CX zero	11100011	disp
INT Interrupt		
Type specified	11001101	type
Type 3	11001100	
INTO-Interrupt on overflow	11001110	
IRET Interrupt return	11001111	
PROCESSOR CONTROL		
CLC Clear carry	11111000	
CMC Complement carry	11110101	
STC Set carry	11111001	
CLD Clear direction	11111100	
STD Set direction	11111101	
CLI Clear interrupt	11111010	
STI Set interrupt	11111011	
HLT Halt	11110100	
WAIT Wait	10011011	
ESC Escape (to external device)	11011 x x x	mod x x x r/m
LOCK Bus lock prefix	11110000	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive,
Less = less positive (more negative) signed values
if d = 1 then "to" reg, if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high, disp-low

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1, if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with Z.F FLAG

SEGMENT OVERRIDE PREFIX

001 reg 110

REG is assigned according to the following table

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file

FLAGS = X X X X (OF) (DF) (IF) (TF) (SF) (ZF) X (AF) X (PF) X (CF)

iAPX 186
HIGH INTEGRATION 16-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - —Enhanced 8086-2 CPU
  - —Clock Generator
  - —2 Independent, High-Speed DMA Channels
  - —Programmable Interrupt Controller
  - —3 Programmable 16-bit Timers
  - —Programmable Memory and Peripheral Chip-Select Logic
  - —Programmable Wait State Generator
  - —Local Bus Controller
- **Available in 8 MHz (80186) and cost effective 6 MHz (80186-6) versions.**
- **High-Performance Processor**
  - —2 Times the Performance of the Standard iAPX 86
  - —4 MByte/Sec Bus Bandwidth Interface
- **Direct Addressing Capability to 1 MByte of Memory**
- **Completely Object Code Compatible with All Existing iAPX 86, 88 Software**
  - —10 New Instruction Types
- **Complete System Development Support**
  - —Development Software: Assembler, PL/M, Pascal, Fortran, and System Utilities
  - —In-Circuit-Emulator (I²ICE™-186)
  - —iRMX™ 86, 88 Compatible (80130 OSF)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**

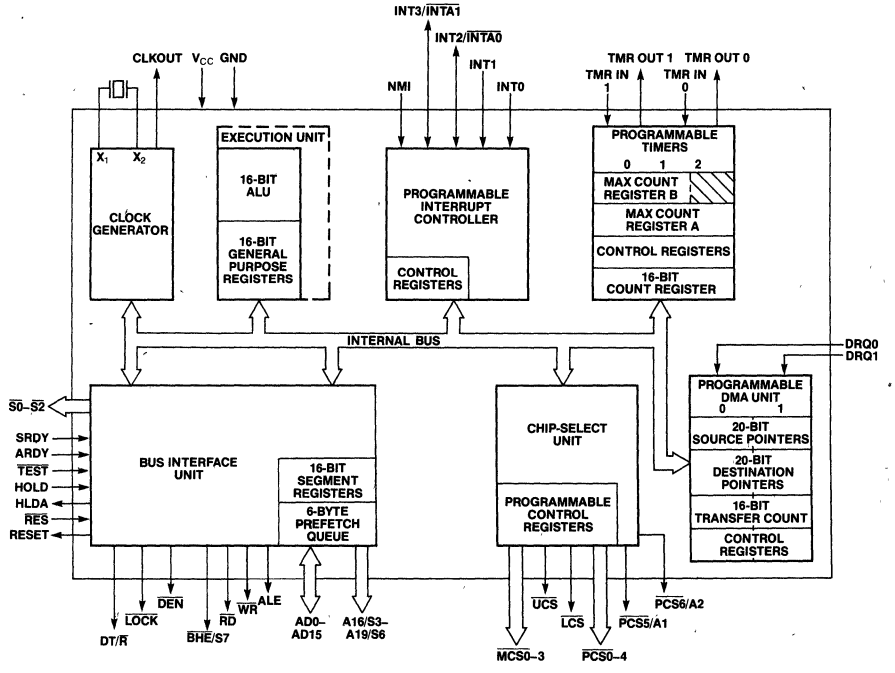


Figure 1. iAPX 186 Block Diagram

JULY 1983
ORDER NUMBER: 210451-003

The Intel iAPX 186 (80186 part number) is a highly integrated 16-bit microprocessor. The iAPX 186 effectively combines 15–20 of the most common iAPX 86 system components onto one. The 80186 provides two times greater throughput than the standard 5 MHz iAPX 86. The iAPX 186 is upward compatible with iAPX 86 and 88 software and adds 10 new instruction types to the existing set.

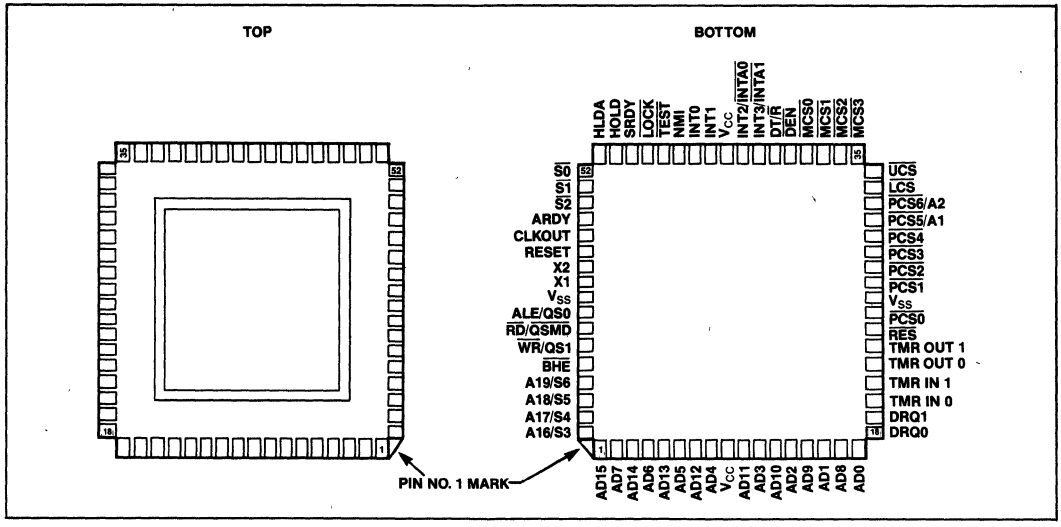


Figure 2. 80186 Pinout Diagram

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	System Power: + 5 volt power supply.
V_{SS}, V_{SS}	26,60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1, X2	59,58	I	Crystal Inputs, X1 and X2, provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{RES}$	24	I	System Reset causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 7 clock cycles after $\overline{RES}$ is returned HIGH. $\overline{RES}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{RES}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then tri-state them.

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{TEST}$	47	I	$\overline{TEST}$ is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80186 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.
INT0, INT1, INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45,44 42 41	I I/O I/O	Maskable Interrupt Requests can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowleged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
A19/S6, A18/S5, A17/S4, A16/S3	65 66 67 68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: S3,S4, and S5 are defined as LOW during T_2–T_4.
AD15–AD0	10–17, 1–8	I/O	Address/Data Bus (0–15) signals constitute the time mutiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH. A_0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D_7 through D_0. It is LOW during T_1 when a byte is to be transferred onto the lower portion of the bus in memory or I/O operations.
$\overline{BHE}$/S7	64	O	During T_1 the Bus High Enable signal should be used to determine if data is to be enabled onto the most significant half of the data bus, pins D_{15}–D_8. $\overline{BHE}$ is LOW during T_1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the higher half of the bus. The S_7 status information is available during T_2, T_3, and T_4. S_7 is logically equivalent to $\overline{BHE}$. The signal is active LOW, and is tristated OFF during bus HOLD.

	Low	High
S6	Processor Cycle	DMA Cycle

$\overline{BHE}$ and A0 Encodings		
$\overline{BHE}$ Value	A0 Value	Function
0	0	Word Transfer
0	1	Byte Transfer on upper half of data bus (D15–D8)
1	0	Byte Transfer on lower half of data bus (D_7–D_0)
1	1	Reserved

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80186 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 8086. The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 8086. Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80186 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction. QS1 | QS0 | Queue Operation 0 | 0 | No queue operation 0 | 1 | First opcode byte fetched from the queue 1 | 1 | Subsequent byte fetched from the queue 1 | 0 | Empty the queue
$\overline{RD}$/$\overline{QSMD}$	62	O	Read Strobe indicates that the 80186 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD." $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism on the $\overline{RD}$ line hols it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80186 should provide ALE, $\overline{WR}$, and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	Asynchronous Ready informs the 80186 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80186. This means that the falling edge of ARDY must be synchronized to the 80186 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle.
SRDY	49	I	Synchronous Ready must be synchronized externally to the 80186. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied LOW.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the LOCK prefix. It remains active until the completion of the instruction following the LOCK prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated. If unused, this line should be tied LOW.

Table 1. 80186 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{S0}$,$\overline{S1}$,$\overline{S2}$	52–54	O	Bus cycle status $\overline{S0}$–$\overline{S2}$ are encoded to provide bus-transaction information: 80186 Bus Cycle Status Information $\overline{S2}$ / $\overline{S1}$ / $\overline{S0}$ / Bus Cycle Initiated 0 0 0 Interrupt Acknowledge 0 0 1 Read I/O 0 1 0 Write I/O 0 1 1 Halt 1 0 0 Instruction Fetch 1 0 1 Read Data from Memory 1 1 0 Write Data to Memory 1 1 1 Passive (no bus cycle) The status pins float during "HOLD." $\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator. The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.
HOLD (input) HLDA (output)	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80186 clock. The 80186 will issue a HLDA (HIGH) in response to a HOLD request at the end of T_4 or T_1. Simultaneous with the issuance of HLDA, the 80186 will float the local bus and control lines. After HOLD is detected as being LOW, the 80186 will lower HLDA. When the 80186 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K–256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K–256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$–3	38,37,36,35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K–512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$–3 are software programmable.
$\overline{PCS0}$ $\overline{PCS1}$-4	25 27,28,29,30	O O	Peripheral Chip Select signals 0–4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$–4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80186. When HIGH the 80186 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the iAPX 186. This architecture is common to the iAPX 86, 88, and 286 microprocessor families as well. The iAPX 186 is a very high integration 16-bit microprocessor. It combines 15–20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard iAPX 86. The 80186 is object code compatible with the iAPX 86, 88 microprocessors and adds 10 new instruction types to the existing iAPX 86, 88 instruction set.

iAPX 186 BASE ARCHITECTURE

The iAPX 86, 88, 186, and 286 family all contain the same basic set of registers, instructions, and addressing modes. The 80186 processor is upward compatible with the 8086, 8088, and 80286 CPUs.

Register Set

The 80186 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers
Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers
Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers
Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers
Two 16-bit special purpose registers record or alter certain aspects of the 80186 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80186 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

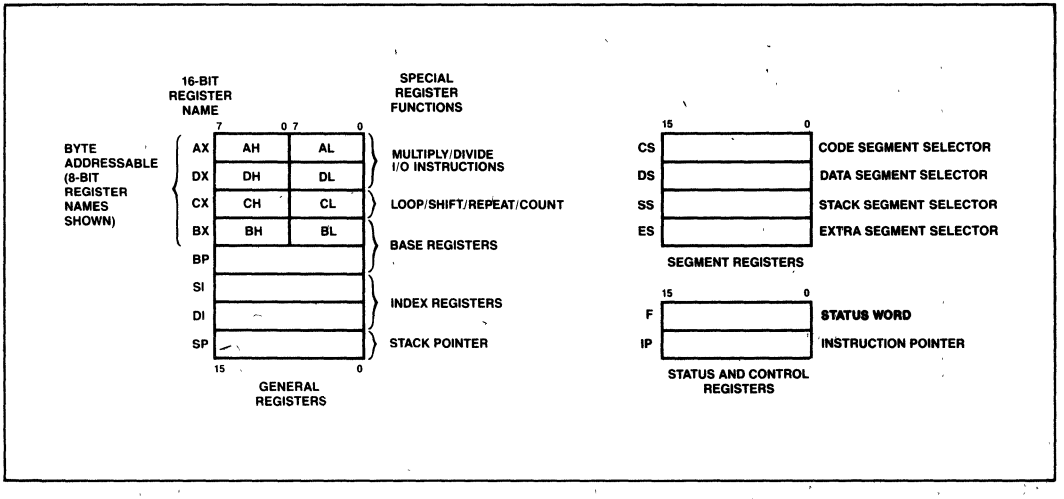


Figure 3a. 80186 General Purpose Register Set

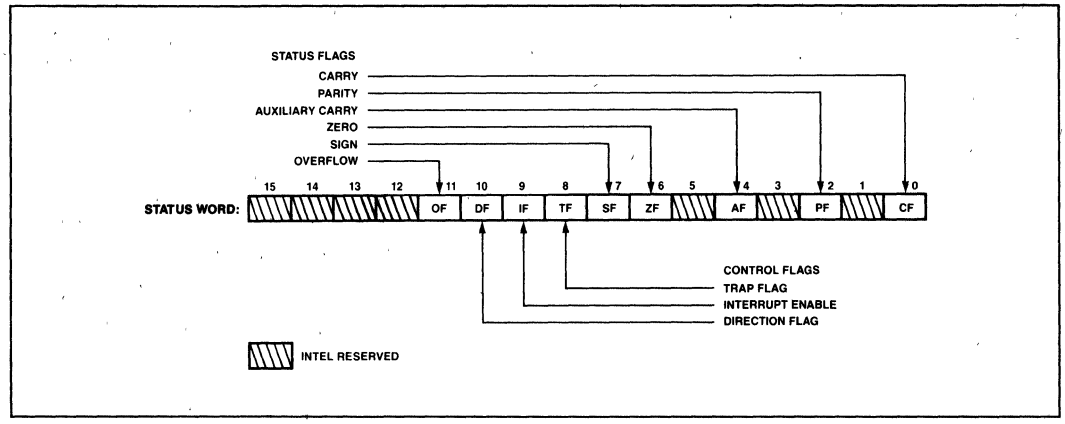


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag–Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80186 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. iAPX 186 Instruction Set

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	**ITERATION CONTROLS**	
JE/JZ	Jump if equal/zero		
JG/JNLE	Jump if greater/not less nor equal	LOOP	Loop
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal/not zero	**INTERRUPTS**	
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4. iAPX 186 Instruction Set (continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

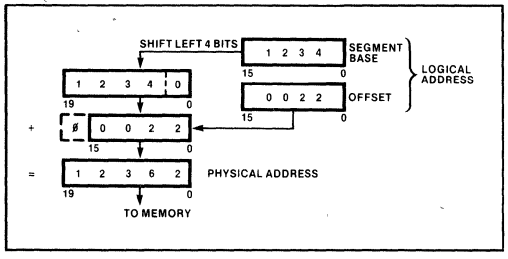


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

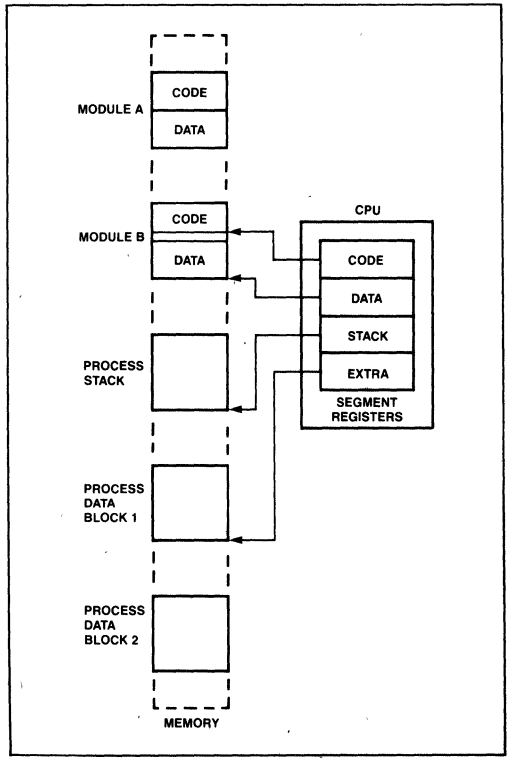


Figure 6. Segmented Memory Helps Structure Software

Addressing Modes

The 80186 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicity chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80186 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the iAPX 186/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the iAPX 186/20 Numeric Data Processor configuration.)

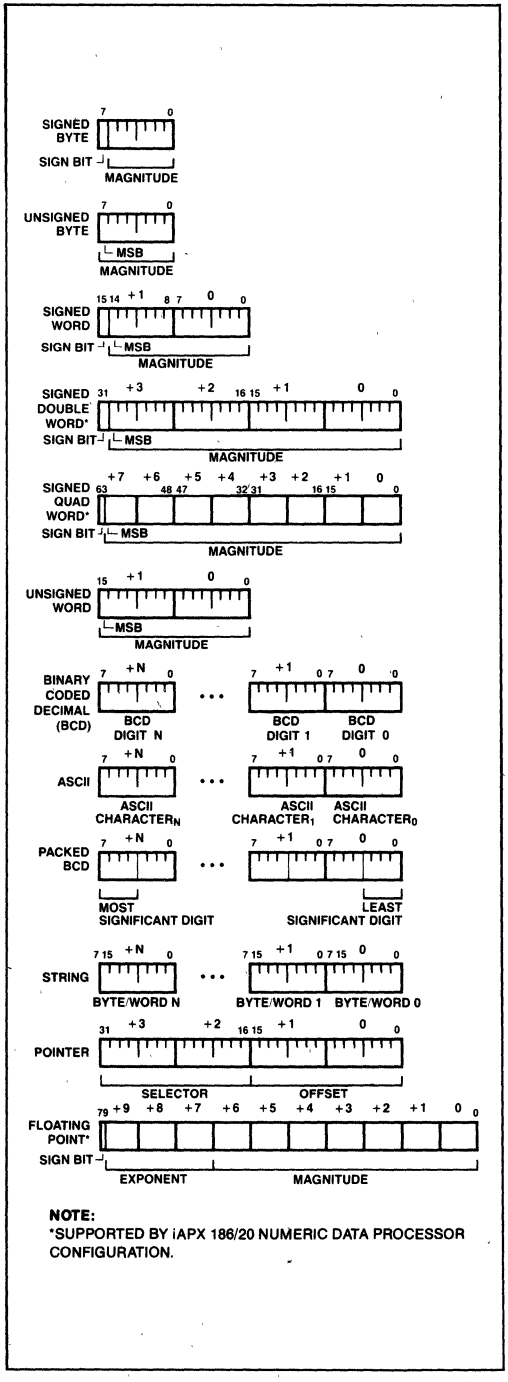
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the iAPX 186.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}-A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*SUPPORTED BY iAPX 186/20 NUMERIC DATA PROCESSOR CONFIGURATION.

Figure 7. iAPX 186 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80186 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80186 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and non-cascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80186 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80186 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)
Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

Table 4. 80186 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INT0 Detected Overflow Exception	4	*1	INT0
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:

*1. These are generated as the result of an instruction execution.

**2. This is handled as in the 8086.

****3. All three timers constitute one source of request to the interrupt controller. The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves. (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number.

4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level.

***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

SINGLE-STEP INTERRUPT (TYPE 1)
Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)
An external interrupt source which cannot be masked.

BREAKPOINT INTERRUPT (TYPE 3)
A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE 4)
Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)
Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)
Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)
Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80186 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80186 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80186 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input

causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80186 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80186 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80186 begins execution with the instruction at physical location FFFF0(H). $\overline{RES}$ also sets some registers to predefined values as shown in Table 5.

Table 5. 80186 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

iAPX 186 CLOCK GENERATOR

The iAPX 186 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the iAPX 186 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the iAPX 186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the iAPX 186. The recommended crystal configuration is shown in Figure 8.

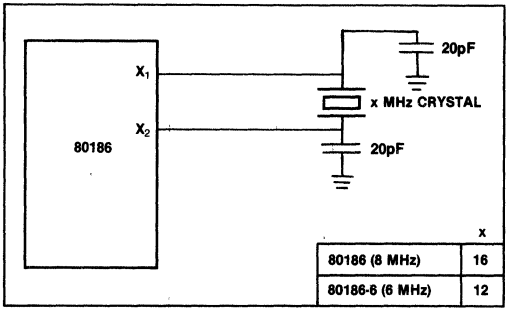


Figure 8. Recommended iAPX 186 Crystal Configuration

Clock Generator

The iAPX 186 clock generator provides the 50% duty cycle processor clock for the iAPX 186. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the iAPX 186. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The iAPX 186 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only on the rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3 or T_W. High-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the iAPX 186, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The iAPX 186 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the iAPX 186 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple iAPX 186 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80186 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

LOCAL BUS CONTROLLER

The iAPX 186 provides a local bus controller to generate the local bus control signals. In addition, it employs a HOLD/HLDA protocol for relinquishing the local bus to other bus masters. It also provides control lines that can be used to enable external buffers and to direct the flow of data on and off the local bus.

Memory/Peripheral Control

The iAPX 186 provides ALE, $\overline{RD}$, and $\overline{WR}$ bus control signals. The $\overline{RD}$ and $\overline{WR}$ signals are used to strobe data from memory to the iAPX 186 or to strobe data from the iAPX 186 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The iAPX 186 local bus controller does not provide a memory/$\overline{I/O}$ signal. If this is required, the user will have to use the $\overline{S2}$ signal (which will require external latching), make the memory and I/O spaces nonoverlapping, or use only the integrated chip-select circuitry.

Transceiver Control

The iAPX 186 generates two control signals to be connected to 8286/8287 transceiver chips. This capability allows the addition of transceivers for extra buffering without adding external logic. These control lines, DT/$\overline{R}$ and $\overline{DEN}$, are generated to control the flow of data through the transceivers. The operation of these signals is shown in Table 6.

Table 6. Transceiver Control Signals Description

Pin Name	Function
$\overline{DEN}$ (Data Enable)	Enables the output drivers of the transceivers. It is active LOW during memory, I/O, or INTA cycles.
DT/$\overline{R}$ (Data Transmit/ Receive)	Determines the direction of travel through the transceivers. A HIGH level directs data away from the processor during write operations, while a LOW level directs data toward the processor during a read operation.

Local Bus Arbitration

The iAPX 186 uses a HOLD/HLDA system of local bus exchange. This provides an asynchronous bus exchange mechanism. This means multiple masters utilizing the same bus can operate at separate clock frequencies. The iAPX 186 provides a single HOLD/HLDA pair through which all other bus masters may gain control of the local bus. This requires external circuitry to arbitrate which external device will gain control of the bus from the iAPX 186 when there is more than one alternate local bus master. When the iAPX 186 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$–$\overline{S2}$, $\overline{LOCK}$, AD0–AD15, A16–A19, $\overline{BHE}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The iAPX 186 HOLD latency time, i.e., the time between HOLD request and HOLD acknowledge, is a function of the activity occurring in the processor when the HOLD request is received. A HOLD request is the highest-priority activity request which the processor may receive: higher than instruction fetching or internal DMA cycles. However, if a DMA cycle is in progress, the iAPX 186 will complete the transfer before relinquishing the bus. This implies that if a HOLD request is received just as a DMA transfer begins, the HOLD latency time can be as great as 4 bus cycles. This will occur if a DMA word transfer operation is taking place from an odd address to an odd address. This is a total of 16 clocks or more, if WAIT states are required. In addition, if locked transfers are performed, the HOLD latency time will be increased by the length of the locked transfer.

Local Bus Controller and Reset

Upon receipt of a RESET pulse from the $\overline{RES}$ input, the local bus controller will perform the following actions:

- Drive $\overline{DEN}$, $\overline{RD}$, and $\overline{WR}$ HIGH for one clock cycle, then float.

  NOTE: $\overline{RD}$ is also provided with an internal pull-up device to prevent the processor from inadvertently entering Queue Status mode during reset.

- Drive $\overline{S0}$–$\overline{S2}$ to the passive state (all HIGH) and then float.
- Drive $\overline{LOCK}$ HIGH and then float.
- Tristate AD0–15, A16–19, $\overline{BHE}$, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

INTERNAL PERIPHERAL INTERFACE

All the iAPX 186 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{15-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80186 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet). In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated iAPX 186 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control and data locations in the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed.

CHIP-SELECT/READY GENERATION LOGIC

The iAPX 186 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The iAPX 186 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes, whereas iAPX 186 memory is arranged in words. This means that if, for example, 16 64K x 1 memories are used, the memory block size will be 128K, not 64K.

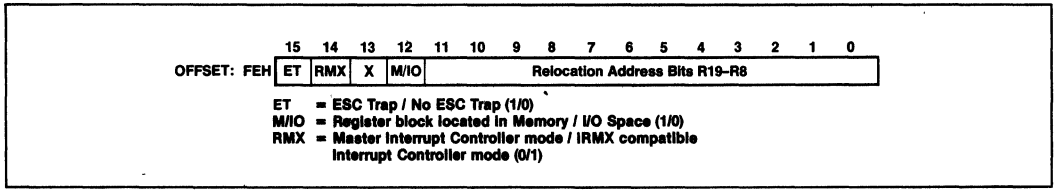


Figure 9. Relocation Register

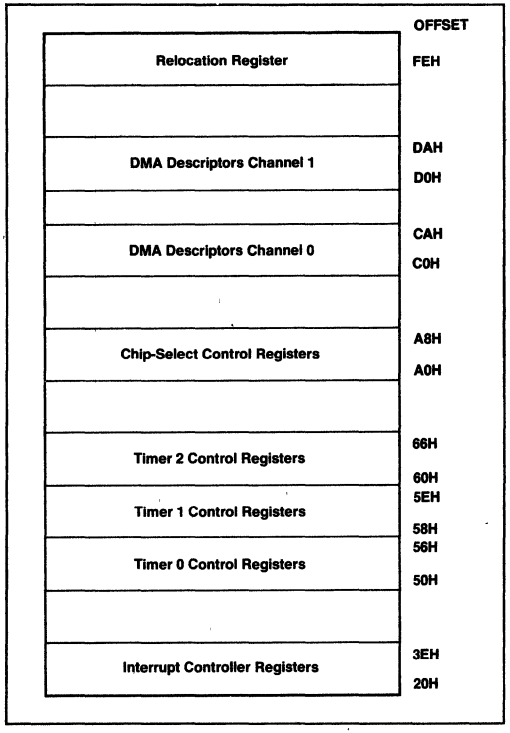


Figure 10. Internal Register Map

Upper Memory $\overline{CS}$

The iAPX 186 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the iAPX 186 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The iAPX 186 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{LCS}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The iAPX 186 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the iAPX 186 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as descibed a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14–8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

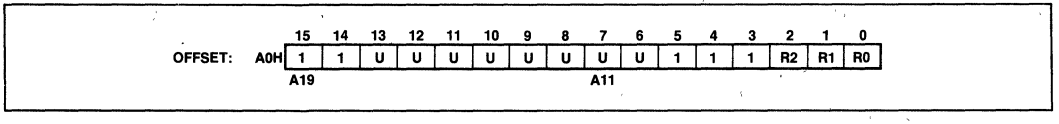


Figure 11. UMCS Register

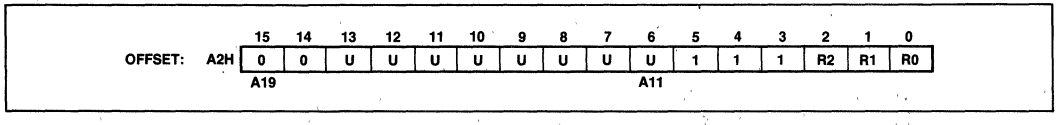


Figure 12. LMCS Register

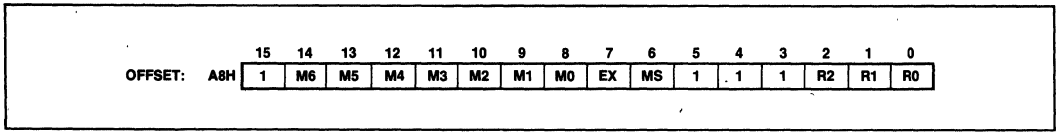


Figure 13. MPCS Register

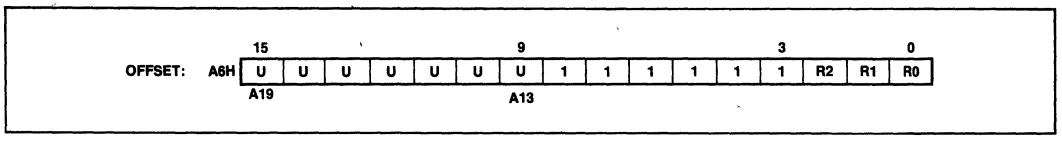


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The iAPX 186 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the iAPX 186. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

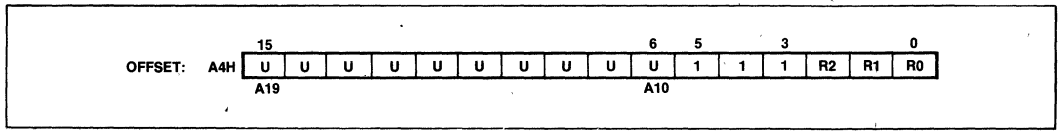


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA —PBA+127
PCS1	PBA+128 —PBA+255
PCS2	PBA+256 —PBA+383
PCS3	PBA+384 —PBA+511
PCS4	PBA+512 —PBA+639
PCS5	PBA+640 —PBA+767
PCS6	PBA+768 —PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided. 1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The iAPX 186 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the iAPX 186 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the iAPX 186. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to

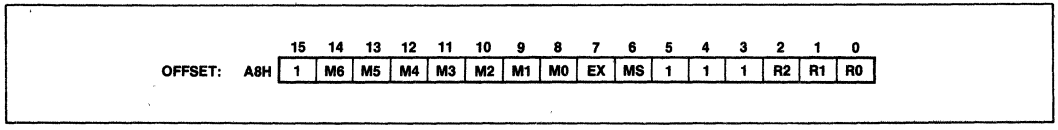


Figure 16. MPCS Register

allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).

- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

The 80186 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Data can be transferred either in bytes (8 bits) or in words (16 bits) to or from even or odd addresses. Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer (by one or two depending on byte or word transfers). Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of one Mword/sec or 2 MBytes/sec.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte or word transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

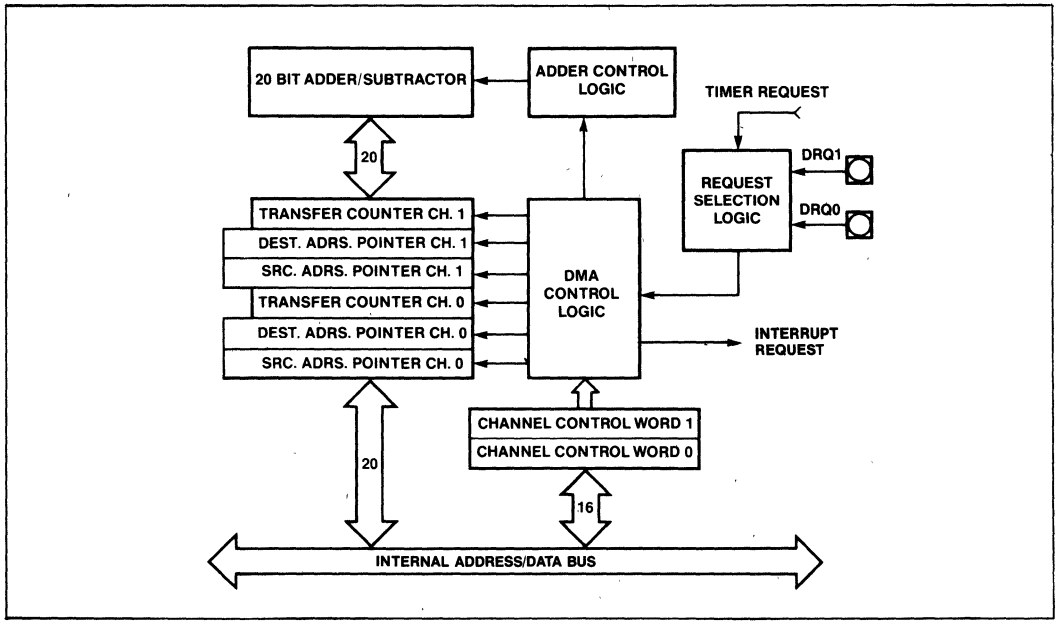


Figure 17. DMA Unit Block Diagram

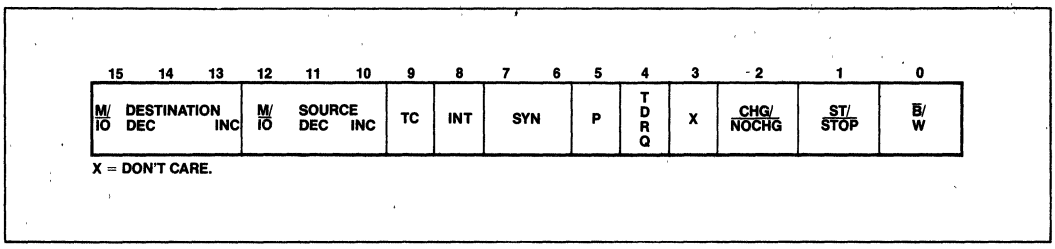


Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80186 DMA channel. This register specifies:

- the mode of synchronization;
- whether bytes or words will be transferred;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

$\overline{B}$/W: Byte/Word (0/1) Transfers.

ST/$\overline{STOP}$: Start/stop (1/0) Channel.

CHG/$\overline{NOCHG}$: Change/Do not change (1/0) ST/$\overline{STOP}$ bit. If this bit is set when writing to the control word, the ST/$\overline{STOP}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{STOP}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on Transfer Count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{STOP}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN: (2 bits)
- 00 No synchronization.
  NOTE: The ST bit will be cleared automatically when the contents of the TC register reach zero regardless of the state of the TC bit.
- 01 Source synchronization.
- 10 Destination synchronization.
- 11 Unused.

SOURCE:INC Increment source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

M/$\overline{IO}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on $\overline{B}$/W) after each transfer.

DEST: INC Increment destination pointer by 1 or 2 ($\overline{B}$/W) after each transfer.

M/$\overline{IO}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depending on $\overline{B}$/W) after each transfer.

P Channel priority—relative to other channel.
- 0 low priority.
- 1 high priority.

Channels will alternate cycles if both set at same priority level.

TDRQ	0: Disable DMA requests from timer 2. 1: Enable DMA requests from timer 2.
Bit 3	Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. If word transfers are performed the pointer is incremented or decremented by two. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers. Higher transfer rates can be obtained if all word transfers are performed to even addresses, since this will allow data to be accessed in a single memory access.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit in the DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, however, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	2MBytes/sec	2MBytes/sec
Source Synch	2MBytes/sec	2MBytes/sec
Destination Synch	1.3MBytes/sec	1.5MBytes/sec

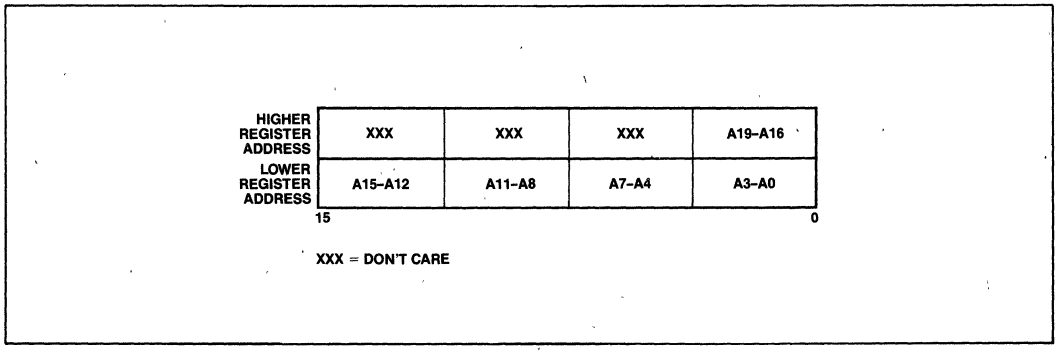


Figure 18a. DMA Memory Pointer Register Format

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80186 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

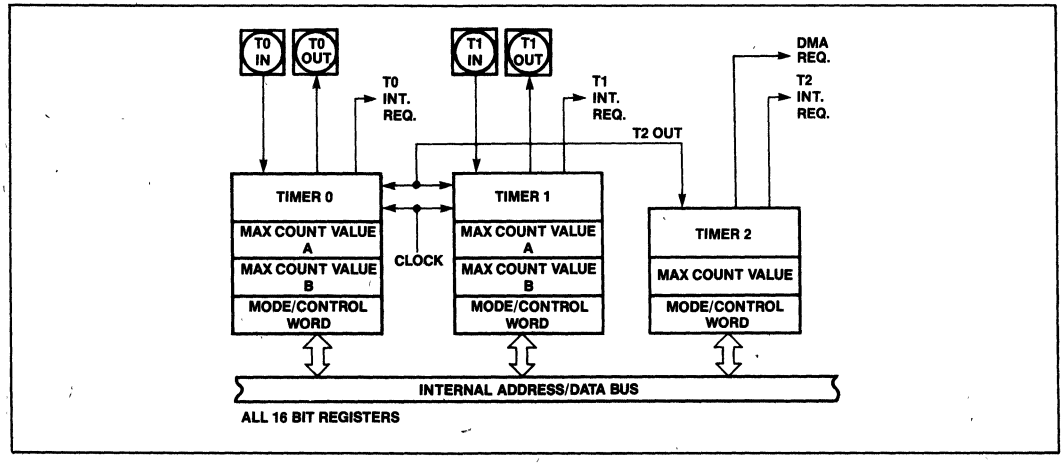


Figure 19. Timer Block Diagram

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

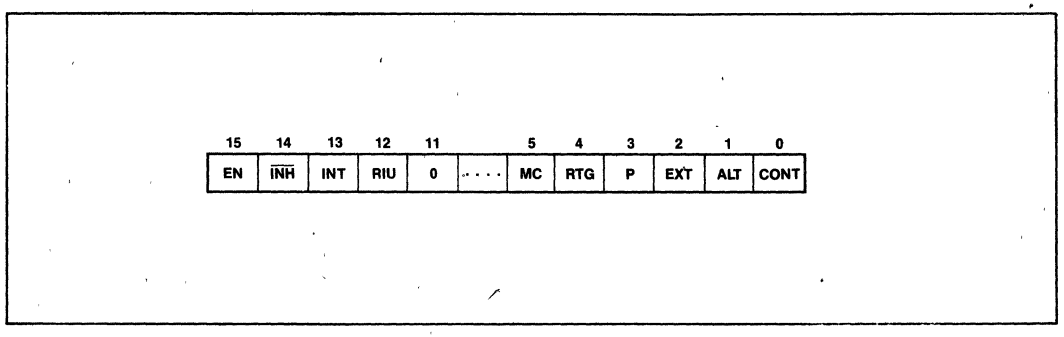


Figure 20. Timer Mode/Control Register

ALT:
The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:
Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT =1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:
The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80186 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:
The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:
Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80186 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:
The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH:
The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:
When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

MC:
The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RIU:
The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80186 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU. Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80186 interrupt controller has its own control registers that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80186 within the iRMX 86 operating system interrupt structure. The controller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80186 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80186 interrupt controller in iRMX 86 mode.

MASTER MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge ouput lines. When the interrupt lines are configured in cascade mode, the 80186 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80186 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in master mode are similar to the 8259A. The interrupt controller responds identically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the inservice bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80186 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80186 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

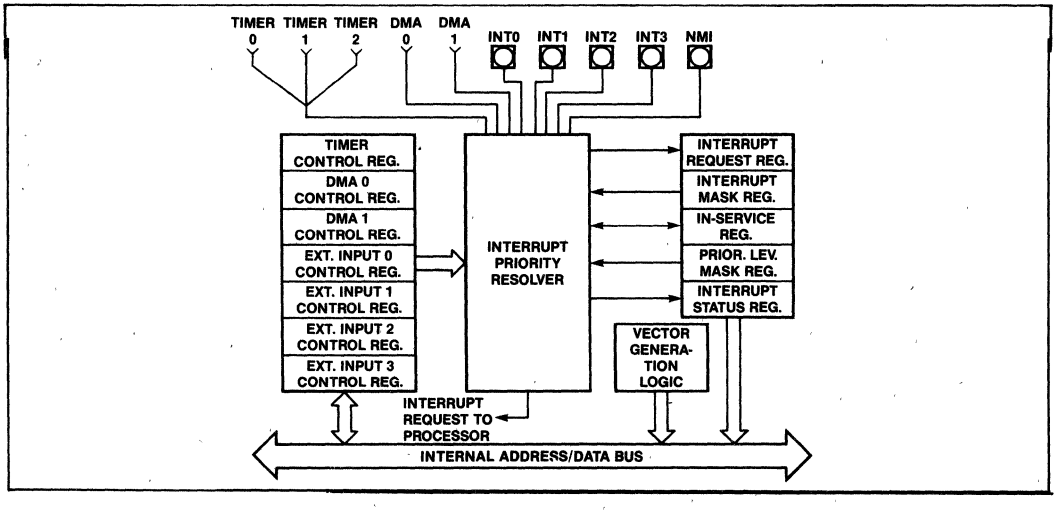


Figure 21. Interrupt Controller Block Diagram

Special Fully Nested Mode
This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80186 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80186 controller until the 80186 in-service bit is reset. In special fully nested mode, the 80186 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80186 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80186 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enough priority is requesting service. Bits 0-4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest-priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80186 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Master Mode Features

Programmable Priority
The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority.) All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command
The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode
The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80186 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring
The 80186 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register
This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register
The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Mask Register
This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

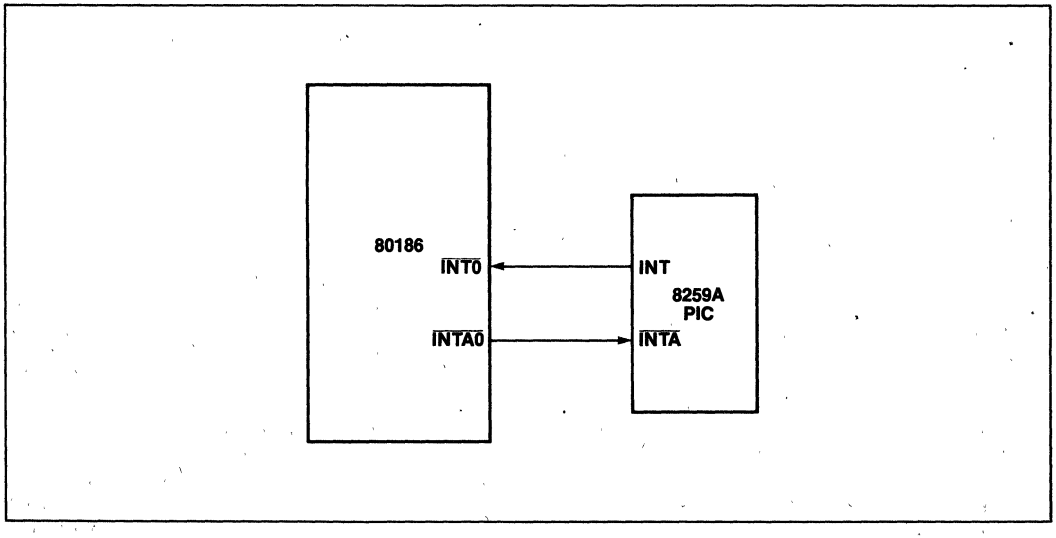


Figure 22. Cascade Mode Interrupt Connection

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX 86 Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

15	14	• • •	10	9	8	7	6	5	4	3	2	1	0
0	0		0	0	0	I3	I2	I1	I0	D1	D0	0	TMR

Figure 24. In-Service, Interrupt Request, and Mask Register Formats

15	14	• • • • • • • • • •	3	2	1	0
0	0		0	PRM2	PRM1	PRM0

Figure 25. Priority Mask Register Format

15	14	• • • • •	7	6	5	4	3	2	1	0
DHLT	0		0	0	0	0	0	IRT2	IRT1	IRT0

Figure 26. Interrupt Status Register Format

Timer, DMA 0, 1; Control Registers
These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0–INT3 Control Registers
These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest Priority = 000, Lowest Priority = 111

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = nonmask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register
The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80186 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

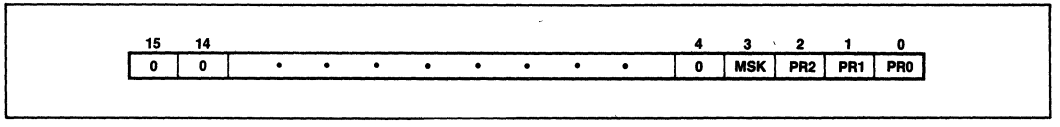


Figure 27. Timer/DMA Control Register Formats

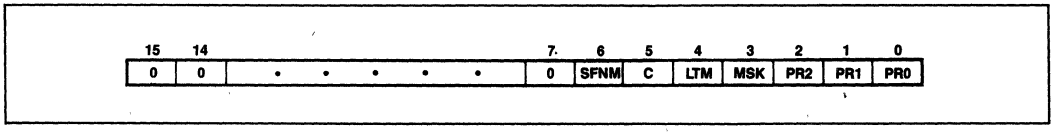


Figure 28. INT0/INT1 Control Register Formats

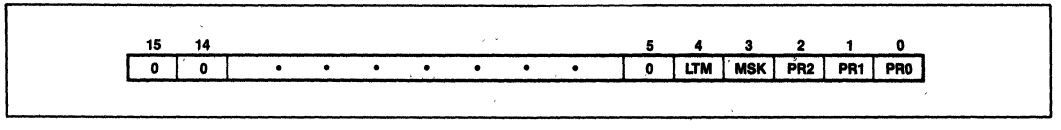


Figure 29. INT2/INT3 Control Register Formats

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_x: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX 86 COMPATIBILITY MODE

This mode allows iRMX 86-80186 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80186 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80186 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80186 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80186 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80186 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX 86 Mode External Interface

The configuration of the 80186 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80186 CPU interrupt input. INT3 functions as an output to send the 80186 slave-interrupt-request to one of the 8 master-PIC-inputs.

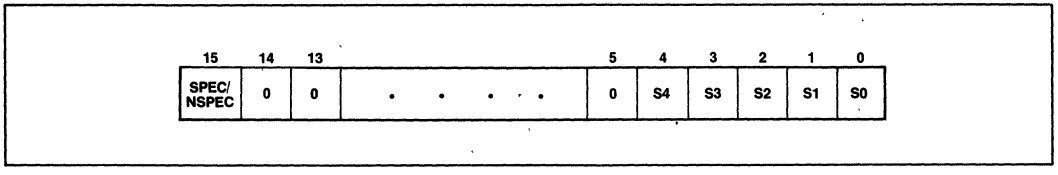


Figure 30. EOI Register Format

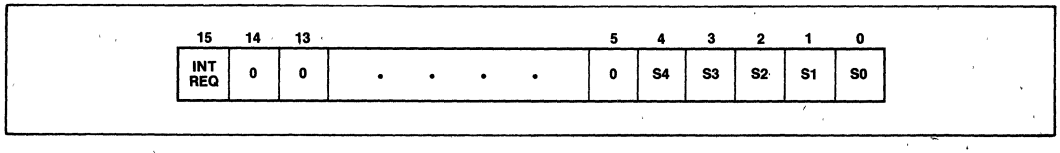


Figure 31. Poll Register Format

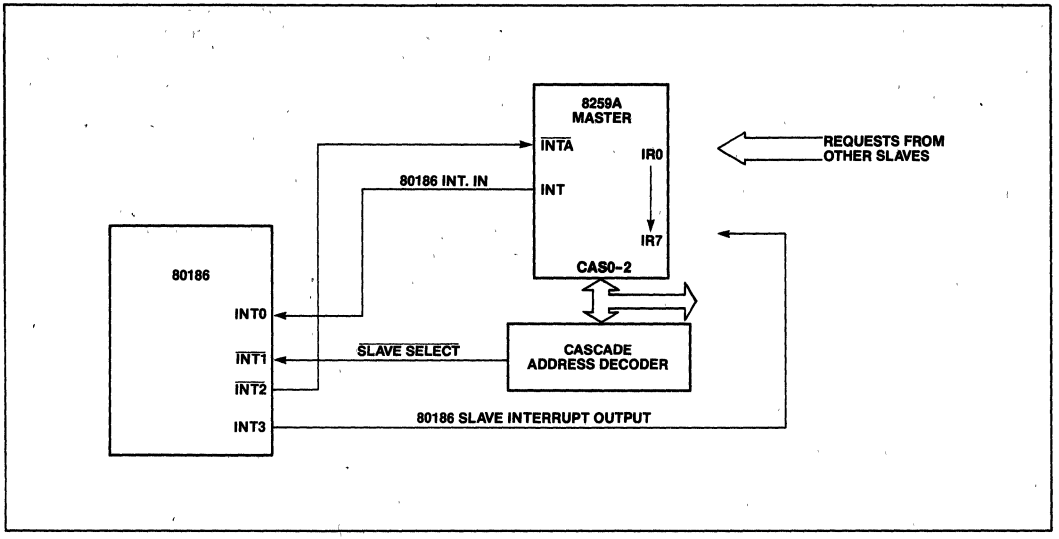


Figure 32. iRMX 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0–2). Slave 8259As do this internally. Because of pin limitations, the 80186 slave address will have to be decoded externally. $\overline{INT1}$ is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80186 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal

interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register
This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

Mask Register
This register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers
These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

	OFFSET
LEVEL 5 CONTROL REGISTER (TIMER 2)	3AH
LEVEL 4 CONTROL REGISTER (TIMER 1)	38H
LEVEL 3 CONTROL REGISTER (DMA 1)	36H
LEVEL 2 CONTROL REGISTER (DMA 0)	34H
LEVEL 0 CONTROL REGISTER (TIMER 0)	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT-REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY-LEVEL MASK REGISTER	2AH
MASK REGISTER	28H
SPECIFIC EOI REGISTER	22H
INTERRUPT VECTOR REGISTER	20H

Figure 33. Interrupt Controller Registers (iRMX 86 Mode)

15	14	13	. . . . .	8	7	6	5	4	3	2	1	0
0	0	0		0	0	0	0	0	0	L2	L1	L0

Figure 34. Specific EOI Register Format

15	14	13	. . . . .	8	7	6	5	4	3	2	1	0
0	0	0		0	0	0	TMR2	TMR1	D1	D0	0	TMR0

Figure 35. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register
This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.
The format of the bits in this register is:
t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register
This register indicates the lowest priority-level interrupt which will be serviced.
The encoding of the bits in this register is:
m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register
This register is defined exactly as in Non-iRMX Mode. (See Fig. 26.)

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

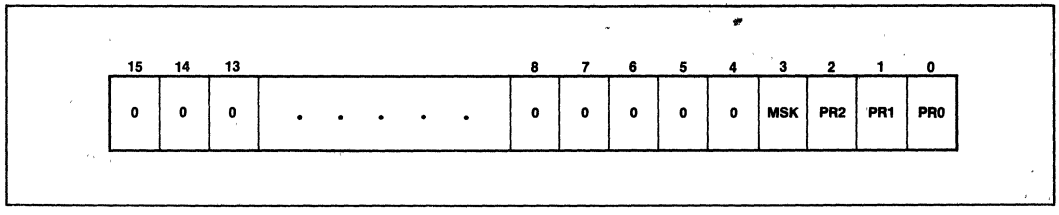


Figure 36. Control Word Format

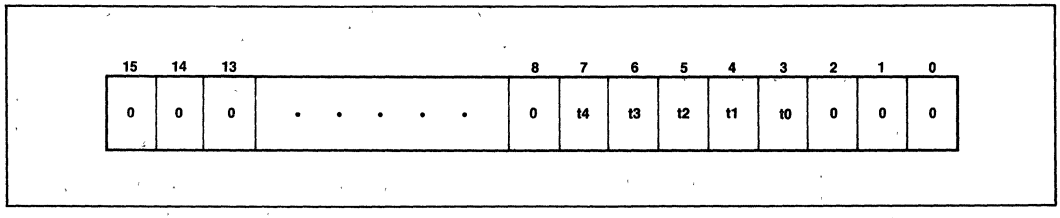


Figure 37. Interrupt Vector Register Format

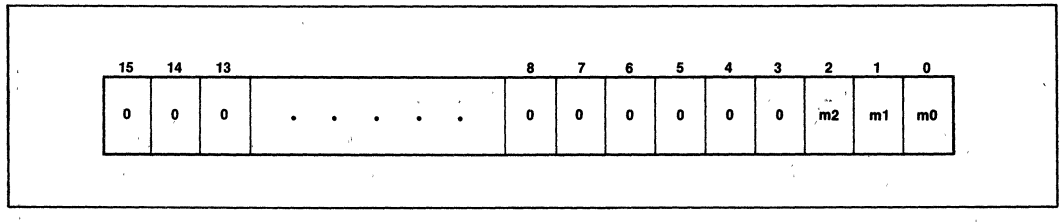


Figure 38. Priority Level Mask Register

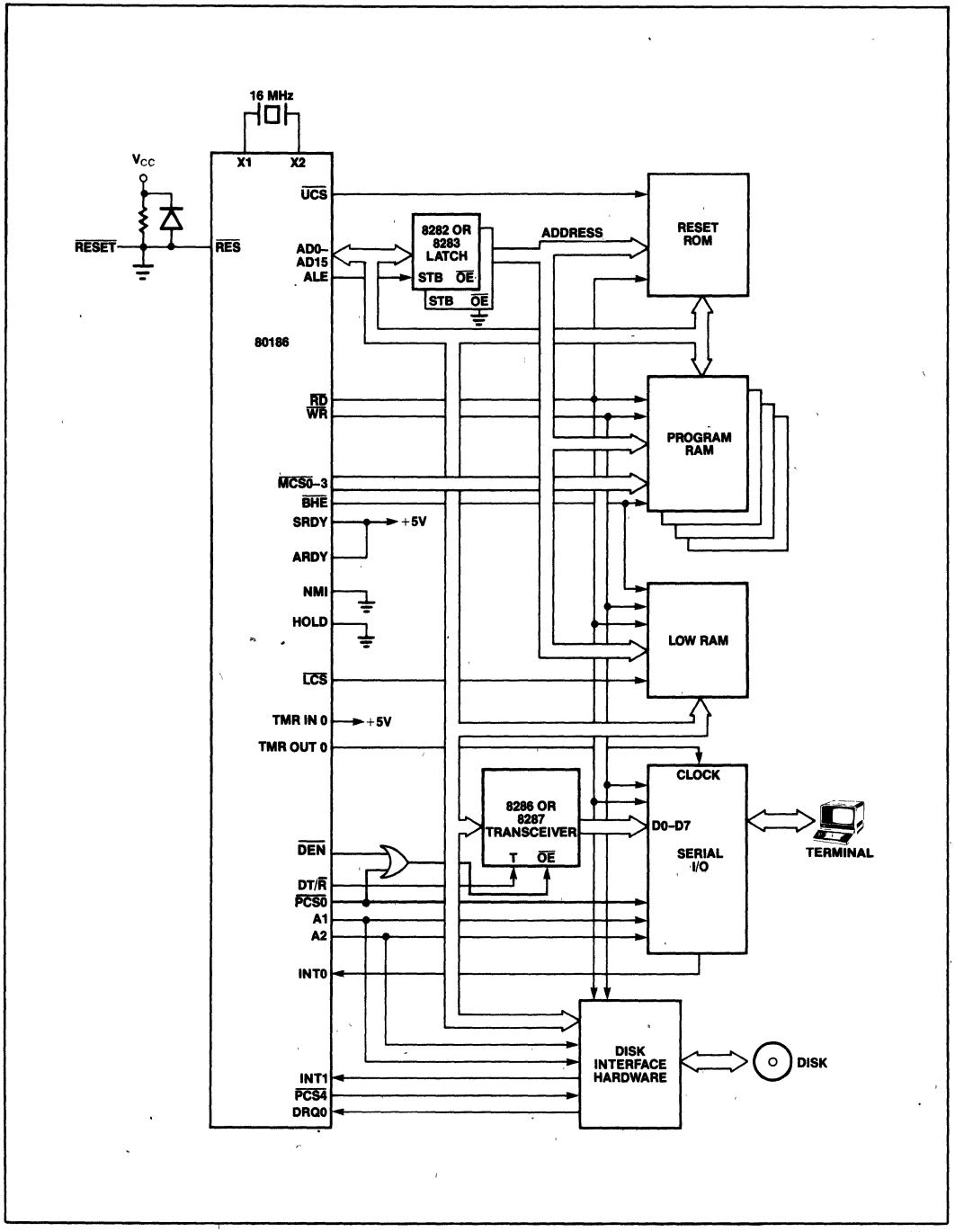


Figure 39. Typical iAPX 186 Computer

AFN-02217C

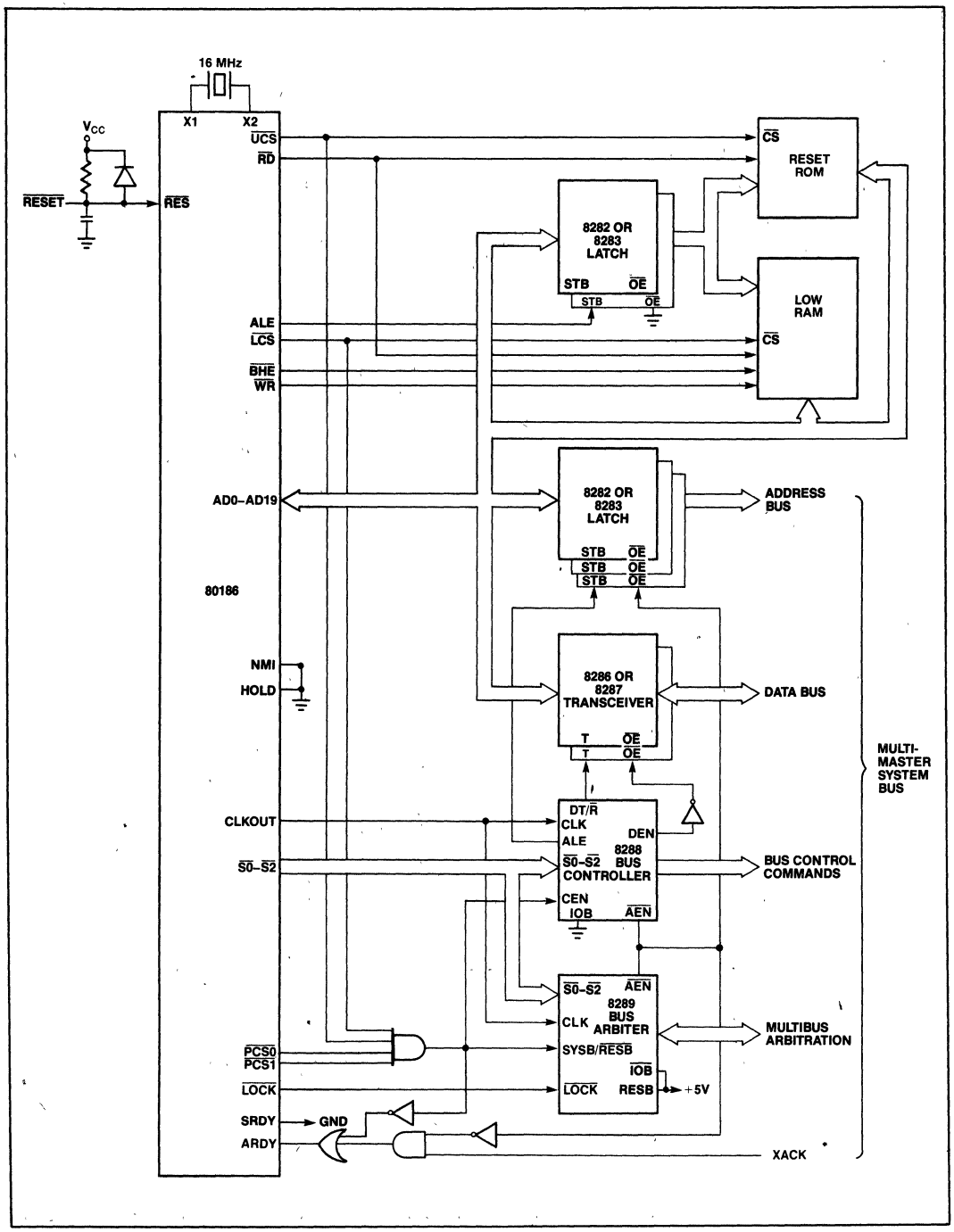


Figure 40. Typical iAPX 186 Multi-Master Bus Interface

PACKAGE

The 80186 is housed in a 68-pin, leadless JEDEC type A hermetic chip carrier. Figure 41 illustrates the package dimensions.

NOTE: The IDT 3M Textool 68-pin JEDEC Socket is required for I²ICE™-186 operation See Figure 42 for details.

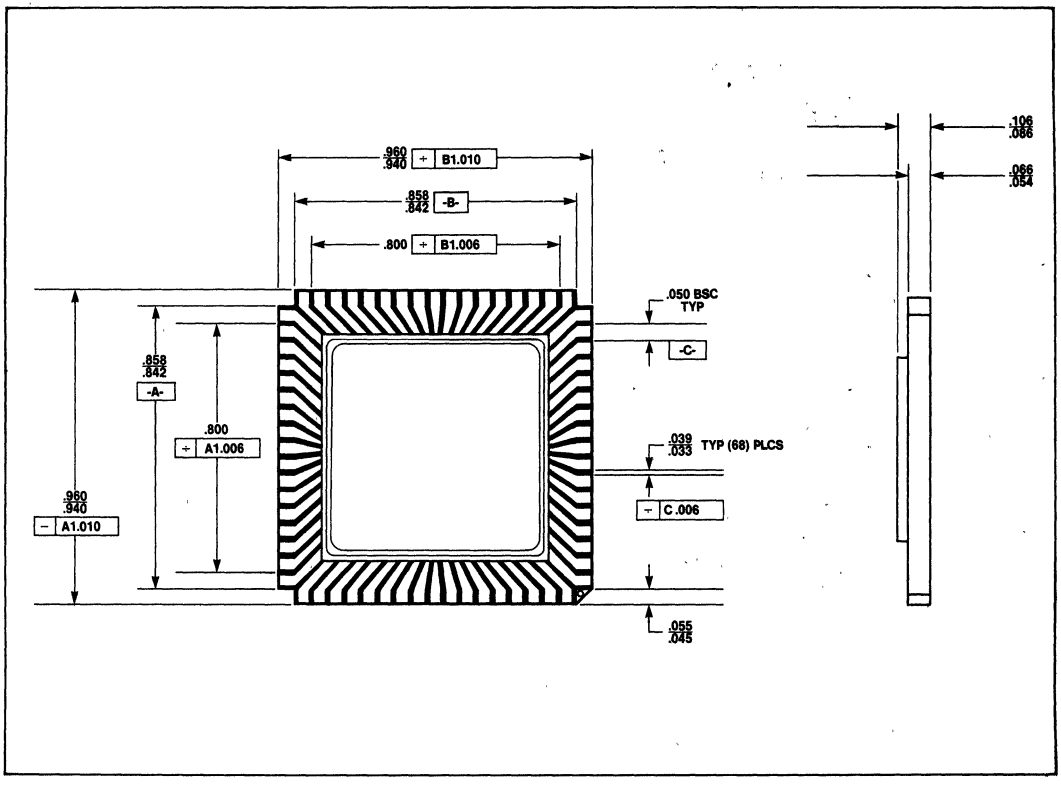
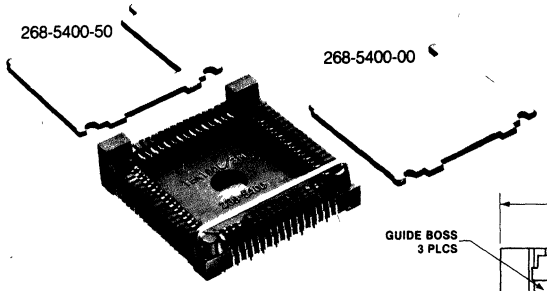
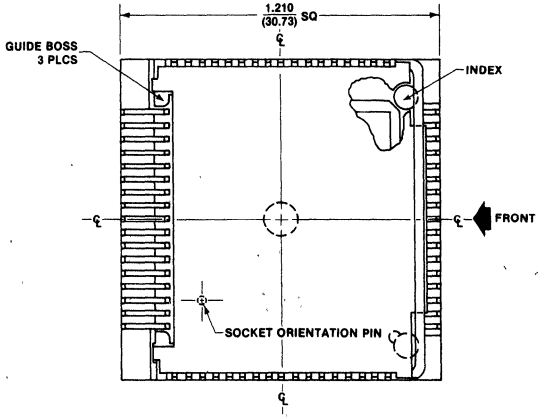
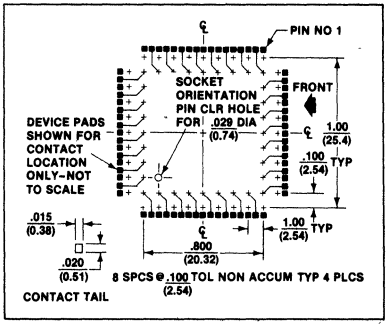


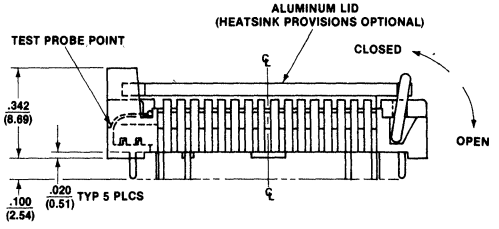
Figure 41. 80186 JEDEC Type A Package





PC BOARD PATTERN





NOTE: Physical dimensions shown are for reference only. Please consult 3M Textool for complete information on the socket.

Figure 42. Textool 68 Lead Chip Carrier Socket

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 3 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°–70°C, V_{CC} = 5V ±10%)
Applicable to 80186 (8 MHz) and 80186-6 (6 MHz)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	Volts	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	Volts	
V_{IH1}	Input High Voltage ($\overline{RES}$)		V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage	3.0	0.45	Volts	I_a = 2.5 mA for $\overline{S0}$-$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage		2.4	Volts	I_{oa} = −400 μA
I_{CC}	Power Supply Current		550 450	mA	Max measured at T_A = 0°C T_A = 70°C
I_{LI}	Input Leakage Current		±10	μA	0V < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	Volts	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		Volts	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	Volts	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	Volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°–70°C, V_{CC} = 5V ± 10%)
80186 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted.
Applicable to 80186 (8 MHz) and 80186-6 (6 MHz)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TDVCL	Data in Setup (A/D)	20		ns	
TCLDX	Data in Hold (A/D)	10		ns	
TARYHCH	Asynchronous Ready (AREADY) active setup time*	20		ns	
TARYLCL	AREADY inactive setup time	35		ns	
TCHARYX	AREADY hold time	15		ns	
TSRYCL	Synchronous Ready (SREADY) transition setup time	35		ns	
TCLSRY	SREADY transition hold time	15		ns	
THVCL	HOLD Setup*	25		ns	
TINVCH	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		ns	
TINVCL	DRQ0, DRQ1, Setup*	25		ns	

*To guarantee recognition at next clock.

A.C. CHARACTERISTICS (Continued)

80186 Master Interface Timing Responses

		80188 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLAV}	Address Valid Delay	5	44	5	63	ns	C_L = 20-200 pF all outputs
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	44	ns	
T_{CHCZ}	Command Lines Float Delay		45		56	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		76	ns	
T_{LHLL}	ALE Width	$T_{CLCL-35}$		$T_{CLCL-35}$		ns	
T_{CHLH}	ALE Active Delay		35		44	ns	
T_{CHLL}	ALE Inactive Delay		35		44	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL-25}$		$T_{CHCL-30}$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	55	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CVCTV}	Control Active Delay 1	5	70	5	87	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	76	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	76	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)		70		87	ns	
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	87	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	76	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CLHAV}	HLDA Valid Delay	10	50	10	67	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL-50}$		$2T_{CLCL-50}$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL-40}$		$2T_{CLCL-40}$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH-25}$		$T_{CLCH-45}$		ns	
T_{CHSV}	Status Active Delay	10	55	10	76	ns	
T_{CLSH}	Status Inactive Delay	10	55	10	76	ns	
T_{CLTMV}	Timer Output Delay		60		75	ns	100 pF max
T_{CLRO}	Reset Delay		60		75	ns	
T_{CHQSV}	Queue Status Delay		35		44	ns	

80186 Chip-Select Timing Responses

Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
T_{CLCSV}	Chip-Select Active Delay		66		80	ns	
T_{CXCSX}	Chip-Selct Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	47	ns	

A.C. CHARACTERISTICS (Continued)

80186 CLKIN Requirements

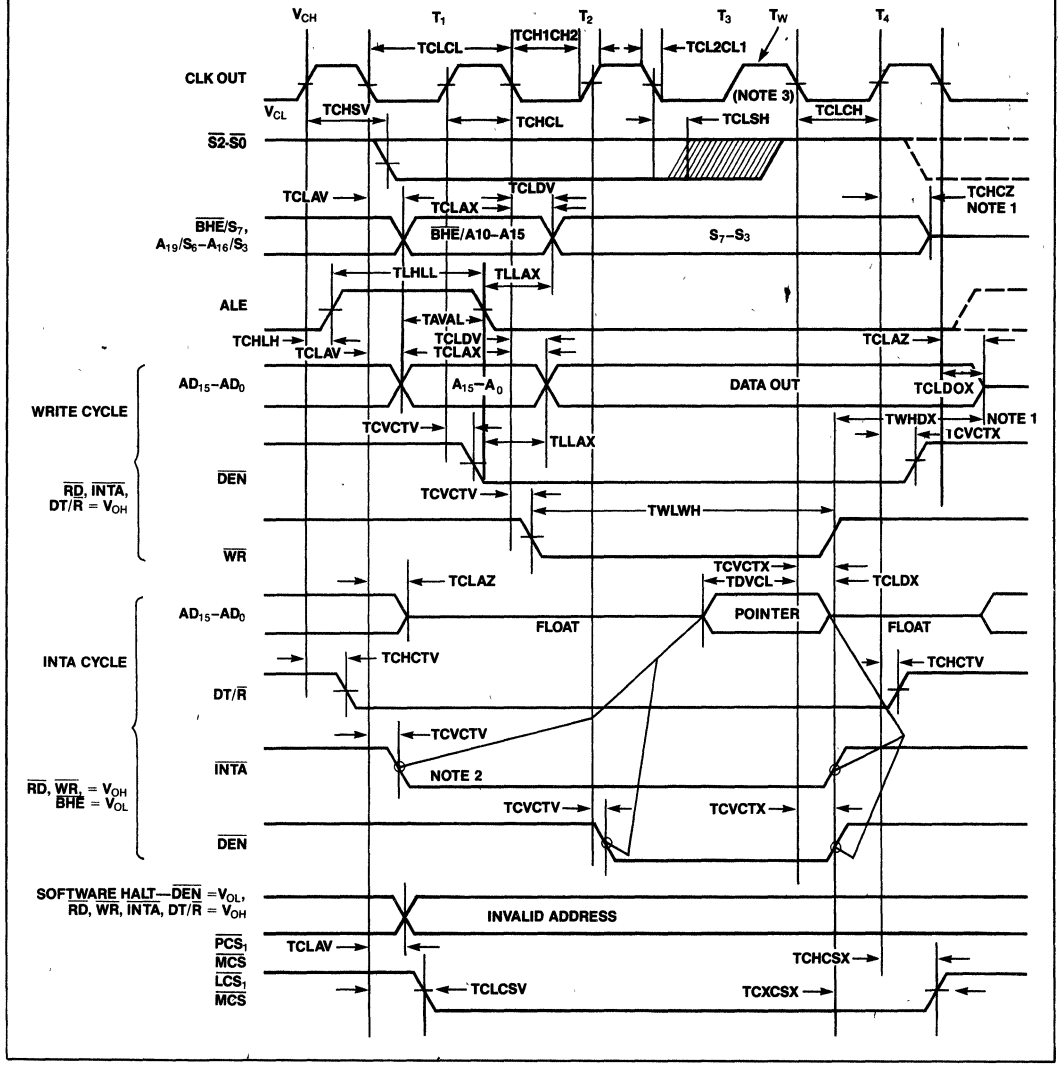
		80186 (8 MHz)		80186-6 (6 MHz)			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCKIN	CLKIN Period	62.5	250	83	250	ns	
TCKHL	CLKIN Fall Time		10		10	ns	3.5 to 1.0 volts
TCKLH	CLKIN Rise Time		10		10	ns	1.0 to 3.5 volts
TCLCK	CLKIN Low Time	25		33		ns	1.5 volts
TCHCK	CLKIN High Time	25		33		ns	1.5 volts

80186 CLKOUT Timing (200 pF load)

Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCICO	CLKIN to CLKOUT Skew		50		62.5	ns	
TCLCL	CLKOUT Period	125	500	167	500	ns	
TCLCH	CLKOUT Low Time	½ TCLCL-7.5		½ TCLCL-7.5		ns	1.5 volts
TCHCL	CLKOUT High Time	½ TCLCL-7.5		½ TCLCL-7.5		ns	1.5 volts
TCH1CH2	CLKOUT Rise Time		15		15	ns	1.0 to 3.5 volts
TCL2CL1	CLKOUT Fall Time		15		15	ns	3.5 to 1. volts

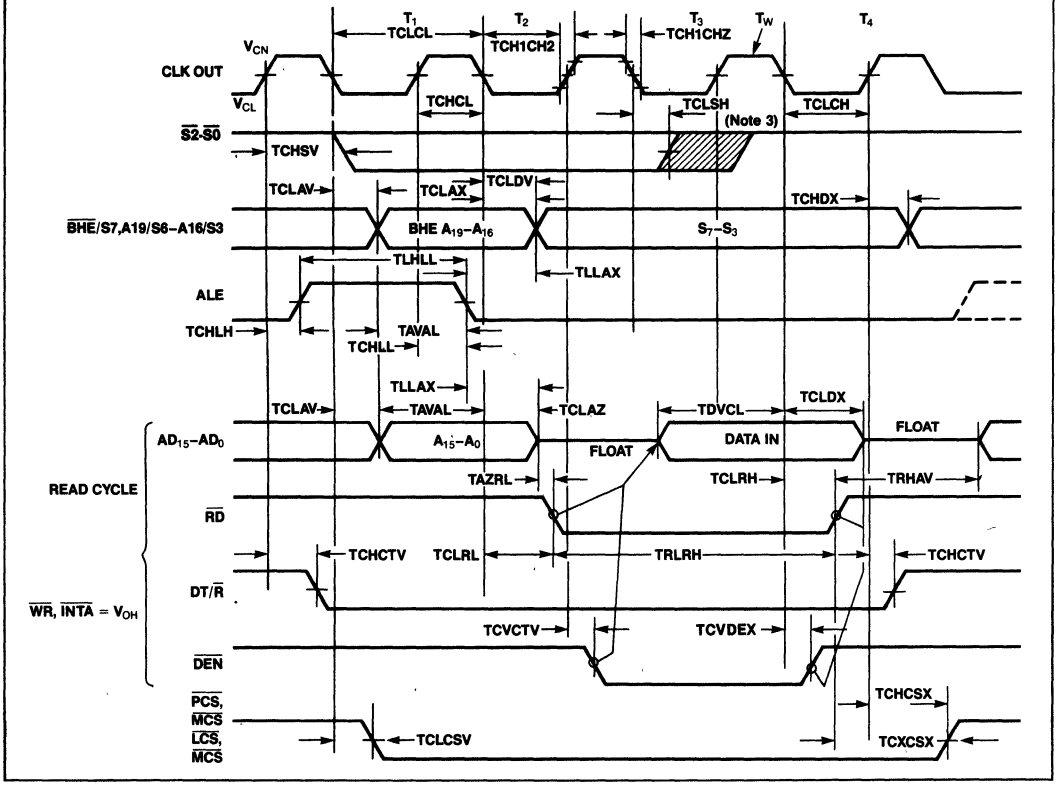
WAVEFORMS

MAJOR CYCLE TIMING



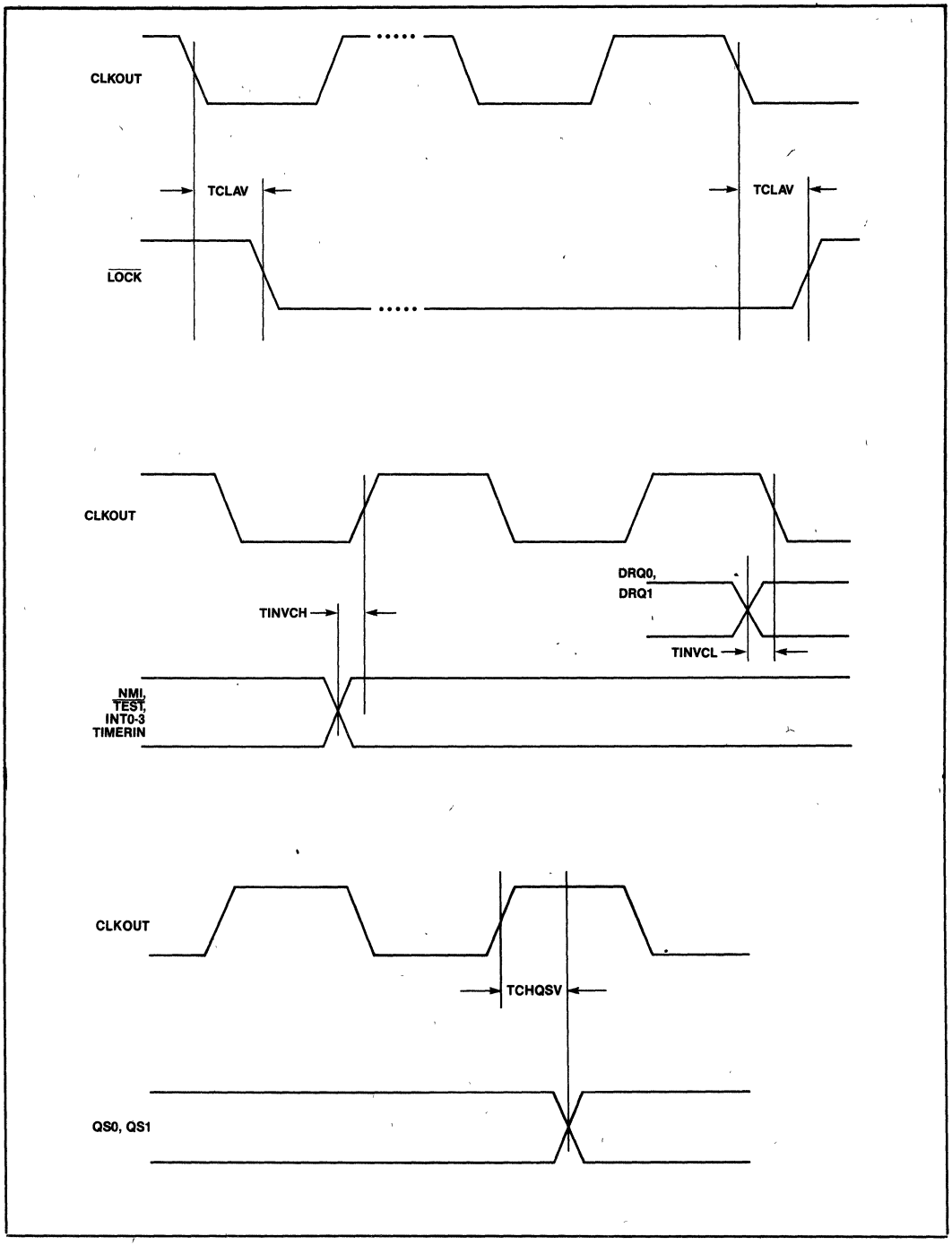
WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)



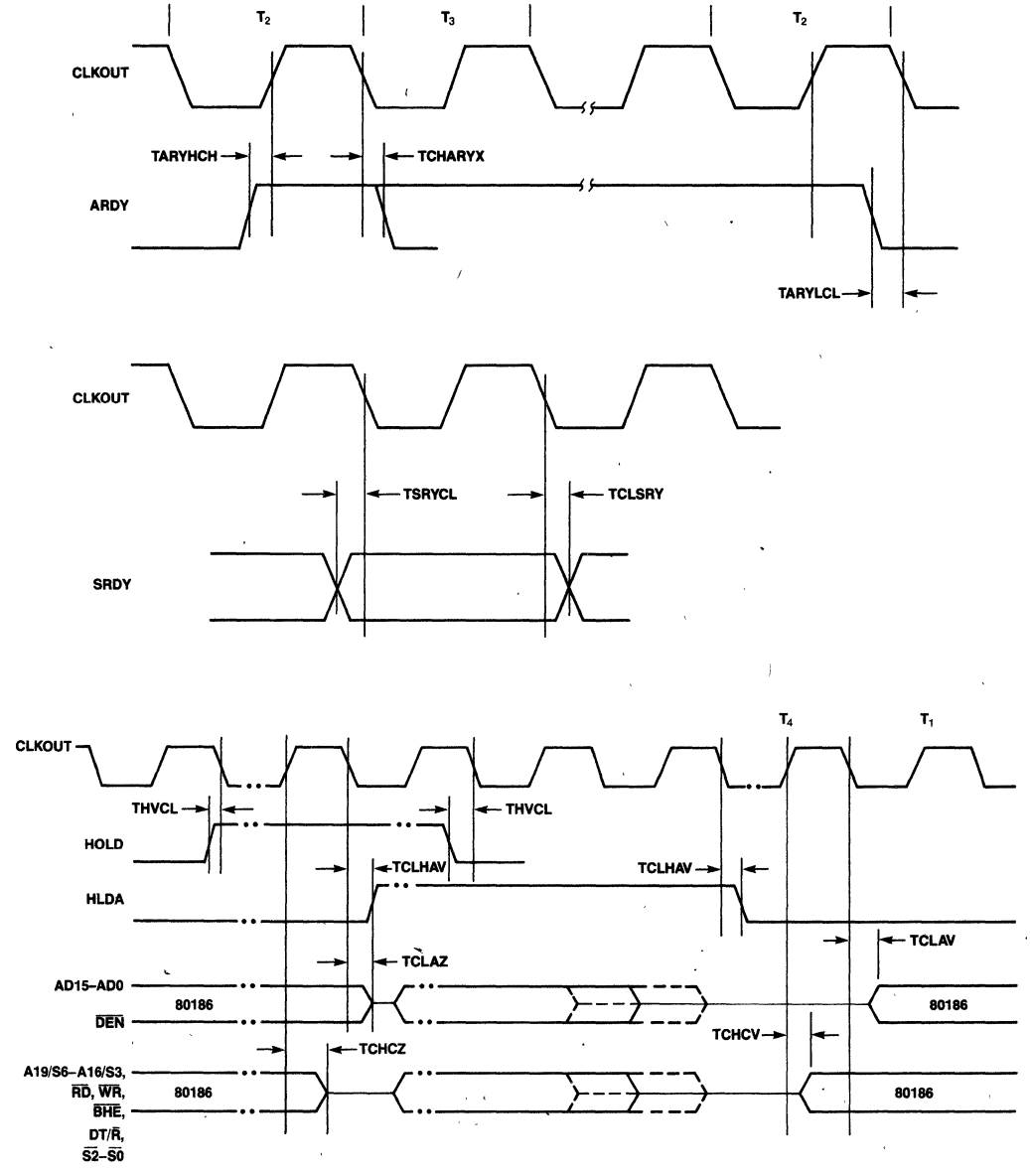
NOTES.
1. Following a Write cycle, the Local Bus is floated by the 80186 only when the 80186 enters a "Hold Acknowledge" state.
2 INTA occurs one clock later in RMX-mode.
3 Status inactive just prior to T_4

WAVEFORMS (Continued)



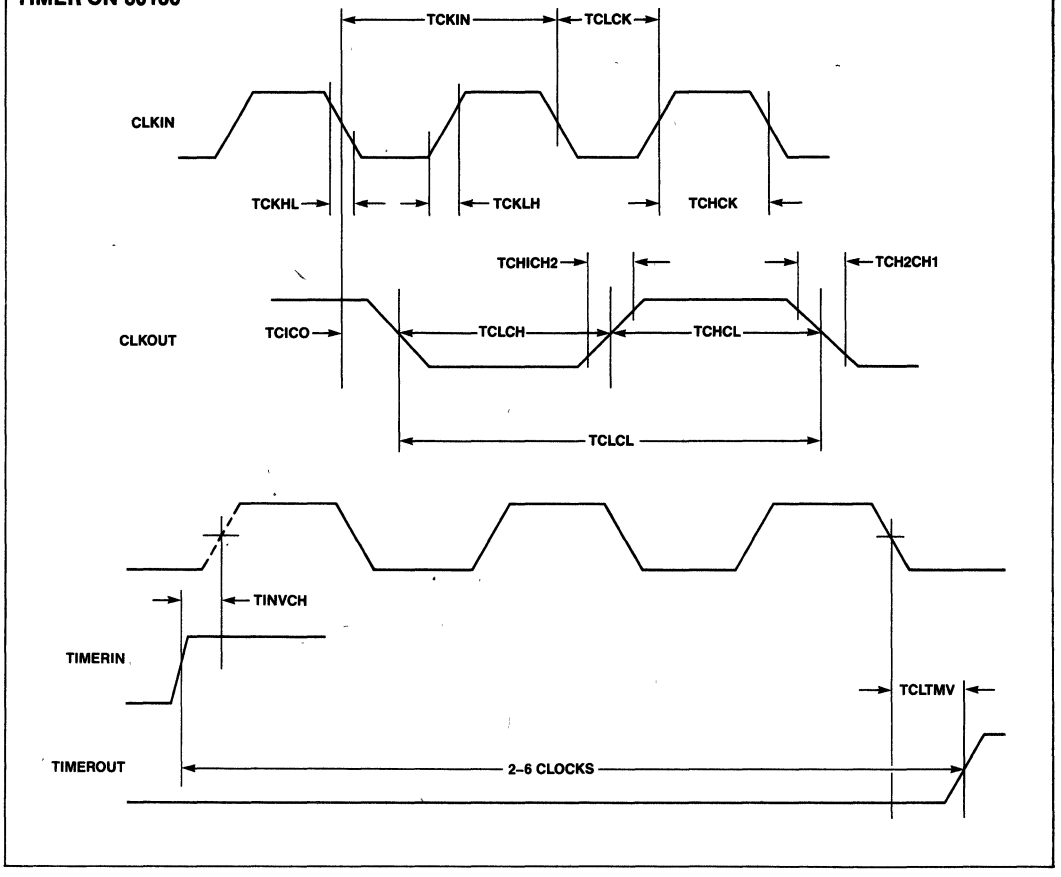
WAVEFORMS (Continued)

HOLD-HLDA TIMING



WAVEFORMS (Continued)

TIMER ON 80186



80186 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

FUNCTION	FORMAT	Clock Cycles	Comments
DATA TRANSFER			
MOV = Move:			
Register to Register/Memory	1 0 0 0 1 0 0 w | mod reg r/m	2/12	
Register/memory to register	1 0 0 0 1 0 1 w | mod reg r/m	2/9	
Immediate to register/memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	12–13	8/16-bit
Immediate to register	1 0 1 1 w reg | data | data if w = 1	3–4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w | addr-low | addr-high	9	
Accumulator to memory	1 0 1 0 0 0 1 w | addr-low | addr-high	8	
Register/memory to segment register	1 0 0 0 1 1 1 0 | mod 0 reg r/m	2/9	
Segment register to register/memory	1 0 0 0 1 1 0 0 | mod 0 reg r/m	2/11	
PUSH = Push:			
Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	16	
Register	0 1 0 1 0 reg	10	
Segment register	0 0 0 reg 1 1 0	9	
Immediate	0 1 1 0 1 0 s 0 | data | data if s = 0	10	
PUSHA = Push All	0 1 1 0 0 0 0 0	36	
POP = Pop:			
Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	20	
Register	0 1 0 1 1 reg	10	
Segment register	0 0 0 reg 1 1 1 (reg ≠ 01)	8	
POPA = Pop All	0 1 1 0 0 0 0 1	51	
XCHG = Exchange:			
Register/memory with register	1 0 0 0 0 1 1 w | mod reg r/m	4/17	
Register with accumulator	1 0 0 1 0 reg	3	
IN = Input from:			
Fixed port	1 1 1 0 0 1 0 w | port	10	
Variable port	1 1 1 0 1 1 0 w	8	
OUT = Output to:			
Fixed port	1 1 1 0 0 1 1 w | port	9	
Variable port	1 1 1 0 1 1 1 w	7	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	11	
LEA = Load EA to register	1 0 0 0 1 1 0 1 | mod reg r/m	6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1 | mod reg r/m (mod ≠ 11)	18	
LES = Load pointer to ES	1 1 0 0 0 1 0 0 | mod reg r/m (mod ≠ 11)	18	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1	2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	3	
PUSHF = Push flags	1 0 0 1 1 1 0 0	9	
POPF = Pop flags	1 0 0 1 1 1 0 1	8	
SEGMENT = Segment Override:			
CS	0 0 1 0 1 1 1 0	2	
SS	0 0 1 1 0 1 1 0	2	
DS	0 0 1 1 1 1 1 0	2	
ES	0 0 1 0 0 1 1 0	2	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
ARITHMETIC			
ADD = Add:			
Reg/memory with register to either	0 0 0 0 0 0 d w | mod reg r/m	3/10	
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | data | data if s w = 01	4/16	
Immediate to accumulator	0 0 0 0 0 1 0 w | data | data if w = 1	3/4	8/16-bit
ADC = Add with carry:			
Reg/memory with register to either	0 0 0 1 0 0 d w | mod reg r/m	3/10	
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | data | data if s w = 01	4/16	
Immediate to accumulator	0 0 0 1 0 1 0 w | data | data if w = 1	3/4	8/16-bit
INC = Increment:			
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 0 r/m	3/15	
Register	0 1 0 0 0 reg	3	
SUB = Subtract:			
Reg/memory and register to either	0 0 1 0 1 0 d w | mod reg r/m	3/10	
Immediate from register/memory	1 0 0 0 0 0 s w | mod 1 0 1 r/m | data | data if s w = 0 1	4/16	
Immediate from accumulator	0 0 1 0 1 1 0 w | data | data if w = 1	3/4	8/16-bit
SBB = Subtract with borrow:			
Reg/memory and register to either	0 0 0 1 1 0 d w | mod reg r/m	3/10	
Immediate from register/memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | data | data if s w = 0 1	4/16	
Immediate from accumulator	0 0 0 1 1 1 0 w | data | data if w = 1	3/4	8/16-bit
DEC = Decrement:			
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 1 r/m	3/15	
Register	0 1 0 0 1 reg	3	
CMP = Compare:			
Register/memory with register	0 0 1 1 1 0 1 w | mod reg r/m	3/10	
Register with register/memory	0 0 1 1 1 0 0 w | mod reg r/m	3/10	
Immediate with register/memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | data | data if s w = 0 1	3/10	
Immediate with accumulator	0 0 1 1 1 1 0 w | data | data if w = 1	3/4	8/16-bit
NEG = Change sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	3	
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	8	
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	4	
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	7	
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	4	
MUL = Multiply (unsigned)	1 1 1 1 0 1 1 w | mod 1 0 0 r/m		
Register-Byte		26–28	
Register-Word		35–37	
Memory-Byte		32–34	
Memory-Word		41–43	
IMUL = Integer multiply (signed)	1 1 1 1 0 1 1 w | mod 1 0 1 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43	
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 | mod reg r/m | data | data if s = 0	22–25/29–32	
DIV = Divide (unsigned)	1 1 1 1 0 1 1 w | mod 1 1 0 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
ARITHMETIC (Continued):			
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w | mod 1 1 1 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1	4	
LOGIC			
Shift/Rotate Instructions:			
Register/Memory,by 1	1 1 0 1 0 0 0 w | mod TTT r/m	2/15	
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	5+n/17+n	
Register/Memory by Count	1 1 0 0 0 0 0 w | mod TTT r/m | count	5+n/17+n	
	TTT Instruction 0 0 0 ROL 0 0 1 ROR 0 1 0 RCL 0 1 1 RCR 1 0 0 SHL/SAL 1 0 1 SHR 1 1 1 SAR		
AND = And:			
Reg/memory and register to either	0 0 1 0 0 0 d w | mod reg r/m	3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | data | data if w = 1	4/16	
Immediate to accumulator	0 0 1 0 0 1 0 w | data | data if w = 1	3/4	8/16-bit
TEST = And function to flags, no result:			
Register/memory and register	1 0 0 0 0 1 0 w | mod reg r/m	3/10	
Immediate data and register/memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	4/10	
Immediate data and accumulator	1 0 1 0 1 0 0 w | data | data if w = 1	3/4	8/16-bit
OR = Or:			
Reg/memory and register to either	0 0 0 0 1 0 d w | mod reg r/m	3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | data | data if w = 1	4/16	
Immediate to accumulator	0 0 0 0 1 1 0 w | data | data if w = 1	3/4	8/16-bit
XOR = Exclusive or:			
Reg/memory and register to either	0 0 1 1 0 0 d w | mod reg r/m	3/10	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | data | data if w = 1	4/16	
Immediate to accumulator	0 0 1 1 0 1 0 w | data | data if w = 1	3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	3	
STRING MANIPULATION:			
MOVS = Move byte/word	1 0 1 0 0 1 0 w	14	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w	22	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w	15	
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w	12	
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w	10	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	14	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	14	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
STRING MANIPULATION (Continued):			
Repeated by count in CX			
MOVS = Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	8+8n	
CMPS = Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+22n	
SCAS = Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+15n	
LODS = Load string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	6+11n	
STOS = Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	6+9n	
INS = Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	8+8n	
OUTS = Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	8+8n	
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	1 1 1 0 1 0 0 0 | disp-low | disp-high	14	
Register/memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r/m	13/19	
Direct intersegment	1 0 0 1 1 0 1 0 | segment offset | segment selector	23	
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r/m (mod ≠ 11)	38	
JMP = Unconditional jump:			
Short/long	1 1 1 0 1 0 1 1 | disp-low	13	
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	13	
Register/memory indirect within segment	1 1 1 1 1 1 1 1 | mod 1 0 0 r/m	11/17	
Direct intersegment	1 1 1 0 1 0 1 0 | segment offset | segment selector	13	
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r/m (mod ≠ 11)	26	
RET = Return from CALL:			
Within segment	1 1 0 0 0 0 1 1	16	
Within seg adding immed to SP	1 1 0 0 0 0 1 0 | data-low | data-high	18	
Intersegment	1 1 0 0 1 0 1 1	22	
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0 | data-low | data-high	25	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
CONTROL TRANSFER (Continued):			
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0 | disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0 | disp	4/13	
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0 | disp	4/13	
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0 | disp	4/13	
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0 | disp	4/13	
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0 | disp	4/13	
JO = Jump on overflow	0 1 1 1 0 0 0 0 | disp	4/13	
JS = Jump on sign	0 1 1 1 1 0 0 0 | disp	4/13	
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1 | disp	4/13	
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1 | disp	4/13	
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1 | disp	4/13	
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1 | disp	4/13	
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1 | disp	4/13	
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1 | disp	4/13	
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 | disp	4/13	
JNS = Jump on not sign	0 1 1 1 1 0 0 1 | disp	4/13	
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1 | disp	5/15	
LOOP = Loop CX times	1 1 1 0 0 0 1 0 | disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1 | disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0 | disp	6/16	
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 | data-low | data-high | L		
L = 0		15	
L = 1		25	
L > 1		22+16(n−1)	
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	8	
INT = Interrupt:			
Type specified	1 1 0 0 1 1 0 1 | type	47	
Type 3	1 1 0 0 1 1 0 0	45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	48/4	
IRET = Interrupt return	1 1 0 0 1 1 1 1	28	
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r/m	33–35	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	1 1 1 1 1 0 0 0	2	
CMC = Complement carry	1 1 1 1 0 1 0 1	2	
STC = Set carry	1 1 1 1 1 0 0 1	2	
CLD = Clear direction	1 1 1 1 1 1 0 0	2	
STD = Set direction	1 1 1 1 1 1 0 1	2	
CLI = Clear interrupt	1 1 1 1 1 0 1 0	2	
STI = Set interrupt	1 1 1 1 1 0 1 1	2	
HLT = Halt	1 1 1 1 0 1 0 0	2	
WAIT = Wait	1 0 0 1 1 0 1 1	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0	2	
ESC = Processor Extension Escape	1 0 0 1 1 T T T | mod LLL r/m (TTT LLL are opcode to processor extension)	6	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

FOOTNOTES

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

NOTE:
EA CALCULATION TIME IS 4 CLOCK CYCLES FOR ALL MODES, AND IS INCLUDED IN THE EXECUTION TIMES GIVEN WHENEVER APPROPRIATE.

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

PRELIMINARY

iAPX 88/10
8-BIT HMOS MICROPROCESSOR
8088/8088-2

- 8-Bit Data Bus Interface
- 16-Bit Internal Architecture
- Direct Addressing Capability to 1 Mbyte of Memory
- Direct Software Compatibility with iAPX 86/10 (8086 CPU)
- 14-Word by 16-Bit Register Set with Symmetrical Operations
- 24 Operand Addressing Modes
- Byte, Word, and Block Operations
- 8-Bit and 16-Bit Signed and Unsigned Arithmetic in Binary or Decimal, Including Multiply and Divide
- Compatible with 8155-2, 8755A-2 and 8185-2 Multiplexed Peripherals
- Two Clock Rates:
  5 MHz for 8088
  8 MHz for 8088-2
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range

The Intel® iAPX 88/10 is a new generation, high performance microprocessor implemented in N-channel, depletion load, silicon gate technology (HMOS), and packaged in a 40-pin CerDIP package. The processor has attributes of both 8- and 16-bit microprocessors. It is directly compatible with iAPX 86/10 software and 8080/8085 hardware and peripherals.

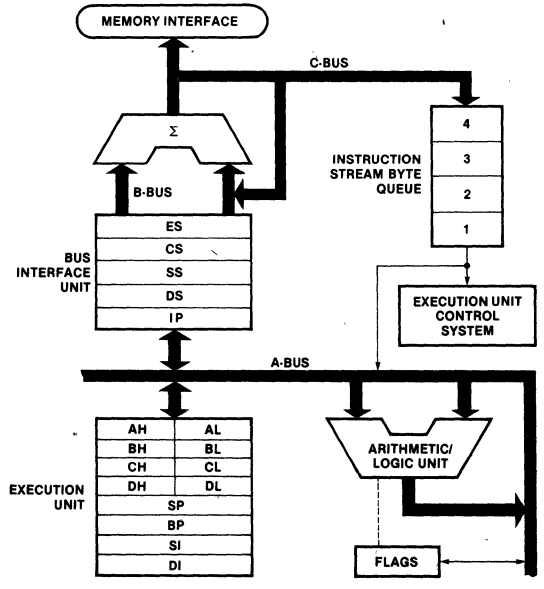


Figure 1. iAPX 88/10 CPU Functional Block Diagram

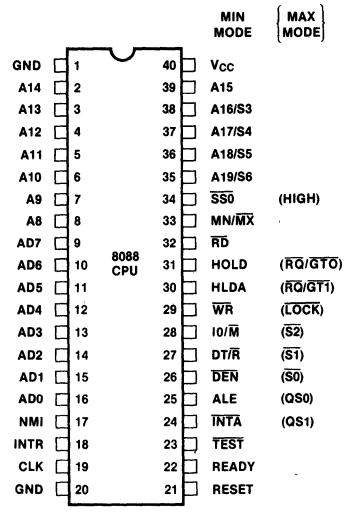


Figure 2. iAPX 88/10 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for 8088 systems in either minimum or maximum mode. The "local bus" in these descriptions is the direct multiplexed bus interface connection to the 8088 (without regard to additional bus buffers).

Symbol	Pin No.	Type	Name and Function
AD7–AD0	9-16	I/O	**Address Data Bus:** These lines constitute the time multiplexed memory/IO address (T1) and data (T2, T3, Tw, and T4) bus. These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A15–A8	2-8, 39	O	**Address Bus:** These lines provide address bits 8 through 15 for the entire bus cycle (T1–T4). These lines do not have to be latched by ALE to remain valid. A15–A8 are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".
A19/S6, A18/S5, A17/S4, A16/S3	34-38	O	**Address/Status:** During T1, these are the four most significant address lines for memory operations. During I/O operations, these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, Tw, and T4. S6 is always low. The status of the interrupt enable flag bit (S5) is updated at the beginning of each clock cycle. S4 and S3 are encoded as shown. This information indicates which segment register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge".
$\overline{RD}$	32	O	**Read:** Read strobe indicates that the processor is performing a memory or I/O read cycle, depending on the state of the IO/$\overline{M}$ pin or S2. This signal is used to read devices which reside on the 8088 local bus. $\overline{RD}$ is active LOW during T2, T3 and Tw of any read cycle, and is guaranteed to remain HIGH in T2 until the 8088 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".
READY	22	I	**READY:** is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The RDY signal from memory or I/O is synchronized by the 8284 clock generator to form READY. This signal is active HIGH. The 8088 READY input is not synchronized. Correct operation is not guaranteed if the set up and hold times are not met.
INTR	18	I	**Interrupt Request:** is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.
$\overline{TEST}$	23	I	**$\overline{TEST}$:** input is examined by the "wait for test" instruction. If the $\overline{TEST}$ input is LOW, execution continues, otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.
NMI	17	I	**Non-Maskable Interrupt:** is an edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from a LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

S4	S3	CHARACTERISTICS
0 (LOW)	0	Alternate Data
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data
S6 is 0 (LOW)		

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
RESET	21	I	**RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution, as described in the instruction set description, when RESET returns LOW. RESET is internally synchronized.
CLK	19	I	**Clock:** provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}	40		**V_{CC}:** is the +5V ±10% power supply pin.
GND	1, 20		**GND:** are the ground pins.
MN/$\overline{MX}$	33	I	**Minimum/Maximum:** indicates what mode the processor is to operate in. The two modes are discussed in the following sections.

The following pin function descriptions are for the 8088 minimum mode (i.e., MN/MX = V_{CC}). Only the pin functions which are unique to minimum mode are described; all other pin functions are as described above.

IO/$\overline{M}$	28	O	**Status Line:** is an inverted maximum mode $\overline{S2}$. It is used to distinguish a memory access from an I/O access. IO/$\overline{M}$ becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (I/O=HIGH, M=LOW). IO/$\overline{M}$ floats to 3-state OFF in local bus "hold acknowledge".
$\overline{WR}$	29	O	**Write:** strobe indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the IO/$\overline{M}$ signal. WR is active for T2, T3, and Tw of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".
$\overline{INTA}$	24	O	**INTA:** is used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3, and Tw of each interrupt acknowledge cycle.
ALE	25	O	**Address Latch Enable:** is provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during clock low of T1 of any bus cycle. Note that ALE is never floated.
DT/$\overline{R}$	27	O	**Data Transmit/Receive:** is needed in a minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically, DT/$\overline{R}$ is equivalent to $\overline{S1}$ in the maximum mode, and its timing is the same as for IO/$\overline{M}$ (T=HIGH, R=LOW). This signal floats to 3-state OFF in local "hold acknowledge".
$\overline{DEN}$	26	O	**Data Enable:** is provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. $\overline{DEN}$ is active LOW during each memory and I/O access, and for $\overline{INTA}$ cycles. For a read or $\overline{INTA}$ cycle, it is active from the middle of T2 until the middle of T4, while for a write cycle, it is active from the beginning of T2 until the middle of T4. $\overline{DEN}$ floats to 3-state OFF during local bus "hold acknowledge".
HOLD, HLDA	30,31	I, O	**HOLD:** indicates that another master is requesting a local bus "hold". To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledgement, in the middle of a T4 or TI clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor lowers HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the set up time.
$\overline{SSO}$	34	O	**Status line:** is logically equivalent to $\overline{S0}$ in the maximum mode. The combination of $\overline{SSO}$, IO/$\overline{M}$ and DT/$\overline{R}$ allows the system to completely decode the current bus cycle status.

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SSO}$	CHARACTERISTICS
1 (HIGH)	0	0	Interrupt Acknowledge
1	0	1	Read I/O port
1	1	0	Write I/O port
1	1	1	Halt
0 (LOW)	0	0	Code access
0	0	1	Read memory
0	1	0	Write memory
0	1	1	Passive

Table 1. Pin Description (Continued)

The following pin function descriptions are for the 8088, 8228 system in maximum mode (i.e., MN/MX=GND.) Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

Symbol	Pin No.	Type	Name and Function
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	26-28	O	**Status:** is active during clock high of T4, T1, and T2, and is returned to the passive state (1,1,1) during T3 or during Tw when READY is HIGH. This status is used by the 8288 bus controller to generate all memory and I/O access control signals. Any change by $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or Tw is used to indicate the end of a bus cycle. These signals float to 3-state OFF during "hold acknowledge". During the first clock cycle after RESET becomes active, these signals are active HIGH. After this first clock, they float to 3-state OFF.
$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$	30, 31	I/O	**Request/Grant:** pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor, so may be left unconnected. The request/grant sequence is as follows (See Figure 8): 1. A pulse of one CLK wide from another local bus master indicates a local bus request ("hold") to the 8088 (pulse 1). 2. During a T4 or TI clock cycle, a pulse one clock wide from the 8088 to the requesting master (pulse 2), indicates that the 8088 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge". The same rules as for HOLD/HOLDA apply as for when the bus is released. 3. A pulse one CLK wide from the requesting master indicates to the 8088 (pulse 3) that the "hold" request is about to end and that the 8088 can reclaim the local bus at the next CLK. The CPU then enters T4. Each master-master exchange of the local bus is a sequence of three pulses. There must be one idle CLK cycle after each bus exchange. Pulses are active LOW. If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met: 1. Request occurs on or before T2. 2. Current cycle is not the low bit of a word. 3. Current cycle is not the first acknowledge of an interrupt acknowledge sequence. 4. A locked instruction is not currently executing. If the local bus is idle when the request is made the two possible events will follow: 1. Local bus will be released during the next clock. 2. A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1 (HIGH)	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{LOCK}$	29	O	**$\overline{LOCK}$:** indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active (LOW). The $\overline{LOCK}$ signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state off in "hold acknowledge".
QS1, QS0	24, 25	O	**Queue Status:** provide status to allow external tracking of the internal 8088 instruction queue. The queue status is valid during the CLK cycle after which the queue operation is performed.
—	34	O	Pin 34 is always high in the maximum mode.

QS1	QS0	CHARACTERISTICS
0 (LOW)	0	No operation
0	1	First byte of opcode from queue
1 (HIGH)	0	Empty the queue
1	1	Subsequent byte from queue

FUNCTIONAL DESCRIPTION

Memory Organization

The processor provides a 20-bit address to memory which locates the byte being referenced. The memory is organized as a linear array of up to 1 million bytes, addressed as 00000(H) to FFFFF(H). The memory is logically divided into code, data, extra data, and stack segments of up to 64K bytes each, with each segment falling on 16-byte boundaries. (See Figure 3.)

All memory references are made relative to base addresses contained in high speed segment registers. The segment types were chosen based on the addressing needs of programs. The segment register to be selected is automatically chosen according to the rules of the following table. All information in one segment type share the same logical attributes (e.g. code or data). By structuring memory into relocatable areas of similar characteristics and by automatically selecting segment registers, programs are shorter, faster, and more structured.

Word (16-bit) operands can be located on even or odd address boundaries. For address and data operands, the least significant byte of the word is stored in the lower valued address location and the most significant byte in the next higher address location. The BIU will automatically execute two fetch or write cycles for 16-bit operands.

Certain locations in memory are reserved for specific CPU operations. (See Figure 4.) Locations from addresses FFFF0H through FFFFFH are reserved for operations including a jump to the initial system initialization routine. Following RESET, the CPU will always begin execution at location FFFF0H where the jump must be located. Locations 00000H through 003FFH are reserved for interrupt operations. Four-byte pointers consisting of a 16-bit segment address and a 16-bit offset address direct program flow to one of the 256 possible interrupt service routines. The pointer elements are assumed to have been stored at their respective places in reserved memory prior to the occurrence of interrupts.

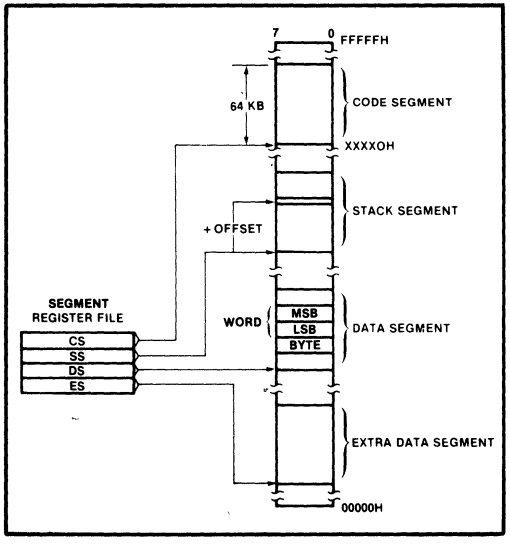


Figure 3. Memory Organization

Minimum and Maximum Modes

The requirements for supporting minimum and maximum 8088 systems are sufficiently different that they cannot be done efficiently with 40 uniquely defined pins. Consequently, the 8088 is equipped with a strap pin (MN/$\overline{MX}$) which defines the system configuration. The definition of a certain subset of the pins changes, dependent on the condition of the strap pin. When the MN/$\overline{MX}$ pin is strapped to GND, the 8088 defines pins 24 through 31 and 34 in maximum mode. When the MN/$\overline{MX}$ pin is strapped to V_{CC}, the 8088 generates bus control signals itself on pins 24 through 31 and 34.

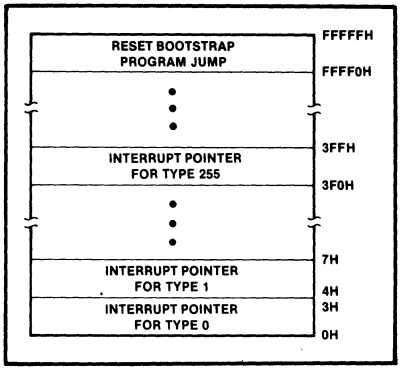


Figure 4. Reserved Memory Locations

Memory Reference Need	Segment Register Used	Segment Selection Rule
Instructions	CODE (CS)	Automatic with all instruction prefetch.
Stack	STACK (SS)	All stack pushes and pops. Memory references relative to BP base register except data references.
Local Data	DATA (DS)	Data references when: relative to stack, destination of string operation, or explicitly overridden.
External (Global) Data	EXTRA (ES)	Destination of string operations: Explicitly selected using a segment override.

The minimum mode 8088 can be used with either a multiplexed or demultiplexed bus. The multiplexed bus configuration is compatible with the MCS-85™ multiplexed bus peripherals (8155, 8156, 8355, 8755A, and 8185). This configuration (See Figure 5) provides the user with a minimum chip count system. This architecture provides the 8088 processing power in a highly integrated form.

The demultiplexed mode requires one latch (for 64K addressability) or two latches (for a full megabyte of addressing). A third latch can be used for buffering if the address bus loading requires it. An 8286 or 8287 transceiver can also be used if data bus buffering is required. (See Figure 6.) The 8088 provides $\overline{DEN}$ and DT/$\overline{R}$ to control the transceiver, and ALE to latch the addresses. This configuration of the minimum mode provides the standard demultiplexed bus structure with heavy bus buffering and relaxed bus timing requirements.

The maximum mode employs the 8288 bus controller. (See Figure 7.) The 8288 decodes status lines $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$, and provides the system with all bus control signals. Moving the bus control to the 8288 provides better source and sink current capability to the control lines, and frees the 8088 pins for extended large system features. Hardware lock, queue status, and two request/grant interfaces are provided by the 8088 in maximum mode. These features allow co-processors in local bus and remote bus configurations.

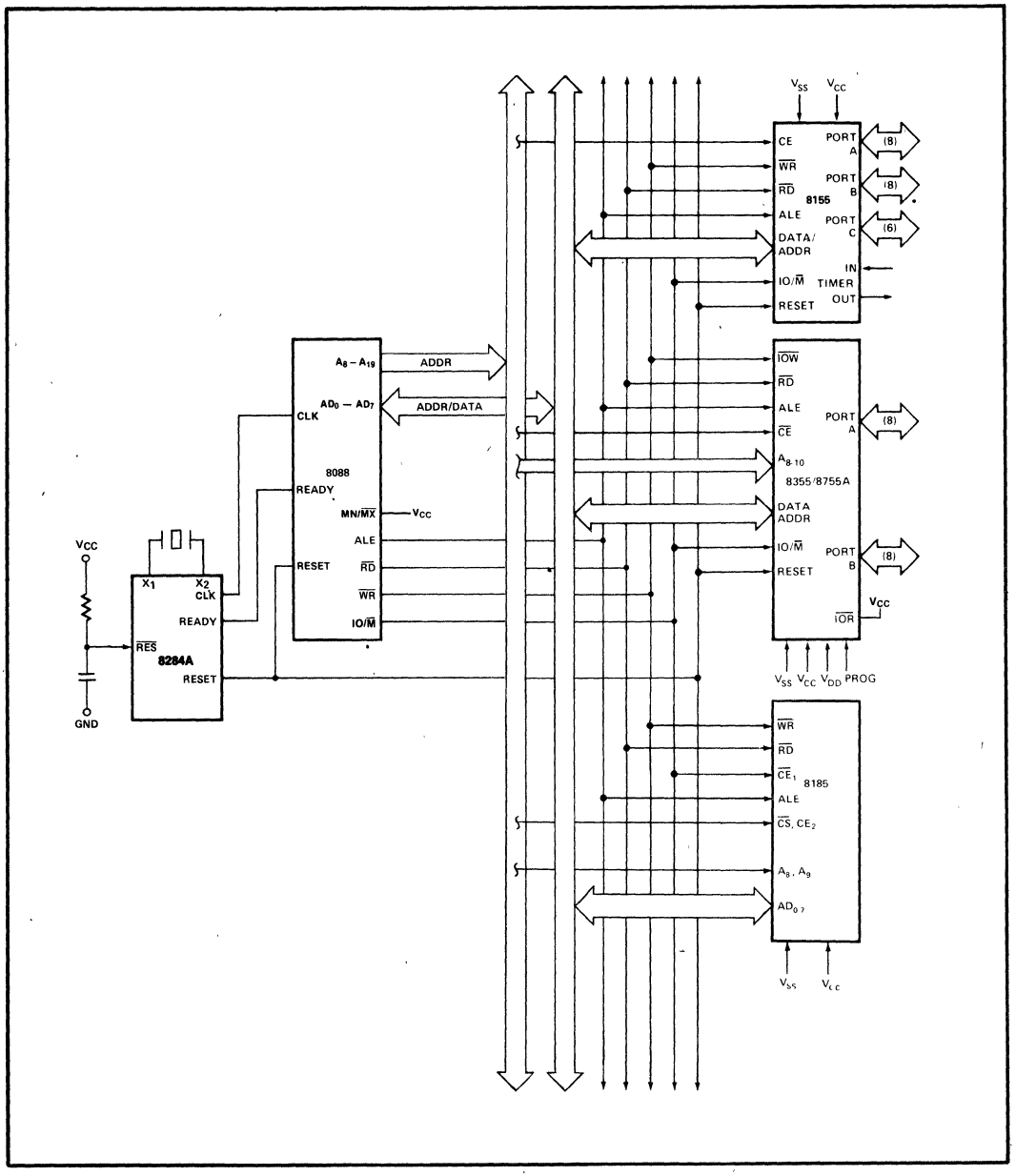


Figure 5. Multiplexed Bus Configuration

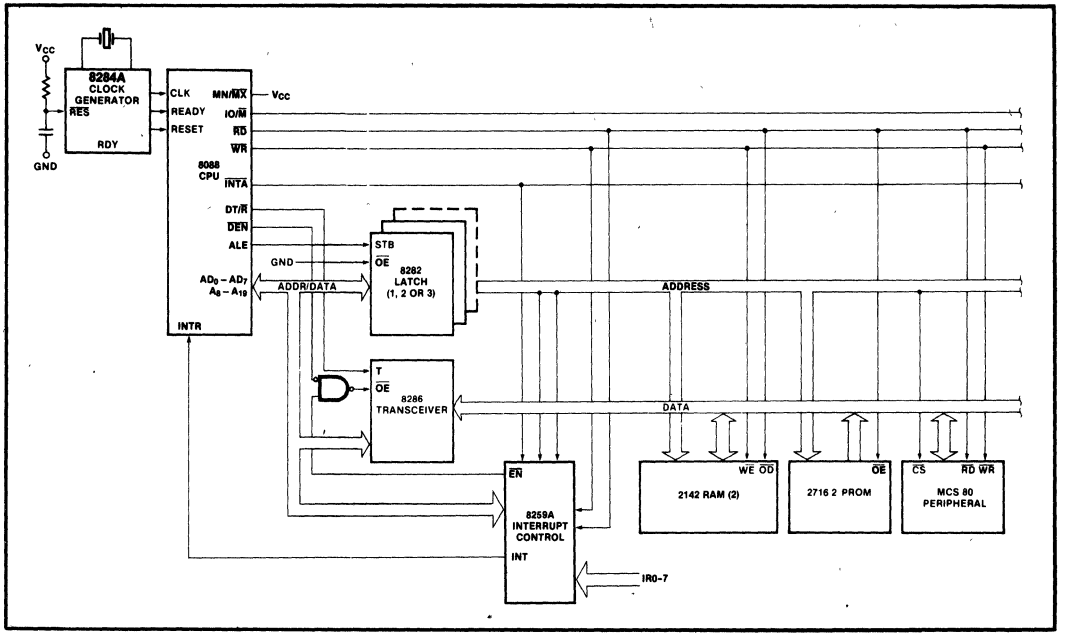


Figure 6. Demultiplexed Bus Configuration

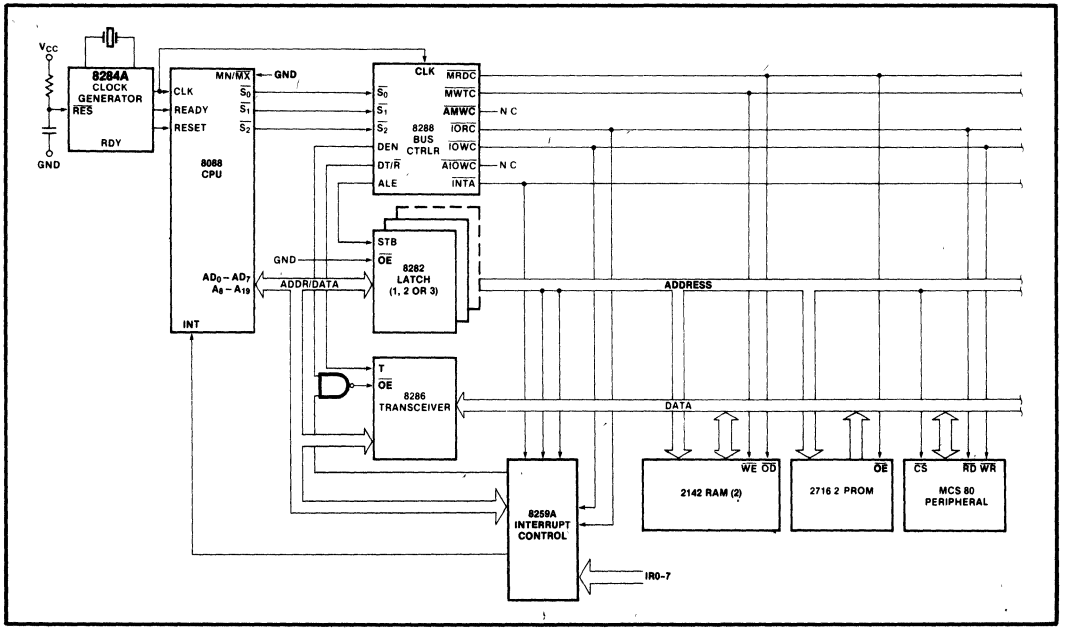


Figure 7. Fully Buffered System Using Bus Controller

Bus Operation

The 8088 address/data bus is broken into three parts — the lower eight address/data bits (AD0-AD7), the middle eight address bits (A8-A15), and the upper four address bits (A16-A19). The address/data bits and the highest four address bits are time multiplexed. This technique provides the most efficient use of pins on the processor, permitting the use of a standard 40 lead package. The middle eight address bits are not multiplexed, i.e. they remain valid throughout each bus cycle. In addition, the bus can be demultiplexed at the processor with a single address latch if a standard, non-multiplexed bus is desired for the system.

Each processor bus cycle consists of at least four CLK cycles. These are referred to as T1, T2, T3, and T4. (See Figure 8). The address is emitted from the processor during T1 and data transfer occurs on the bus during T3 and T4. T2 is used primarily for changing the direction of the bus during read operations. In the event that a "NOT READY" indication is given by the addressed device,

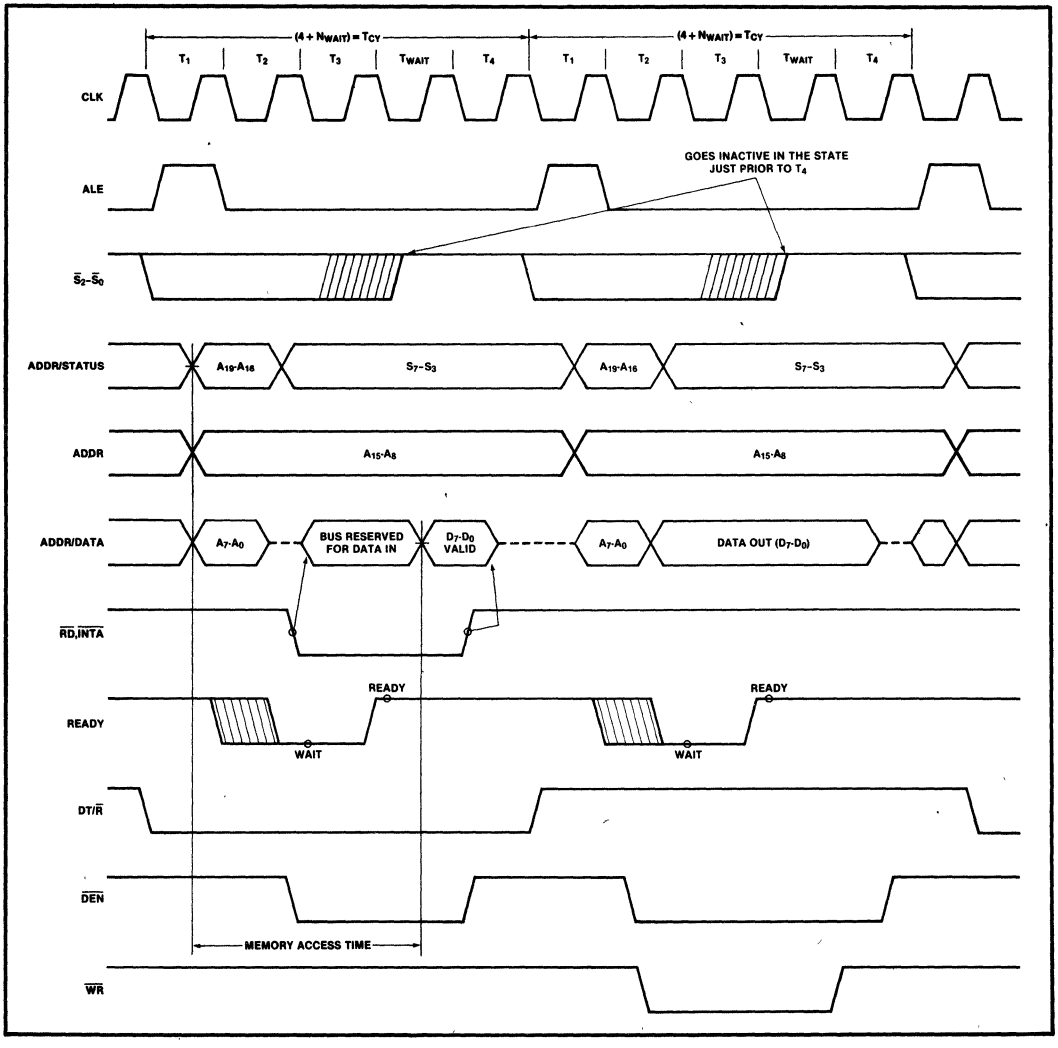


Figure 8. Basic System Timing

"wait" states (Tw) are inserted between T3 and T4. Each inserted "wait" state is of the same duration as a CLK cycle. Periods can occur between 8088 driven bus cycles. These are referred to as "idle" states (Ti), or inactive CLK cycles. The processor uses these cycles for internal housekeeping.

During T1 of any bus cycle, the ALE (address latch enable) signal is emitted (by either the processor or the 8288 bus controller, depending on the MN/$\overline{MX}$ strap). At the trailing edge of this pulse, a valid address and certain status information for the cycle may be latched.

Status bits $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ are used by the bus controller, in maximum mode, to identify the type of bus transaction according to the following table:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CHARACTERISTICS
0 (LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O
0	1	0	Write I/O
0	1	1	Halt
1 (HIGH)	0	0	Instruction Fetch
1	0	1	Read Data from Memory
1	1	0	Write Data to Memory
1	1	1	Passive (no bus cycle)

Status bits S3 through S6 are multiplexed with high order address bits and are therefore valid during T2 through T4. S3 and S4 indicate which segment register was used for this bus cycle in forming the address according to the following table:

S_4	S_3	CHARACTERISTICS
0 (LOW)	0	Alternate Data (extra segment)
0	1	Stack
1 (HIGH)	0	Code or None
1	1	Data

S5 is a reflection of the PSW interrupt enable bit. S6 is always equal to 0.

I/O Addressing

In the 8088, I/O operations can address up to a maximum of 64K I/O registers. The I/O address appears in the same format as the memory address on bus lines A15-A0. The address lines A19-A16 are zero in I/O operations. The variable I/O instructions, which use register DX as a pointer, have full address capability, while the direct I/O instructions directly address one or two of the 256 I/O byte locations in page 0 of the I/O address space. I/O ports are addressed in the same manner as memory locations.

Designers familiar with the 8085 or upgrading an 8085 design should note that the 8085 addresses I/O with an 8-bit address on both halves of the 16-bit address bus. The 8088 uses a full 16-bit address on its lower 16 address lines.

EXTERNAL INTERFACE

Processor Reset and Initialization

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin. The 8088 RESET is required to be HIGH for greater than four clock cycles. The 8088 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 7 clock cycles. After this interval the 8088 operates normally, beginning with the instruction in absolute location FFFF0H. (See Figure 4.) The RESET input is internally synchronized to the processor clock. At initialization, the HIGH to LOW transition of RESET must occur no sooner than 50 μs after power up, to allow complete initialization of the 8088.

If INTR is asserted sooner than nine clock cycles after the end of RESET, the processor may execute one instruction before responding to the interrupt.

All 3-state outputs float to 3-state OFF during RESET. Status is active in the idle state for the first clock after RESET becomes active and then floats to 3-state OFF.

Interrupt Operations

Interrupt operations fall into two classes: software or hardware initiated. The software initiated interrupts and software aspects of hardware interrupts are specified in the instruction set description in the iAPX 88 book or the iAPX 86,88 User's Manual. Hardware interrupts can be classified as nonmaskable or maskable.

Interrupts result in a transfer of control to a new program location. A 256 element table containing address pointers to the interrupt service program locations resides in absolute locations 0 through 3FFH (see Figure 4), which are reserved for this purpose. Each element in the table is 4 bytes in size and corresponds to an interrupt "type." An interrupting device supplies an 8-bit type number, during the interrupt acknowledge sequence, which is used to vector through the appropriate element to the new interrupt service program location.

Non-Maskable Interrupt (NMI)

The processor provides a single non-maskable interrupt (NMI) pin which has higher priority than the maskable interrupt request (INTR) pin. A typical use would be to activate a power failure routine. The NMI is edge-triggered on a LOW to HIGH transition. The activation of this pin causes a type 2 interrupt.

NMI is required to have a duration in the HIGH state of greater than two clock cycles, but is not required to be synchronized to the clock. Any higher going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves (2 bytes in the case of word moves) of a block type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur

before, during, or after the servicing of NMI. Another high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

Maskable Interrupt (INTR)

The 8088 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable (IF) flag bit. The interrupt request signal is level triggered. It is internally synchronized during each clock cycle on the high-going edge of CLK. To be responded to, INTR must be present (HIGH) during the clock period preceding the end of the current instruction or the end of a whole move for a block type instruction. During interrupt response sequence, further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt, or single step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored, the enable bit will be zero unless specifically set by an instruction.

During the response sequence (See Figure 9), the processor executes two successive (back to back) interrupt acknowledge cycles. The 8088 emits the LOCK signal (maximum mode only) from T2 of the first bus cycle until T2 of the seoond. A local bus "hold" request will not be honored until the end of the second bus cycle. In the second bus cycle, a byte is fetched from the external interrupt system (e.g., 8259A PIC) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table. An INTR signal left HIGH will be continually responded to within the limitations of the enable bit and sample period. The interrupt return instruction includes a flags pop which returns the status of the original interrupt enable bit when it restores the flags.

HALT

When a software HALT instruction is executed, the processor indicates that it is entering the HALT state in one of two ways, depending upon which mode is strapped. In minimum mode, the processor issues ALE, delayed by one clock cycle, to allow the system to latch the halt status. Halt status is available on IO/$\overline{M}$, DT/$\overline{R}$, and $\overline{SSO}$. In maximum mode, the processor issues appropriate HALT status on $\overline{S2}$, $\overline{S1}$, and $\overline{S0}$, and the 8288 bus controller issues one ALE. The 8088 will not leave the HALT state when a local bus hold is entered while in HALT. In this case, the processor reissues the HALT indicator at the end of the local bus hold. An interrupt request or RESET will force the 8088 out of the HALT state.

Read/Modify/Write (Semaphore) Operations via LOCK

The LOCK status information is provided by the processor when consecutive bus cycles are required during the execution of an instruction. This allows the processor to perform read/modify/write operations on memory (via the "exchange register with memory" instruction), without another system bus master receiving intervening memory cycles. This is useful in multiprocessor system configurations to accomplish "test and set lock" operations. The $\overline{LOCK}$ signal is activated (LOW) in the clock cycle following decoding of the LOCK prefix instruction. It is deactivated at the end of the last bus cycle of the instruction following the LOCK prefix. While $\overline{LOCK}$ is active, a request on a $\overline{RQ}/\overline{GT}$ pin will be recorded, and then honored at the end of the LOCK.

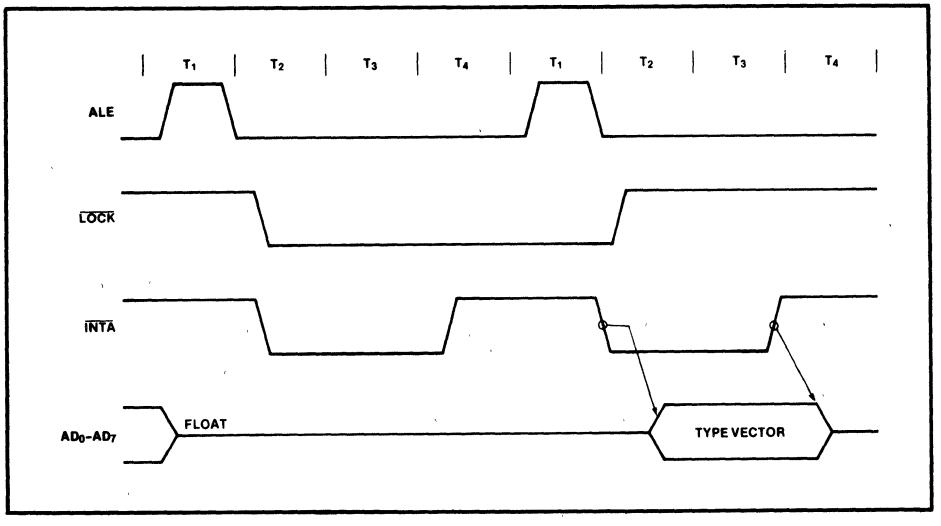


Figure 9. Interrupt Acknowledge Sequence

External Synchronization via $\overline{TEST}$

As an alternative to interrupts, the 8088 provides a single software-testable input pin ($\overline{TEST}$). This input is utilized by executing a WAIT instruction. The single WAIT instruction is repeatedly executed until the $\overline{TEST}$ input goes active (LOW). The execution of WAIT does not consume bus cycles once the queue is full.

If a local bus request occurs during WAIT execution, the 8088 3-states all output drivers. If interrupts are enabled, the 8088 will recognize interrupts and process them. The WAIT instruction is then refetched, and reexecuted.

Basic System Timing

In minimum mode, the MN/$\overline{MX}$ pin is strapped to V_{CC} and the processor emits bus control signals compatible with the 8085 bus structure. In maximum mode, the MN/$\overline{MX}$ pin is strapped to GND and the processor emits coded status information which the 8288 bus controller uses to generate MULTIBUS compatible bus control signals.

System Timing — Minimum System

(See Figure 8.)

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal. The trailing (low going) edge of this signal is used to latch the address information, which is valid on the address/data bus (AD0–AD7) at this time, into the 8282/8283 latch. Address lines A8 through A15 do not need to be latched because they remain valid throughout the bus cycle. From T1 to T4 the IO/$\overline{M}$ signal indicates a memory or I/O operation. At T2 the address is removed from the address/data bus and the bus goes to a high impedance state. The read control signal is also asserted at T2. The read ($\overline{RD}$) signal causes the addressed device to enable its data bus drivers to the local bus. Some time later, valid data will be available on the bus and the addressed device will drive the READY line HIGH. When the processor returns the read signal to a HIGH level, the addressed device will again 3-state its bus drivers. If a transceiver (8286/8287) is required to buffer the 8088 local bus, signals DT/$\overline{R}$ and $\overline{DEN}$ are provided by the 8088.

A write cycle also begins with the assertion of ALE and the emission of the address. The IO/$\overline{M}$ signal is again asserted to indicate a memory or I/O write operation. In T2, immediately following the address emission, the processor emits the data to be written into the addressed location. This data remains valid until at least the middle of T4. During T2, T3, and T_W, the processor asserts the write control signal. The write ($\overline{WR}$) signal becomes active at the beginning of T2, as opposed to the read, which is delayed somewhat into T2 to provide time for the bus to float.

The basic difference between the interrupt acknowledge cycle and a read cycle is that the interrupt acknowledge ($\overline{INTA}$) signal is asserted in place of the read ($\overline{RD}$) signal and the address bus is floated. (See Figure 9.). In the second of two successive $\overline{INTA}$ cycles, a byte of information is read from the data bus, as supplied by the interrupt system logic (i.e. 8259A priority interrupt controller). This byte identifies the source (type) of the interrupt. It is multiplied by four and used as a pointer into the interrupt vector lookup table, as described earlier.

Bus Timing — Medium Complexity Systems

(See Figure 10.)

For medium complexity systems, the MN/$\overline{MX}$ pin is connected to GND and the 8288 bus controller is added to the system, as well as an 8282/8283 latch for latching the system address, and an 8286/8287 transceiver to allow for bus loading greater than the 8088 is capable of handling. Signals ALE, $\overline{DEN}$, and DT/$\overline{R}$ are generated by the 8288 instead of the processor in this configuration, although their timing remains relatively the same. The 8088 status outputs ($\overline{S2}$, $\overline{S1}$, and $\overline{S0}$) provide type of cycle information and become 8288 inputs. This bus cycle information specifies read (code, data, or I/O), write (data or I/O), interrupt acknowledge, or software halt. The 8288 thus issues control signals specifying memory read or write, I/O read or write, or interrupt acknowledge. The 8288 provides two types of write strobes, normal and advanced, to be applied as required. The normal write strobes have data valid at the leading edge of write. The advanced write strobes have the same timing as read strobes, and hence, data is not valid at the leading edge of write. The 8286/8287 transceiver receives the usual T and $\overline{OE}$ inputs from the 8288's DT/$\overline{R}$ and $\overline{DEN}$ outputs.

The pointer into the interrupt vector table, which is passed during the second $\overline{INTA}$ cycle, can derive from an 8259A located on either the local bus or the system bus. If the master 8289A priority interrupt controller is positioned on the local bus, a TTL gate is required to disable the 8286/8287 transceiver when reading from the master 8259A during the interrupt acknowledge sequence and software "poll".

The 8088 Compared to the 8086

The 8088 CPU is an 8-bit processor designed around the 8086 internal structure. Most internal functions of the 8088 are identical to the equivalent 8086 functions. The 8088 handles the external bus the same way the 8086 does with the distinction of handling only 8 bits at a time. Sixteen-bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 8088 and 8086 are outlined below. The engineer who is unfamiliar with the 8086 is referred to the iAPX 86, 88 User's Manual, Chapters 2 and 4, for function description and instruction set information. Internally, there are three differences between the 8088 and the 8086. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 8088, whereas the 8086 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 8088 BIU will fetch a new instruction to load into the queue each time there is a 1 byte hole (space available) in the queue. The 8086 waits until a 2-byte space is available.
- The internal execution time of the instruction set is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU is also limited by the speed of instruction fetches. This latter problem only occurs when a series of simple operations occur. When the more sophisticated instructions of the 8088 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 8088 and 8086 are completely software compatible by virture of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally as well on an 8088 or an 8086.

The hardware interface of the 8088 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes:

- A8–A15 — These pins are only address outputs on the 8088. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 8088 and has been eliminated.
- $\overline{SSO}$ provides the $\overline{S0}$ status information in the minimum mode. This output occurs on pin 34 in minimum mode only. DT/$\overline{R}$, IO/$\overline{M}$, and $\overline{SSO}$ provide the complete bus status in minimum mode.
- IO/$\overline{M}$ has been inverted to be compatible with the MCS-85 bus structure.
- ALE is delayed by one clock cycle in the minimum mode when entering HALT, to allow the status to be latched with ALE.

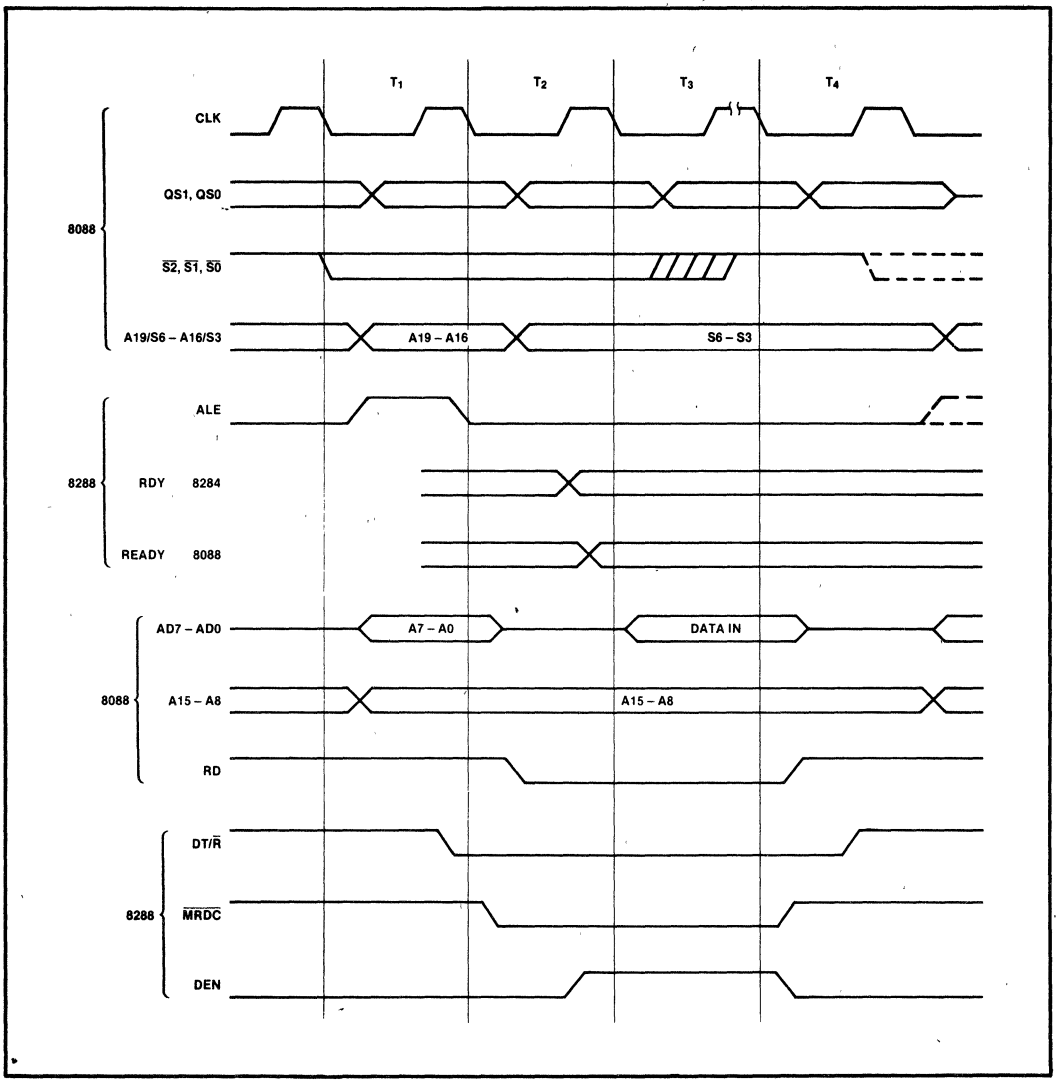


Figure 10. Medium Complexity System Timing

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to Ground . . . . . −1.0 to +7V
Power Dissipation . . . . . 2.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

(8088: T_A = 0°C to 70°C, V_{CC} = 5V ±10%)*
(8088-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current: 8088 8088-2 P8088		340 350 250	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	0V ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC}+1.0	V	
C_{IN}	Capacitance if Input Buffer (All input except AD_0-AD_7, RQ/GT		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0-AD_7, RQ/GT		15	pF	fc = 1 MHz

*Note: For Extended Temperature EXPRESS V_{CC} = 5V ± 5%

A.C. CHARACTERISTICS (8088: T_A = 0°C to 70°C, V_{CC} = 5V ±10%)* (8088-2: T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

MINIMUM COMPLEXITY SYSTEM TIMING REQUIREMENTS

		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		ns	
TCLDX	Data in Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 8284 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 8284 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 8088	118		68		ns	
TCHRYX	READY Hold Time into 8088	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 3)	−8		−8		ns	
THVCH	HOLD Setup Time	35		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$ Setup Time (See Note 2)	30		15		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

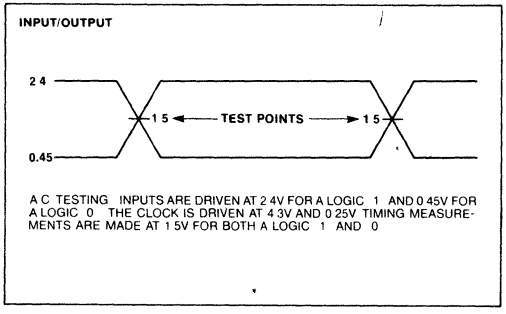
*Note: For Extended Temperature EXPRESS V_{CC} = 5V ± 5%

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

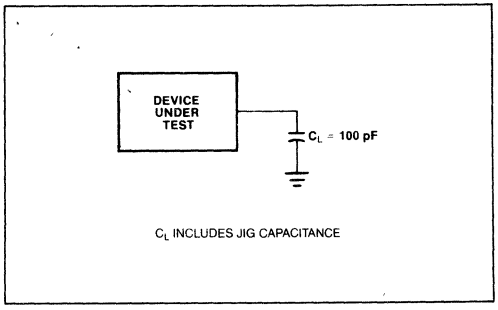
		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLAV	Address Valid Delay	10	110	10	60	ns	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	ALE Width	TCLCH−20		TCLCH−10		ns	
TCLLH	ALE Active Delay		80		50	ns	
TCHLL	ALE Inactive Delay		85		55	ns	
TLLAX	Address Hold Time to ALE Inactive	TCHCL−10		TCHCL−10		ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TWHDX	Data Hold Time After $\overline{WR}$	TCLCH−30		TCLCH−30		ns	
TCVCTV	Control Active Delay 1	10	110	10	70	ns	
TCHCTV	Control Active Delay 2	10	110	10	60	ns	
TCVCTX	Control Inactive Delay	10	110	10	70	ns	
TAZRL	Address Float to READ Active	0		0		ns	
TCLRL	$\overline{RD}$ Active Delay	10	165	10	100	ns	
TCLRH	$\overline{RD}$ Inactive Delay	10	150	10	80	ns	
TRHAV	$\overline{RD}$ Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	
TCLHAV	HLDA Valid Delay	10	160	10	100	ns	
TRLRH	$\overline{RD}$ Width	2TCLCL−75		2TCLCL−50		ns	
TWLWH	$\overline{WR}$ Width	2TCLCL−60		2TCLCL−40		ns	
TAVAL	Address Valid to ALE Low	TCLCH−60		TCLCH−40		ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TØHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 THE CLOCK IS DRIVEN AT 4 3V AND 0 25V TIMING MEASUREMENTS ARE MADE AT 1 5V FOR BOTH A LOGIC 1 AND 0

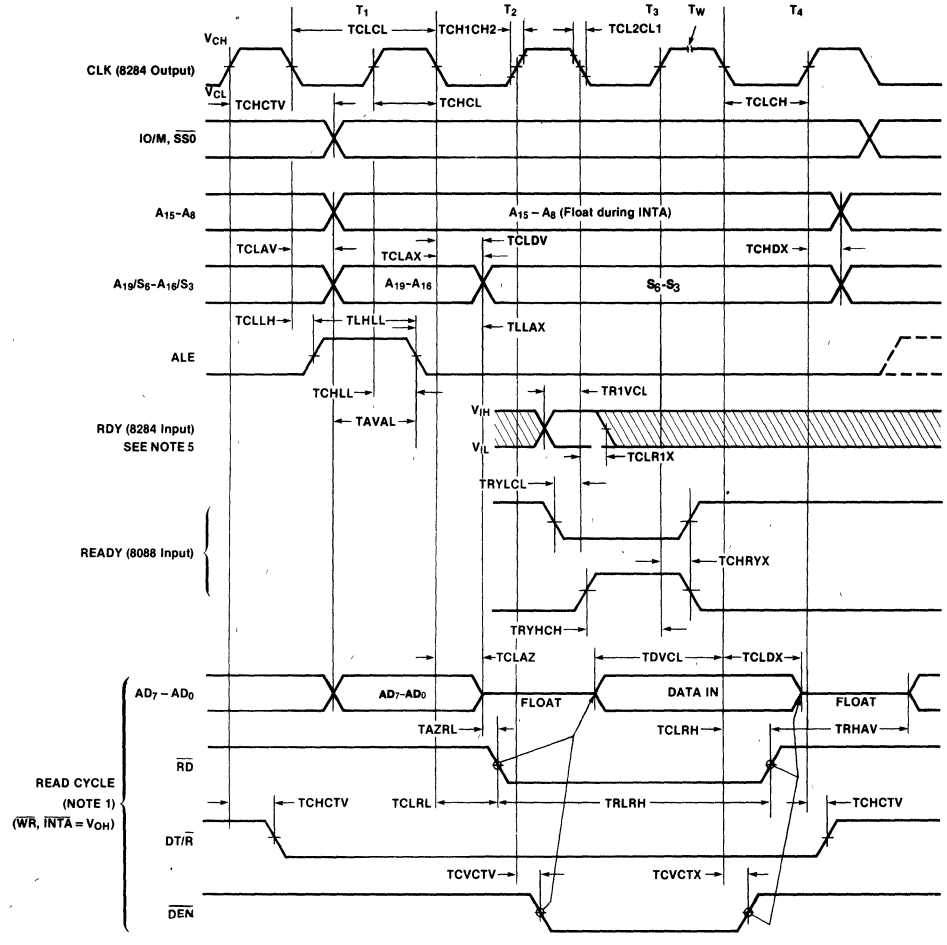
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

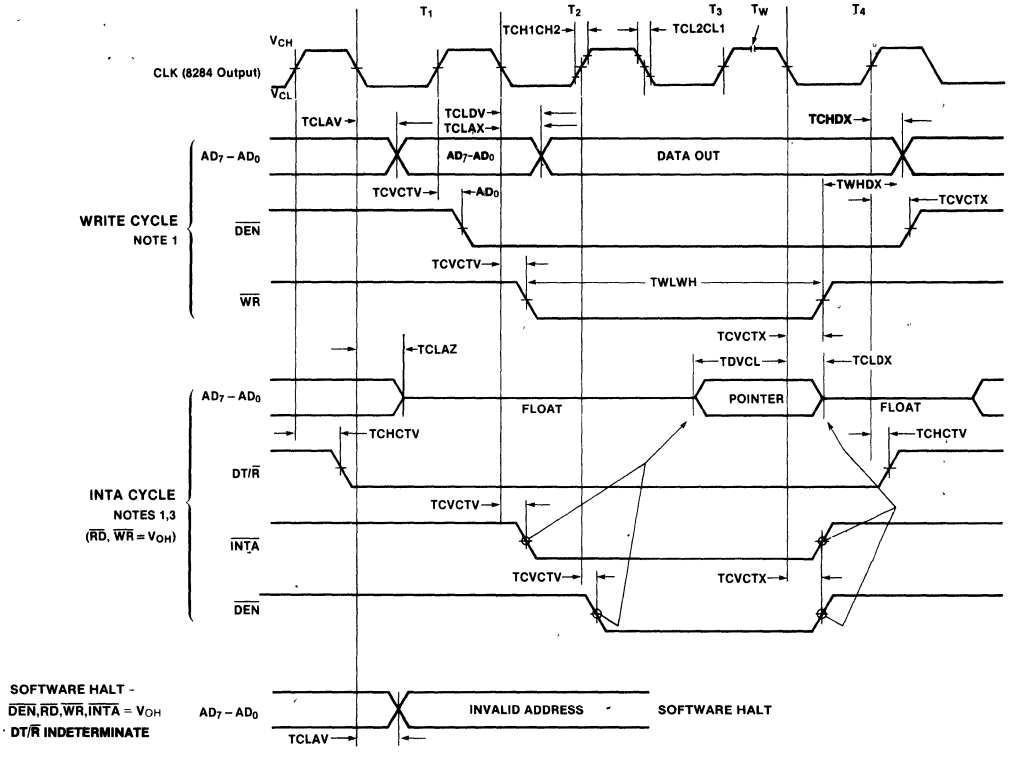
WAVEFORMS

BUS TIMING—MINIMUM MODE SYSTEM



WAVEFORMS (Continued)

BUS TIMING—MINIMUM MODE SYSTEM (Continued)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING BOTH INTA CYCLES. CONTROL SIGNALS ARE SHOWN FOR THE SECOND INTA CYCLE.
4. SIGNALS AT 8284 ARE SHOWN FOR REFERENCE ONLY.
5. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

A.C. CHARACTERISTICS

MAX MODE SYSTEM (USING 8288 BUS CONTROLLER)

TIMING REQUIREMENTS

		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		20		ns	
TCLDX	Data In Hold Time	10		10		ns	
TR1VCL	RDY Setup Time into 8284 (See Notes 1, 2)	35		35		ns	
TCLR1X	RDY Hold Time into 8284 (See Notes 1, 2)	0		0		ns	
TRYHCH	READY Setup Time into 8088	118		68		ns	
TCHRYX	READY Hold Time into 8088	30		20		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		−8		ns	
TINVCH	Setup Time for Recognition (INTR, NMI, TEST) (See Note 2)	30		15		ns	
TGVCH	RQ/GT Setup Time	30		15		ns	
TCHGX	RQ Hold Time into 8086	40		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 8284 or 8288 shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state (8 ns into T3 state).
4. Applies only to T2 state (8 ns into T3 state).

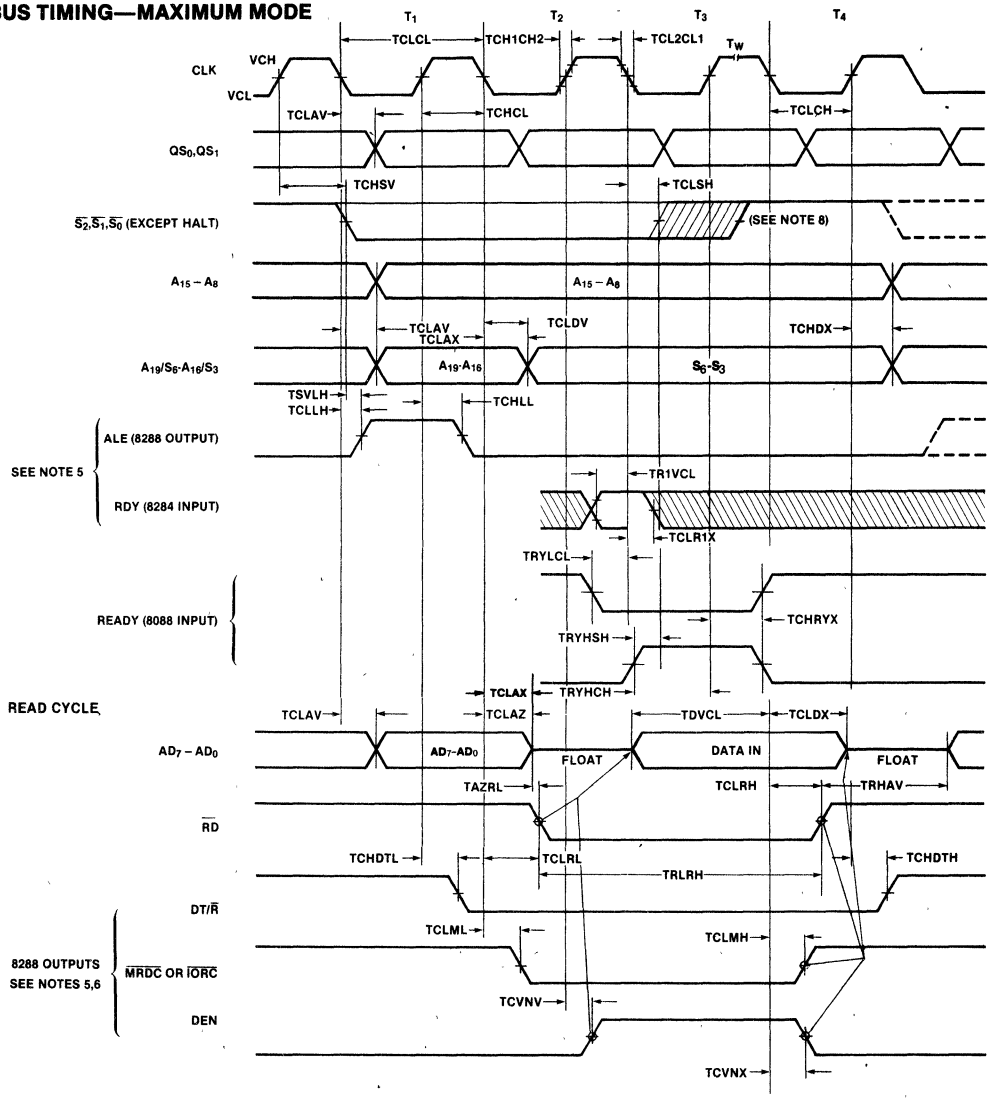
A.C. CHARACTERISTICS

TIMING RESPONSES

		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLML	Command Active Delay (See Note 1)	10	35	10	35	ns	
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110		65	ns	
TCHSV	Status Active Delay	10	110	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15	ns	
TSVMCH	Status Valid to MCE High (See Note 1)		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15	ns	
TCLMCH	CLK Low to MCE High (See Note 1)		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15	ns	
TCLMCL	MCE Inactive Delay (See Note 1)		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	ns	
TAZRL	Address Float to Read Active	0		0		ns	
TCLRL	RD Active Delay	10	165	10	100	ns	
TCLRH	RD Inactive Delay	10	150	10	80	ns	
TRHAV	RD Inactive to Next Address Active	TCLCL−45		TCLCL−40		ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30	ns	
TCLGL	GT Active Delay		85		50	ns	
TCLGH	GT Inactive Delay		85		50	ns	
TRLRH	RD Width	2TCLCL−75		2TCLCL 50		ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

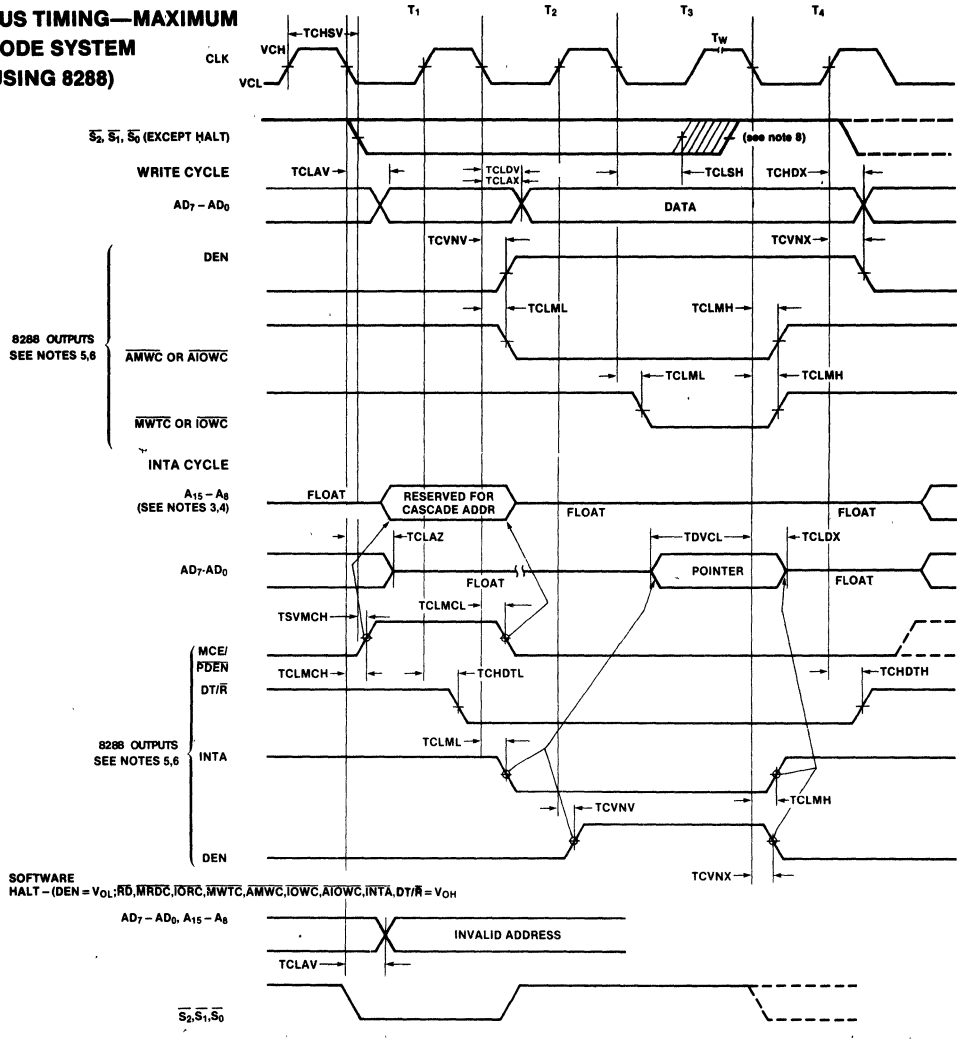
WAVEFORMS

BUS TIMING—MAXIMUM MODE



WAVEFORMS (Continued)

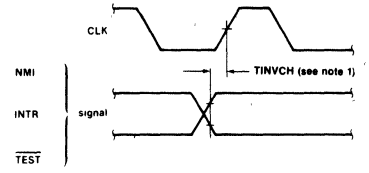
BUS TIMING—MAXIMUM MODE SYSTEM (USING 8288)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. CASCADE ADDRESS IS VALID BETWEEN FIRST AND SECOND INTA CYCLES.
4. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING BOTH INTA CYCLES. CONTROL FOR POINTER ADDRESS IS SHOWN FOR SECOND INTA CYCLE.
5. SIGNALS AT 8284 OR 8288 ARE SHOWN FOR REFERENCE ONLY
6. THE ISSUANCE OF THE 8288 COMMAND AND CONTROL SIGNALS (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
7. ALL TIMING MEASUREMENTS ARE MADE AT 1 5V UNLESS OTHERWISE NOTED
8. STATUS INACTIVE IN STATE JUST PRIOR TO T_4.

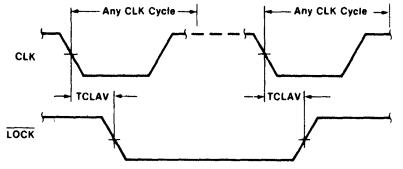
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

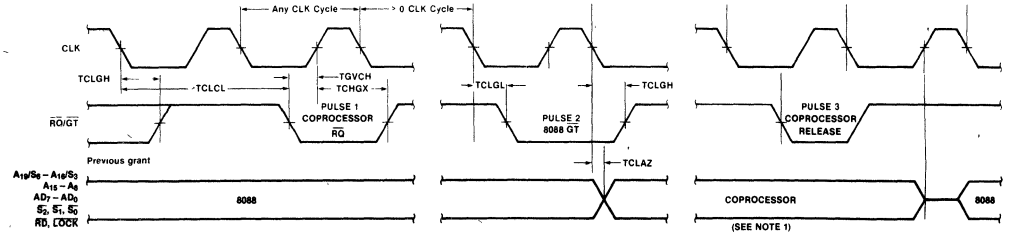


NOTE: 1. SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)

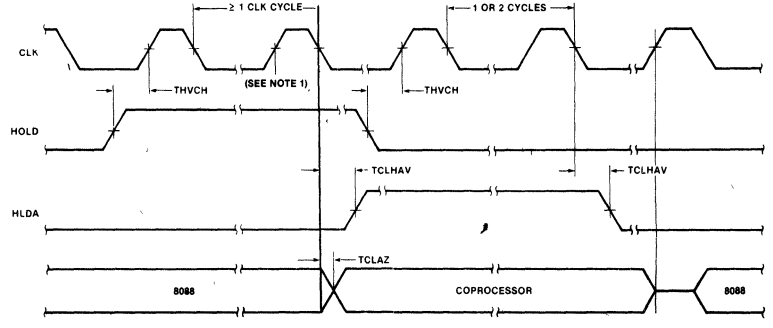


REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)



NOTE 1 THE COPROCESSOR MAY NOT DRIVE THE BUSSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION

HOLD/HOLD ACKNOWLEDGE TIMING (MINIMUM MODE ONLY)



iAPX 86/10, 88/10
INSTRUCTION SET SUMMARY

DATA TRANSFER

MOV - Move	76543210	76543210	76543210	76543210
Register/memory to/from register	100010dw	mod reg r/m		
Immediate to register/memory	1100011w	mod 000 r/m	data	data if w 1
Immediate to register	1011w reg	data	data if w 1	
Memory to accumulator	1010000w	addr low	addr high	
Accumulator to memory	1010001w	addr low	addr high	
Register/memory to segment register	10001110	mod 0 reg r/m		
Segment register to register/memory	10001100	mod 0 reg r/m		

PUSH Push		
Register/memory	11111111	mod 110 r/m
Register	01010 reg	
Segment register	000 reg 110	

POP Pop		
Register/memory	10001111	mod 000 r/m
Register	01011 reg	
Segment register	000 reg 111	

XCHG Exchange		
Register/memory with register	1000011w	mod reg r/m
Register with accumulator	10010 reg	

IN=Input from		
Fixed port	1110010w	port
Variable port	1110110w	

OUT = Output to		
Fixed port	1110011w	port
Variable port	1110111w	
XLAT=Translate byte to AL	11010111	
LEA=Load EA to register	10001101	mod reg r/m
LDS=Load pointer to DS	11000101	mod reg r/m
LES=Load pointer to ES	11000100	mod reg r/m
LAHF=Load AH with flags	10011111	
SAHF Store AH into flags	10011110	
PUSHF=Push flags	10011100	
POPF=Pop flags	10011101	

ARITHMETIC

ADD Add				
Reg./memory with register to either	000000dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 000 r/m	data	data if s w 01
Immediate to accumulator	0000010w	data	data if w 1	

ADC Add with carry				
Reg./memory with register to either	000100dw	mod reg r/m		
Immediate to register/memory	100000sw	mod 010 r/m	data	data if s w 01
Immediate to accumulator	0001010w	data	data if w 1	

INC Increment	
Register/memory	1111111w mod 000 r/m
Register	01000 reg
AAA=ASCII adjust for add	00110111
DAA=Decimal adjust for add	00100111

SUB Subtract				
Reg./memory and register to either	001010dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 101 r/m	data	data if s w 01
Immediate from accumulator	0010110w	data	data if w 1	

SBB = Subtract with borrow				
Reg./memory and register to either	000110dw	mod reg r/m		
Immediate from register/memory	100000sw	mod 011 r/m	data	data if s w 01
Immediate from accumulator	0001110w	data	data if w 1	

DEC Decrement	76543210	76543210	76543210	76543210
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG Change sign	1111011w	mod 011 r/m		

CMP Compare				
Register/memory and register	0011100w	mod reg r/m		
Immediate with register/memory	100000sw	mod 111 r/m	data	data if s w 01
Immediate with accumulator	0011110w	data	data if w 1	
AAS ASCII adjust for subtract	00111111			
DAS Decimal adjust for subtract	00101111			
MUL Multiply (unsigned)	1111011w	mod 100 r/m		
IMUL Integer multiply (signed)	1111011w	mod 101 r/m		
AAM ASCII adjust for multiply	11010100	00001010		
DIV Divide (unsigned)	1111011w	mod 110 r/m		
IDIV Integer divide (signed)	1111011w	mod 111 r/m		
AAD ASCII adjust for divide	11010101	00001010		
CBW Convert byte to word	10011000			
CWD Convert word to double word	10011001			

LOGIC

NOT Invert	1111011w	mod 010 r/m
SHL/SAL Shift logical/arithmetic left	110100vw	mod 100 r/m
SHR Shift logical right	110100vw	mod 101 r/m
SAR Shift arithmetic right	110100vw	mod 111 r/m
ROL Rotate left	110100vw	mod 000 r/m
ROR Rotate right	110100vw	mod 001 r/m
RCL Rotate through carry flag left	110100vw	mod 010 r/m
RCR Rotate through carry right	110100vw	mod 011 r/m

AND And				
Reg. memory and register to either	001000dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 100 r/m	data	data if w 1
Immediate to accumulator	0010010w	data	data if w 1	

TEST And function to flags no result				
Register/memory and register	1000010w	mod reg r/m		
Immediate data and register/memory	1111011w	mod 000 r/m	data	data if w 1
Immediate data and accumulator	1010100w	data	data if w 1	

OR Or				
Reg./memory and register to either	000010dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 001 r/m	data	data if w 1
Immediate to accumulator	0000110w	data	data if w 1	

XOR Exclusive or				
Reg./memory and register to either	001100dw	mod reg r/m		
Immediate to register/memory	1000000w	mod 110 r/m	data	data if w 1
Immediate to accumulator	0011010w	data	data if w 1	

STRING MANIPULATION

REP=Repeat	1111001z
MOVS=Move byte/word	1010010w
CMPS=Compare byte/word	1010011w
SCAS=Scan byte/word	1010111w
LODS=Load byte/wd to AL/AX	1010110w
STOS=Stor byte/wd from AL/A	1010101w

INSTRUCTION SET SUMMARY (Continued)

CONTROL TRANSFER

CALL = Call.	76543210	76543210	76543210
Direct within segment	11101000	disp low	disp high
Indirect within segment	11111111	mod 010 r/m	
Direct intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 011 r/m	

JMP = Unconditional Jump:			
Direct within segment	11101001	disp-low	disp high
Direct within segment-short	11101011	disp	
Indirect within segment	11111111	mod 100 r/m	
Direct intersegment	11101010	offset low	offset high
		seg-low	seg high
Indirect intersegment	11111111	mod 101 r/m	

RET = Return from CALL			
Within segment	11000011		
Within seg adding immed to SP	11000010	data low	data high
Intersegment	11001011		
Intersegment adding immediate to SP	11001010	data low	data high

JE/JZ=Jump on equal/zero	01110100	disp
JL/JNGE=Jump on less/not greater or equal	01111100	disp
JLE/JNG=Jump on less or equal/not greater	01111110	disp
JB/JNAE=Jump on below/not above or equal	01110010	disp
JBE/JNA=Jump on below or equal/ not above	01110110	disp
JP/JPE=Jump on parity/parity even	01111010	disp
JO=Jump on overflow	01110000	disp
JS=Jump on sign	01111000	disp
JNE/JNZ=Jump on not equal/not zero	01110101	disp
JNL/JGE=Jump on not less/greater or equal	01111101	disp
JNLE/JG=Jump on not less or equal/ greater	01111111	disp

	76543210	76543210
JNB/JAE Jump on not below/above or equal	01110011	disp
JNBE/JA Jump on not below or equal/above	01110111	disp
JNP/JPO Jump on not par/par odd	01111011	disp
JNO Jump on not overflow	01110001	disp
JNS Jump on not sign	01111001	disp
LOOP Loop CX times	11100010	disp
LOOPZ/LOOPE Loop while zero/equal	11100001	disp
LOOPNZ/LOOPNE Loop while not zero/equal	11100000	disp
JCXZ Jump on CX zero	11100011	disp

INT Interrupt		
Type specified	11001101	type
Type 3	11001100	
INTO Interrupt on overflow	11001110	
IRET Interrupt return	11001111	

PROCESSOR CONTROL

CLC Clear carry	11111000	
CMC Complement carry	11110101	
STC Set carry	11111001	
CLD Clear direction	11111100	
STD Set direction	11111101	
CLI Clear interrupt	11111010	
STI Set interrupt	11111011	
HLT Halt	11110100	
WAIT Wait	10011011	
ESC Escape (to external device)	11011xxx	mod x x x r/m
LOCK Bus lock prefix	11110000	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive,
Less = less positive (more negative) signed values
if d = 1 then "to" reg, if d = 0 then "from" reg
if w = 1 then word instruction, if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high disp-low

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1, if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

001 reg 110

REG is assigned according to the following table

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file

FLAGS = X X X X (OF) (DF) (IF) (TF) (SF) (ZF) X (AF) X (PF) X (CF)

PRELIMINARY

iAPX 188
HIGH INTEGRATION 8-BIT MICROPROCESSOR

- **Integrated Feature Set**
  - —Enhanced 8088-2 CPU
  - —Clock Generator
  - —2 Independent, High-Speed DMA Channels
  - —Programmable Interrupt Controller
  - —3 Programmable 16-bit Timers
  - —Programmable Memory and Peripheral Chip-Select Logic
  - —Programmable Wait State Generator
  - —Local Bus Controller
- **8-Bit Data Bus Interface; 16-bit internal architecture**
- **Available in 8MHz (80186) and cost effective 6 MHz (80186-6) versions**
- **High-Performance 8 MHz Processor**
  - —2 Times the Performance of the Standard iAPX 88
  - —2 MByte/Sec Bus Bandwidth Interface
- **Completely Object Code Compatible with All Existing iAPX 86, 88 Software**
  - —10 New Instruction Types
- **Direct Addressing Capability to 1 MByte of Memory**
- **Complete System Development Support**
  - —Development Software: Assembler, PL/M, Pascal, Fortran, and System Utilities
  - —In-Circuit-Emulator (ICE™ -188)
  - —iRMX™ 86, 88 Compatible (80130 OSF)
- **High Performance Numerical Coprocessing Capability Through 8087 Interface**

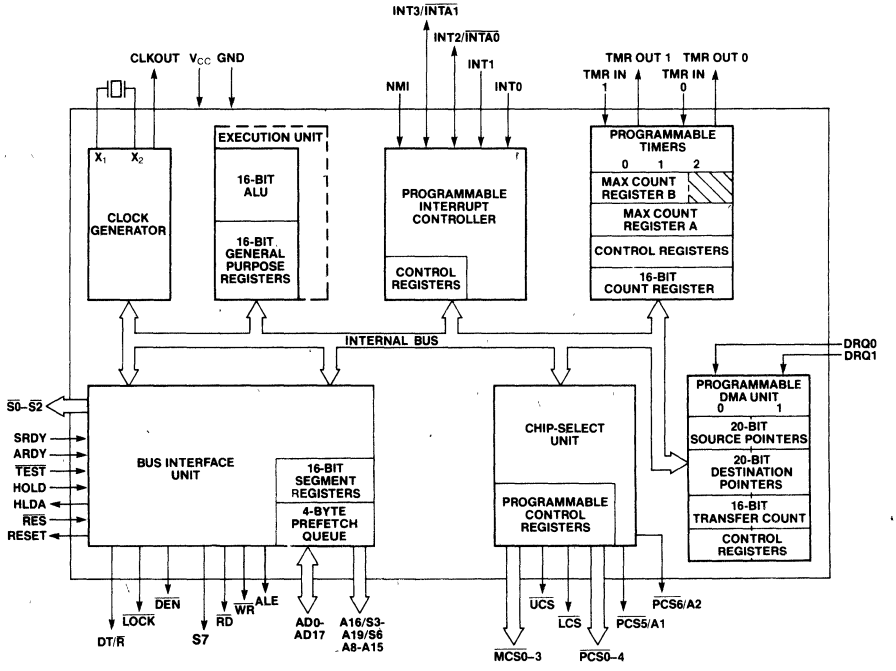


Figure 1. iAPX 188 Block Diagram

OCTOBER 1982
ORDER NUMBER: 210706-001

The Intel iAPX 188 (80188 part number) is a highly integrated microprocessor with an 8-bit data bus interface and a 16-bit internal architecture to give high performance. The iAPX 188 effectively combines 15-20 of the most common iAPX 88 system components onto one. The 80188 provides two times greater throughput than the standard 5 MHz IAPX 88. The iAPX 188 is upward compatible with iAPX 86 and 88 software and adds 10 new instruction types to the existing set.

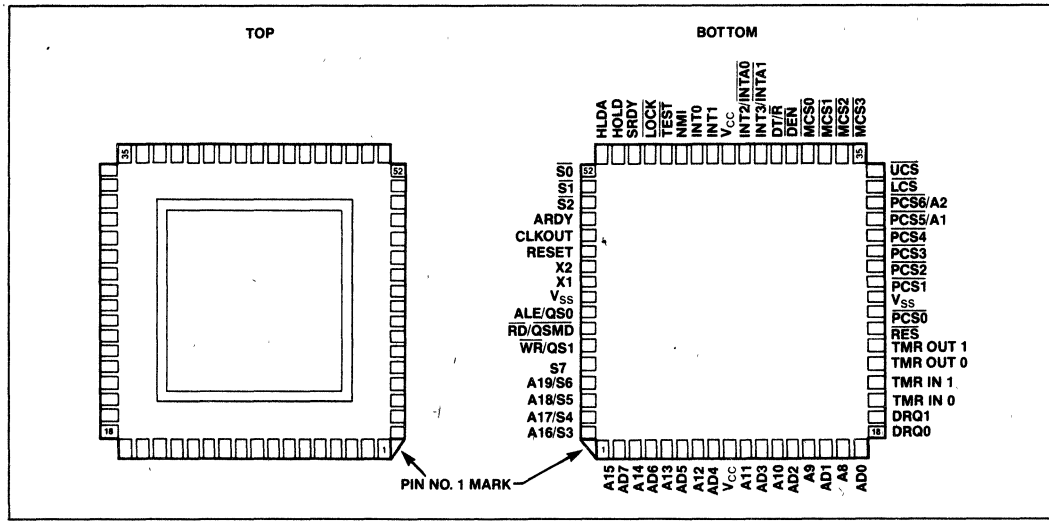


Figure 2. 80188 Pinout Diagram

Table 1. 80186 Pin Description

Symbol	Pin No.	Type	Name and Function
V_{CC}, V_{CC}	9, 43	I	System Power: + 5 volt power supply.
V_{SS}, V_{SS}	26,60	I	System Ground.
RESET	57	O	Reset Output indicates that the 80186 CPU is being reset, and can be used as a system reset. It is active HIGH, synchronized with the processor clock, and lasts an integer number of clock periods corresponding to the length of the $\overline{RES}$ signal.
X1, X2	59,58	I	Crystal Inputs, X1 and X2, provide an external connection for a fundamental mode parallel resonant crystal for the internal crystal oscillator. X1 can interface to an external clock instead of a crystal. In this case, minimize the capacitance on X2 or drive X2 with complemented X1. The input or oscillator frequency is internally divided by two to generate the clock signal (CLKOUT).
CLKOUT	56	O	Clock Output provides the system with a 50% duty cycle waveform. All device pin timings are specified relative to CLKOUT. CLKOUT has sufficient MOS drive capabilities for the 8087 Numeric Processor Extension.
$\overline{RES}$	24	I	System Reset causes the 80186 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80186 clock. The 80186 begins fetching instructions approximately 7 clock cycles after $\overline{RES}$ is returned HIGH. $\overline{RES}$ is required to be LOW for greater than 4 clock cycles and is internally synchronized. For proper initialization, the LOW-to-HIGH transition of $\overline{RES}$ must occur no sooner than 50 microseconds after power up. This input is provided with a Schmitt-trigger to facilitate power-on $\overline{RES}$ generation via an RC network. When $\overline{RES}$ occurs, the 80188 will drive the status lines to an inactive level for one clock, and then tri-state them.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{TEST}$	47	I	$\overline{TEST}$ is examined by the WAIT instruction. If the $\overline{TEST}$ input is HIGH when "WAIT" execution begins, instruction execution will suspend. $\overline{TEST}$ will be resampled until it goes LOW, at which time execution will resume. If interrupts are enabled while the 80188 is waiting for $\overline{TEST}$, interrupts will be serviced. This input is synchronized internally.
TMR IN 0, TMR IN1	20 21	I I	Timer Inputs are used either as clock or control signals, depending upon the programmed timer mode. These inputs are active HIGH (or LOW-to-HIGH transitions are counted) and internally synchronized.
TMR OUT 0, TMR OUT 1	22 23	O O	Timer outputs are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.
DRQ0 DRQ1	18 19	I I	DMA Request is driven HIGH by an external device when it desires that a DMA channel (Channel 0 or 1) perform a transfer. These signals are active HIGH, level-triggered, and internally synchronized.
NMI	46	I	Non-Maskable Interrupt is an edge-triggered input which causes a type 2 interrupt. NMI is not maskable internally. A transition from a LOW to HIGH initiates the interrupt at the next instruction boundary. NMI is latched internally. An NMI duration of one clock or more will guarantee service. This input is internally synchronized.
INT0, INT1, INT2/$\overline{INTA0}$ INT3/$\overline{INTA1}$	45,44 42 41	I I/O I/O	Maskable Interrupt Requests can be requested by strobing one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured via software to provide active-LOW interrupt-acknowledge output signals. All interrupt inputs may be configured via software to be either edge- or level-triggered. To ensure recognition, all interrupt requests must remain active until the interrupt is acknowleged. When iRMX mode is selected, the function of these pins changes (see Interrupt Controller section of this data sheet).
A19/S6, A18/S5, A17/S4, A16/S3	65–68	O O O O	Address Bus Outputs (16–19) and Bus Cycle Status (3–6) reflect the four most significant address bits during T_1. These signals are active HIGH. During T_2, T_3, T_W, and T_4, status information is available on these lines as encoded below: S3,S4, and S5 are defined as LOW during T_2–T_4

	Low	High
S6	Processor Cycle	DMA Cycle

AD7-AD0	2,4,6,8, 11,13,15,17	I/O	Address/Data Bus (0-7) signals constitute the time mutiplexed memory or I/O address (T_1) and data (T_2, T_3, T_W, and T_4) bus. The bus is active HIGH.
A15-A8	1,3,5,7 10,12,14,16	O	Address-only Bus (8-15), containing valid address from T_1-T_4. The bus is active HIGH.
S7	64	O	This signal is always HIGH to indicate that the 80188 has an 8-bit data bus, and is tri-state OFF during bus HOLD.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
ALE/QS0	61	O	Address Latch Enable/Queue Status 0 is provided by the 80188 to latch the address into the 8282/8283 address latches. ALE is active HIGH. Addresses are guaranteed to be valid on the trailing edge of ALE. The ALE rising edge is generated off the rising edge of the CLKOUT immediately preceding T_1 of the associated bus cycle, effectively one-half clock cycle earlier than in the standard 80188 The trailing edge is generated off the CLKOUT rising edge in T_1 as in the 80188 Note that ALE is never floated.
$\overline{WR}$/QS1	63	O	Write Strobe/Queue Status 1 indicates that the data on the bus is to be written into a memory or an I/O device. $\overline{WR}$ is active for T_2, T_3, and T_W of any write cycle. It is active LOW, and floats during "HOLD." It is driven HIGH for one clock during Reset, and then floated. When the 80188 is in queue status mode, the ALE/QS0 and $\overline{WR}$/QS1 pins provide information about processor/instruction queue interaction.

QS1	QS0	Queue Operation
0	0	No queue operation
0	1	First opcode byte fetched from the queue
1	1	Subsequent byte fetched from the queue
1	0	Empty the queue

$\overline{RD}$/$\overline{QSMD}$	62	O	Read Strobe indicates that the 80188 is performing a memory or I/O read cycle. $\overline{RD}$ is active LOW for T_2, T_3, and T_W of any read cycle. It is guaranteed not to go LOW in T_2 until after the Address Bus is floated. $\overline{RD}$ is active LOW, and floats during "HOLD." $\overline{RD}$ is driven HIGH for one clock during Reset, and then the output driver is floated. A weak internal pull-up mechanism on the $\overline{RD}$ line hols it HIGH when the line is not driven. During RESET the pin is sampled to determine whether the 80188 should provide ALE, $\overline{WR}$, and $\overline{RD}$, or if the Queue-Status should be provided. $\overline{RD}$ should be connected to GND to provide Queue-Status data.
ARDY	55	I	Asynchronous Ready informs the 80188 that the addressed memory space or I/O device will complete a data transfer. The ARDY input pin will accept an asynchronous input, and is active HIGH. Only the rising edge is internally synchronized by the 80188. This means that the falling edge of ARDY must be synchronized to the 80188 clock. If connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active to terminate a bus cycle. If unused, this line should be tied low.
SRDY	49	I	Synchronous Ready must be synchronized externally to the 80188. The use of SRDY provides a relaxed system-timing specification on the Ready input. This is accomplished by eliminating the one-half clock cycle which is required for internally resolving the signal level when using the ARDY input. This line is active HIGH. If this line is connected to V_{CC}, no WAIT states are inserted. Asynchronous ready (ARDY) or synchronous ready (SRDY) must be active before a bus cycle is terminated. If unused, this line should be tied low.
$\overline{LOCK}$	48	O	$\overline{LOCK}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{LOCK}$ is active LOW. The $\overline{LOCK}$ signal is requested by the LOCK prefix instruction and is activated at the beginning of the first data cycle associated with the instruction following the $\overline{LOCK}$ prefix. It remains active until the completion of the instruction following the $\overline{LOCK}$ prefix. No prefetches will occur while $\overline{LOCK}$ is asserted. $\overline{LOCK}$ is active LOW, is driven HIGH for one clock during RESET, and then floated. If unused, this line should be tied low.

Table 1. 80188 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{S0}$,$\overline{S1}$,$\overline{S2}$	52-54	O	Bus cycle status $\overline{S0}$-$\overline{S2}$ are encoded to provide bus-transaction information: 80188 Bus Cycle Status Information $\overline{S2}$ / $\overline{S1}$ / $\overline{S0}$ / Bus Cycle Initiated 0 0 0 Interrupt Acknowledge 0 0 1 Read I/O 0 1 0 Write I/O 0 1 1 Halt 1 0 0 Instruction Fetch 1 0 1 Read Data from Memory 1 1 0 Write Data to Memory 1 1 1 Passive (no bus cycle) The status pins float during "HOLD." $\overline{S2}$ may be used as a logical M/$\overline{IO}$ indicator, and $\overline{S1}$ as a DT/$\overline{R}$ indicator. The status lines are driven HIGH for one clock during Reset, and then floated until a bus cycle begins.
HOLD (input) HLDA (output)	50 51	I O	HOLD indicates that another bus master is requesting the local bus. The HOLD input is active HIGH. HOLD may be asynchronous with respect to the 80188 clock. The 80188 will issue a HLDA in response to a HOLD request at the end of T_4 or T_I. Simultaneous with the issuance of HLDA, the 80188 will float the local bus and control lines. After HOLD is detected as being LOW, the 80188 will lower HLDA. When the 80188 needs to run another bus cycle, it will again drive the local bus and control lines.
$\overline{UCS}$	34	O	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K-256K block) of memory. This line is not floated during bus HOLD. The address range activating $\overline{UCS}$ is software programmable.
$\overline{LCS}$	33	O	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K-256K) of memory. This line is not floated during bus HOLD. The address range activating $\overline{LCS}$ is software programmable.
$\overline{MCS0}$-3	38,37,36,35	O	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined mid-range portion of memory (8K-512K). These lines are not floated during bus HOLD. The address ranges activating $\overline{MCS0}$-3 are software programmable.
$\overline{PCS0}$-4	25,27-30	O	Peripheral Chip Select signals 0-4 are active LOW when a reference is made to the defined peripheral area (64K byte I/O space). These lines are not floated during bus HOLD. The address ranges activating $\overline{PCS0}$-4 are software programmable.
$\overline{PCS5}$/A1	31	O	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating $\overline{PCS5}$ is software programmable. When programmed to provide latched A1, rather than $\overline{PCS5}$, this pin will retain the previously latched value of A1 during a bus HOLD. A1 is active HIGH.
$\overline{PCS6}$/A2	32	O	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating $\overline{PCS6}$ is software programmable. When programmed to provide latched A2, rather than $\overline{PCS6}$, this pin will retain the previously latched value of A2 during a bus HOLD. A2 is active HIGH.
DT/$\overline{R}$	40	O	Data Transmit/Receive controls the direction of data flow through the external 8286/8287 data bus transceiver. When LOW, data is transferred to the 80188. When HIGH the 80188 places write data on the data bus.
$\overline{DEN}$	39	O	Data Enable is provided as an 8286/8287 data bus transceiver output enable. $\overline{DEN}$ is active LOW during each memory and I/O access. $\overline{DEN}$ is HIGH whenever DT/$\overline{R}$ changes state.

FUNCTIONAL DESCRIPTION

Introduction

The following Functional Description describes the base architecture of the iAPX 188. This architecture is common to the iAPX 86, 88, and 286 microprocessor families as well. The iAPX 186 is a very high integration 8-bit microprocessor. It combines 15-20 of the most common microprocessor system components onto one chip while providing twice the performance of the standard iAPX 88. The 80188 is object code compatible with the iAPX 86, 88 microprocessors and adds 10 new instruction types to the existing iAPX 86, 88 instruction set.

iAPX 188 BASE ARCHITECTURE

The iAPX 86, 88, 186, 188, and 286 family all contain the same basic set of registers, instructions and addressing modes. The 80188 processor is upward compatible with the 8086, 8088, 80186, and 80286 CPUs.

Register Set

The 80188 base architecture has fourteen registers as shown in Figures 3a and 3b. These registers are grouped into the following categories.

General Registers

Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used as 16-bit registers or split into pairs of separate 8-bit registers.

Segment Registers

Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers

Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode selects the specific registers for operand and address calculations.

Status and Control Registers

Two 16-bit special purpose registers record or alter certain aspects of the 80188 processor state. These are the Instruction Pointer Register, which contains the offset address of the next sequential instruction to be executed, and the Status Word Register, which contains status and control flag bits (see Figures 3a and 3b).

Status Word Description

The Status Word records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80188 within a given operating mode (bits 8, 9, and 10). The Status Word Register is 16-bits wide. The function of the Status Word bits is shown in Table 2.

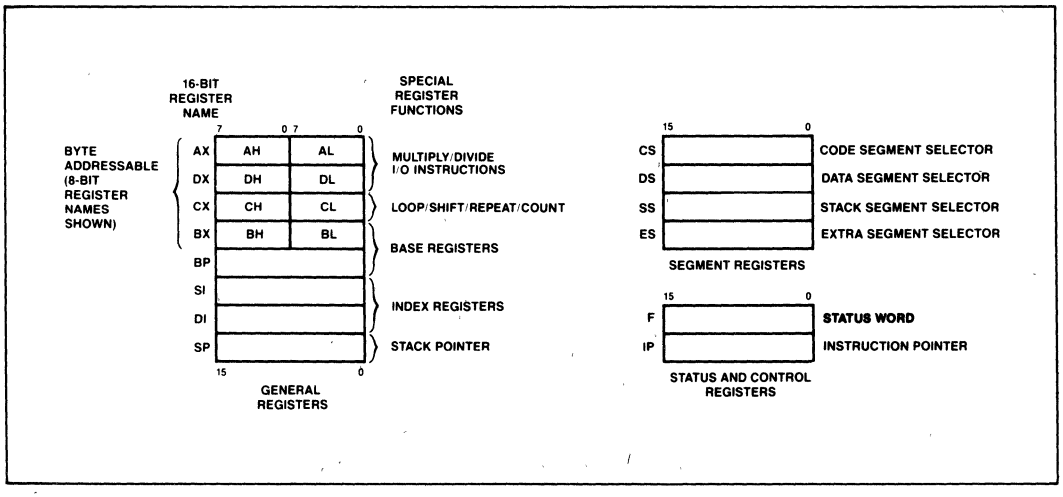


Figure 3a. 80188 General Purpose Register Set

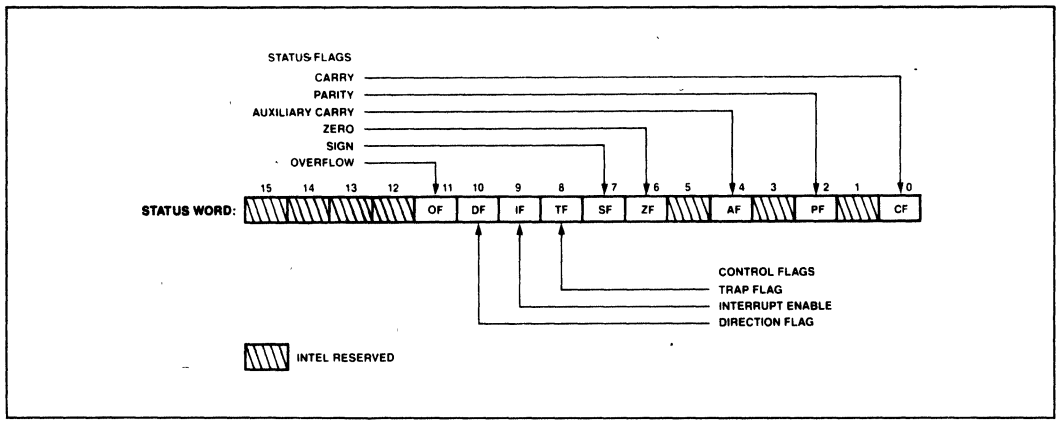


Figure 3b. Status Word Format

Table 2. Status Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag–Set if result is zero, cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index register when set. Clearing DF causes auto increment.
11	OF	Overflow Flag—Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high-level instructions, and processor control. These categories are summarized in Figure 4.

An 80188 instruction can reference anywhere from zero to several operands. An operand can reside in a register, in the instruction itself, or in memory. Specific operand addressing modes are discussed later in this data sheet.

Memory Organization

Memory is organized in sets of segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit base segment and a 16-bit offset. The 16-bit base values are contained in one of four internal segment registers (code, data, stack, extra). The physical address is calculated by shifting the base value LEFT by four bits and adding the 16-bit offset value to yield a 20-bit physical address (see Figure 5). This allows for a 1 MByte physical address size.

All instructions that address operands in memory must specify the base segment and the 16-bit offset value. For speed and compact instruction encoding, the segment register used for physical address generation is implied by the addressing mode used (see Table 3). These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs.

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
HIGH LEVEL INSTRUCTIONS	
ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4. iAPX 188 Instruction Set

AFN-01483A

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	**ITERATION CONTROLS**	
JE/JZ	Jump if equal/zero		
JG/JNLE	Jump if greater/not less nor equal	LOOP	Loop
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal/not zero	**INTERRUPTS**	
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4. iAPX 188 Instruction Set (continued)

To access operands that do not reside in one of the four immediately available segments, a full 32-bit pointer can be used to reload both the base (segment) and offset values.

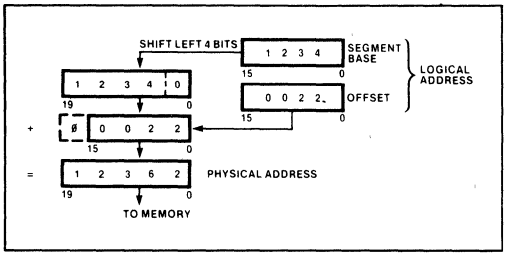


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Instruction prefetch and immediate data.
Stack	Stack (SS)	All stack pushes and pops; any memory references which use BP Register as a base register.
External Data (Global)	Extra (ES)	All string instruction references which use the DI register as an index.
Local Data	Data (DS)	All other data references.

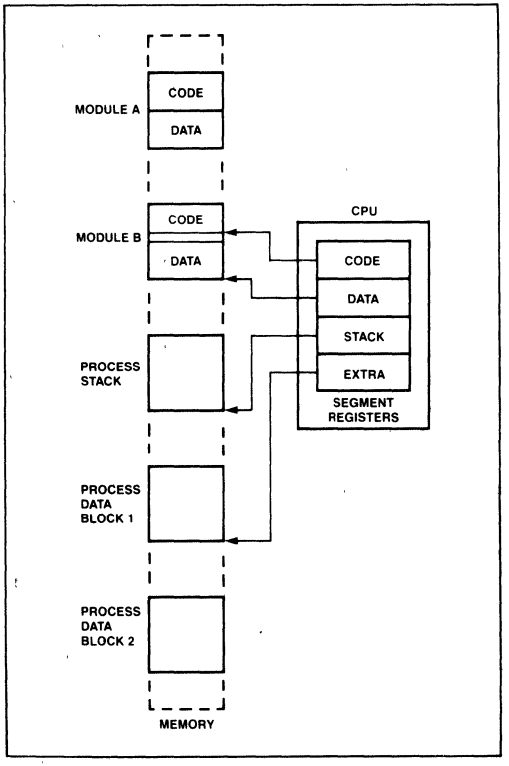


Figure 6. Segmented Memory Helps Structure Software

AFN-01483A

Addressing Modes

The 80188 provides eight categories of addressing modes to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

- *Register Operand Mode:* The operand is located in one of the 8- or 16-bit general registers.
- *Immediate Operand Mode:* The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: a segment base and an offset. The segment base is supplied by a 16-bit segment register either implicity chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset, also called the effective address, is calculated by summing any combination of the following three address elements:

- the *displacement* (an 8- or 16-bit immediate value contained in the instruction);
- the *base* (contents of either the BX or BP base registers); and
- the *index* (contents of either the SI or DI index registers).

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

- *Direct Mode:* The operand's offset is contained in the instruction as an 8- or 16-bit displacement element.
- *Register Indirect Mode:* The operand's offset is in one of the registers SI, DI, BX, or BP.
- *Based Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of a base register (BX or BP).
- *Indexed Mode:* The operand's offset is the sum of an 8- or 16-bit displacement and the contents of an index register (SI or DI).
- *Based Indexed Mode:* The operand's offset is the sum of the contents of a base register and an index register.
- *Based Indexed Mode with Displacement:* The operand's offset is the sum of a base register's contents, an index register's contents, and an 8- or 16-bit displacement.

Data Types

The 80188 directly supports the following data types:

- *Integer:* A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32- and 64-bit integers are supported using the iAPX 188/20 Numeric Data Processor.
- *Ordinal:* An unsigned binary numeric value contained in an 8-bit byte or a 16-bit word.
- *Pointer:* A 16- or 32-bit quantity, composed of a 16-bit offset component or a 16-bit segment base component in addition to a 16-bit offset component.
- *String:* A contiguous sequence of bytes or words. A string may contain from 1 to 64K bytes.
- *ASCII:* A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
- *BCD:* A byte (unpacked) representation of the decimal digits 0–9.
- *Packed BCD:* A byte (packed) representation of two decimal digits (0–9). One digit is stored in each nibble (4-bits) of the byte.
- *Floating Point:* A signed 32-, 64-, or 80-bit real number representation. (Floating point operands are supported using the iAPX 188/20 Numeric Data Processor configuration.)

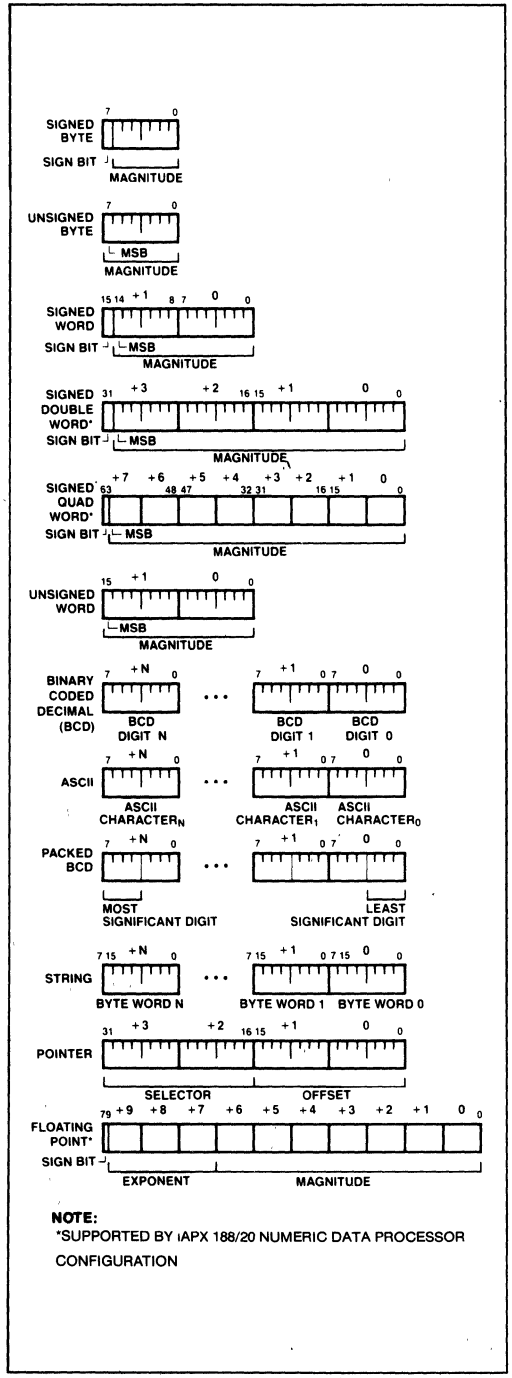
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the iAPX 188.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. Separate instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}-A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Status Word) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable.



NOTE:
*SUPPORTED BY iAPX 188/20 NUMERIC DATA PROCESSOR CONFIGURATION

Figure 7. iAPX 188 Supported Data Types

Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. If the exception was caused by executing an ESC instruction with the ESC trap bit set in the relocation register, the return instruction will point to the ESC instruction, or to the segment override prefix immediately preceding the ESC instruction if the prefix was present. In all other cases, the return address from an exception will point at the instruction immediately following the instruction causing the exception.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. Table 4 shows the 80188 predefined types and default priority levels. For each interrupt, an 8-bit vector must be supplied to the 80188 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. In addition, internal peripherals and non-cascaded external interrupts will generate their own vectors through the internal interrupt controller. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

Interrupt Sources

The 80188 can service interrupts generated by software or hardware. The software interrupts are generated by specific instructions (INT, ESC, unused OP, etc.) or the results of conditions specified by instructions (array bounds check, INT0, DIV, IDIV, etc.). All interrupt sources are serviced by an indirect call through an element of a vector table. This vector table is indexed by using the interrupt vector type (Table 4), multiplied by four. All hardware-generated interrupts are sampled at the end of each instruction. Thus, the software interrupts will begin service first. Once the service routine is entered and interrupts are enabled, any hardware source of sufficient priority can interrupt the service routine in progress.

The software generated 80188 interrupts are described below.

DIVIDE ERROR EXCEPTION (TYPE 0)
Generated when a DIV or IDIV instruction quotient cannot be expressed in the number of bits in the destination.

Table 4. 80188 Interrupt Vectors

Interrupt Name	Vector Type	Default Priority	Related Instructions
Divide Error Exception	0	*1	DIV, IDIV
Single Step Interrupt	1	12**2	All
NMI	2	1	All
Breakpoint Interrupt	3	*1	INT
INT0 Detected Overflow Exception	4	*1	INT0
Array Bounds Exception	5	*1	BOUND
Unused-Opcode Exception	6	*1	Undefined Opcodes
ESC Opcode Exception	7	*1***	ESC Opcodes
Timer 0 Interrupt	8	2A****	
Timer 1 Interrupt	18	2B****	
Timer 2 Interrupt	19	2C****	
Reserved	9	3	
DMA 0 Interrupt	10	4	
DMA 1 Interrupt	11	5	
INT0 Interrupt	12	6	
INT1 Interrupt	13	7	
INT2 Interrupt	14	8	
INT3 Interrupt	15	9	

NOTES:

*1. These are generated as the result of an instruction execution.

**2 This is handled as in the 8088

****3 All three timers constitute one source of request to the interrupt controller The Timer interrupts all have the same default priority level with respect to all other interrupt sources. However, they have a defined priority ordering amongst themselves (Priority 2A is higher priority than 2B.) Each Timer interrupt has a separate vector type number

4. Default priorities for the interrupt sources are used only if the user does not program each source into a unique priority level

***5. An escape opcode will cause a trap only if the proper bit is set in the peripheral control block relocation register.

SINGLE-STEP INTERRUPT (TYPE 1)
Generated after most instructions if the TF flag is set. Interrupts will not be generated after prefix instructions (e.g., REP), instructions which modify segment registers (e.g., POP DS), or the WAIT instruction.

NON-MASKABLE INTERRUPT—NMI (TYPE 2)
An external interrupt source which cannot be masked.

BREAKPOINT INTERRUPT (TYPE 3)
A one-byte version of the INT instruction. It uses 12 as an index into the service routine address table (because it is a type 3 interrupt).

INT0 DETECTED OVERFLOW EXCEPTION (TYPE 4)
Generated during an INT0 instruction if the 0F bit is set.

ARRAY BOUNDS EXCEPTION (TYPE 5)
Generated during a BOUND instruction if the array index is outside the array bounds. The array bounds are located in memory at a location indicated by one of the instruction operands. The other operand indicates the value of the index to be checked.

UNUSED OPCODE EXCEPTION (TYPE 6)
Generated if execution is attempted on undefined opcodes.

ESCAPE OPCODE EXCEPTION (TYPE 7)
Generated if execution is attempted of ESC opcodes (D8H–DFH). This exception will only be generated if a bit in the relocation register is set. The return address of this exception will point to the ESC instruction causing the exception. If a segment override prefix preceded the ESC instruction, the return address will point to the segment override prefix.

Hardware-generated interrupts are divided into two groups: maskable interrupts and non-maskable interrupts. The 80188 provides maskable hardware interrupt request pins INT0–INT3. In addition, maskable interrupts may be generated by the 80188 integrated DMA controller and the integrated timer unit. The vector types for these interrupts is shown in Table 4. Software enables these inputs by setting the interrupt flag bit (IF) in the Status Word. The interrupt controller is discussed in the peripheral section of this data sheet.

Further maskable interrupts are disabled while servicing an interrupt because the IF bit is reset as part of the response to an interrupt or exception. The saved Status Word will reflect the enable status of the processor prior to the interrupt. The interrupt flag will remain zero unless specifically set. The interrupt return instruction restores the Status Word, thereby restoring the original status of IF bit. If the interrupt return re-enables interrupts, and another interrupt is pending, the 80188 will immediately service the highest-priority interrupt pending, i.e., no instructions of the main line program will be executed.

Non-Maskable Interrupt Request (NMI)

A non-maskable interrupt (NMI) is also provided. This interrupt is serviced regardless of the state of the IF bit. A typical use of NMI would be to activate a power failure routine. The activation of this input

causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed. The IF bit is cleared at the beginning of an NMI interrupt to prevent maskable interrupts from being serviced.

Single-Step Interrupt

The 80188 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single-step interrupt and is controlled by the single-step flag bit (TF) in the Status Word. Once this bit is set, an internal single-step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single-stepped.

Initialization and Processor Reset

Processor initialization or startup is accomplished by driving the $\overline{RES}$ input pin LOW. $\overline{RES}$ forces the 80188 to terminate all execution and local bus activity. No instructionor bus activity will occur as long as $\overline{RES}$ is active. After $\overline{RES}$ becomes inactive and an internal processing interval elapses, the 80188 begins execution withthe instruction at physical location FFFF0(H). RES also sets some registers to predefined values as shown in Table 5.

Table 5. 80188 Initial Register State after RESET

Status Word	F002(H)
Instruction Pointer	0000(H)
Code Segment	FFFF(H)
Data Segment	0000(H)
Extra Segment	0000(H)
Stack Segment	0000(H)
Relocation Register	20FF(H)
UMCS	FFFB(H)

THE 80188 COMPARED TO THE 80186

The 80188 CPU is an 8-bit processor designed around the 80186 internal structure. Most internal functions of the 80188 are identical to the equivalent 80186 functions. The 80188 handles the external bus the same way the 80186 does with the distinction of handling only 8 bits at a time. Sixteen bit operands are fetched or written in two consecutive bus cycles. Both processors will appear identical to the software engineer, with the exception of execution time. The internal register structure is identical and all instructions have the same end result. The differences between the 80188 and 80186 are outlined below. Internally, there are three differences between the 80188 and the 80186. All changes are related to the 8-bit bus interface.

- The queue length is 4 bytes in the 80188, whereas the 80186 queue contains 6 bytes, or three words. The queue was shortened to prevent overuse of the bus by the BIU when prefetching instructions. This was required because of the additional time necessary to fetch instructions 8 bits at a time.
- To further optimize the queue, the prefetching algorithm was changed. The 80188 BIU will fetch a new instruction to load into the queue each time there is a 1-byte hole (space available) in the queue. The 80186 waits until a 2-byte space is available.
- The internal execution time of the instruction is affected by the 8-bit interface. All 16-bit fetches and writes from/to memory take an additional four clock cycles. The CPU may also be limited by the speed of instruction fetches when a series of simple operations occur. When the more sophisticated instructions of the 80188 are being used, the queue has time to fill and the execution proceeds as fast as the execution unit will allow.

The 80188 and 80186 are completely software compatible by virture of their identical execution units. Software that is system dependent may not be completely transferable, but software that is not system dependent will operate equally well on an 80188 or an 80186.

The hardware interface of the 80188 contains the major differences between the two CPUs. The pin assignments are nearly identical, however, with the following functional changes.

- A8-A15—These pins are only address outputs on the 80188. These address lines are latched internally and remain valid throughout a bus cycle in a manner similar to the 8085 upper address lines.
- $\overline{BHE}$ has no meaning on the 80188 and has been eliminated.

iAPX 188 CLOCK GENERATOR

The iAPX 188 provides an on-chip clock generator for both internal and external clock generation. The clock generator features a crystal oscillator, a divide-by-two counter, synchronous and asynchronous ready inputs, and reset circuitry.

Oscillator

The oscillator circuit of the iAPX 188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the iAPX 186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the iAPX 188. The recommended crystal configuration is shown in Figure 8.

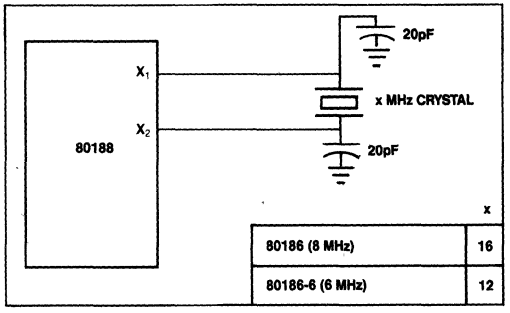


Figure 8. Recommended iAPX 188 Crystal Configuration

Clock Generator

The iAPX 188 clock generator provides the 50% duty cycle processor clock for the iAPX 188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the iAPX 188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The iAPX 188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_w until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only onthe rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3 or T_w. HIGH-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_w until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the iAPX 188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The iAPX 188 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the iAPX 188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple iAPX 188 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

Oscillator

The oscillator circuit of the iAPX 188 is designed to be used with a parallel resonant fundamental mode crystal. This is used as the time base for the iAPX 186. The crystal frequency selected will be double the CPU clock frequency. Use of an LC or RC circuit is not recommended with this oscillator. If an external oscillator is used, it can be connected directly to input pin X1 in lieu of a crystal. The output of the oscillator is not directly available outside the iAPX 188. The recommended crystal configuration is shown in Figure 8.

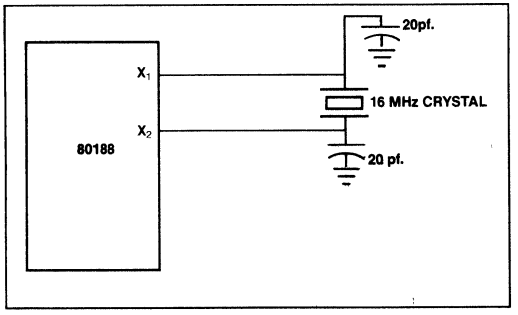


Figure 8. Recommended iAPX 188 Crystal Configuration

Clock Generator

The iAPX 188 clock generator provides the 50% duty cycle processor clock for the iAPX 188. It does this by dividing the oscillator output by 2 forming the symmetrical clock. If an external oscillator is used, the state of the clock generator will change on the falling edge of the oscillator signal. The CLKOUT pin provides the processor clock signal for use outside the iAPX 188. This may be used to drive other system components. All timings are referenced to the output clock.

READY Synchronization

The iAPX 188 provides both synchronous and asynchronous ready inputs. Asynchronous ready synchronization is accomplished by circuitry which samples ARDY in the middle of T_2, T_3 and again in the middle of each T_W until ARDY is sampled HIGH. One-half CLKOUT cycle of resolution time is used. Full synchronization is performed only onthe rising edge of ARDY, i.e., the falling edge of ARDY must be synchronized to the CLKOUT signal if it will occur during T_2, T_3 or T_W. HIGH-to-LOW transitions of ARDY must be performed synchronously to the CPU clock.

A second ready input (SRDY) is provided to interface with externally synchronized ready signals. This input is sampled at the end of T_2, T_3 and again at the end of each T_W until it is sampled HIGH. By using this input rather than the asynchronous ready input, the half-clock cycle resolution time penalty is eliminated.

This input must satisfy set-up and hold times to guarantee proper operation of the circuit.

In addition, the iAPX 188, as part of the integrated chip-select logic, has the capability to program WAIT states for memory and peripheral blocks. This is discussed in the Chip Select/Ready Logic description.

RESET Logic

The iAPX 188 provides both a $\overline{RES}$ input pin and a synchronized RESET pin for use with other system components. The $\overline{RES}$ input pin on the iAPX 188 is provided with hysteresis in order to facilitate power-on Reset generation via an RC network. RESET is guaranteed to remain active for at least five clocks given a $\overline{RES}$ input of at least six clocks. RESET may be delayed up to two and one-half clocks behind $\overline{RES}$.

Multiple iAPX 188 processors may be synchronized through the $\overline{RES}$ input pin, since this input resets both the processor and divide-by-two internal counter in the clock generator. In order to insure that the divide-by-two counters all begin counting at the same time, the active going edge of $\overline{RES}$ must satisfy a 25 ns setup time before the falling edge of the 80188 clock input. In addition, in order to insure that all CPUs begin executing in the same clock cycle, the reset must satisfy a 25 ns setup time before the rising edge of the CLKOUT signal of all the processors.

INTERNAL PERIPHERAL INTERFACE

All the iAPX 188 integrated peripherals are controlled via 16-bit registers contained within an internal 256-byte control block. This control block may be mapped into either memory or I/O space. Internal logic will recognize the address and respond to the bus cycle. During bus cycles to internal registers, the bus controller will signal the operation externally (i.e., the $\overline{RD}$, $\overline{WR}$, status, address, data, etc., lines will be driven as in a normal bus cycle), but D_{7-0}, SRDY, and ARDY will be ignored. The base address of the control block must be on an even 256-byte boundary (i.e., the lower 8 bits of the base address are all zeros). All of the defined registers within this control block may be read or written by the 80188 CPU at any time. The location of any register contained within the 256-byte control block is determined by the current base address of the control block.

The control block base address is programmed via a 16-bit relocation register contained within the control block at offset FEH from the base address of the control block (see Figure 9). It provides the upper 12 bits of the base address of the control block. Note that mapping the control register block into an address range corresponding to a chip-select range is not recommended (the chip select circuitry is discussed later in this data sheet). In addition, bit 12 of this register determines whether the control block will be mapped into I/O or memory space. If this bit is 1, the control block will be located in memory space, whereas if the bit is 0, the control block will be located in I/O space. If the control register block is mapped into I/O space, the upper 4 bits of the base address must be programmed as 0 (since I/O addresses are only 16 bits wide).

In addition to providing relocation information for the control block, the relocation register contains bits which place the interrupt controller into iRMX mode, and cause the CPU to interrupt upon encountering ESC instructions. At RESET, the relocation register is set to 20FFH. This causes the control block to start at FF00H in I/O space. An offset map of the 256-byte control register block is shown in Figure 10.

The integrated iAPX 188 peripherals operate semi-autonomously from the CPU. Access to them for the most part is via software read/write of the control and data locations in the control block. Most of these registers can be both read and written. A few dedicated lines, such as interrupts and DMA request provide real-time communication between the CPU and peripherals as in a more conventional system utilizing discrete peripheral blocks. The overall interaction and function of the peripheral blocks has not substantially changed. The data access from/to the 256-byte internal control block will always be 16-bit and done in one bus cycle.

CHIP-SELECT/READY GENERATION LOGIC

The iAPX 188 contains logic which provides programmable chip-select generation for both memories and peripherals. In addition, it can be programmed to provide READY (or WAIT state) generation. It can also provide latched address bits A1 and A2. The chip-select lines are active for all memory and I/O cycles in their programmed areas, whether they be generated by the CPU or by the integrated DMA unit.

Memory Chip Selects

The iAPX 188 provides 6 memory chip select outputs for 3 address areas: upper memory, lower memory, and midrange memory. One each is provided for upper memory and lower memory, while four are provided for midrange memory.

The range for each chip select is user-programmable and can be set to 2K, 4K, 8K, 16K, 32K, 64K, 128K (plus 1K and 256K for upper and lower chip selects). In addition, the beginning or base address of the midrange memory chip select may also be selected. Only one chip select may be programmed to be active for any memory location at a time. All chip select sizes are in bytes.

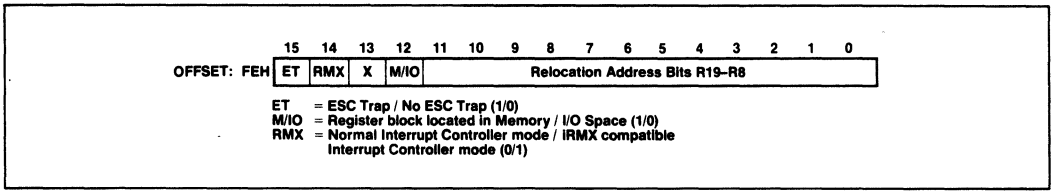


Figure 9. Relocation Register

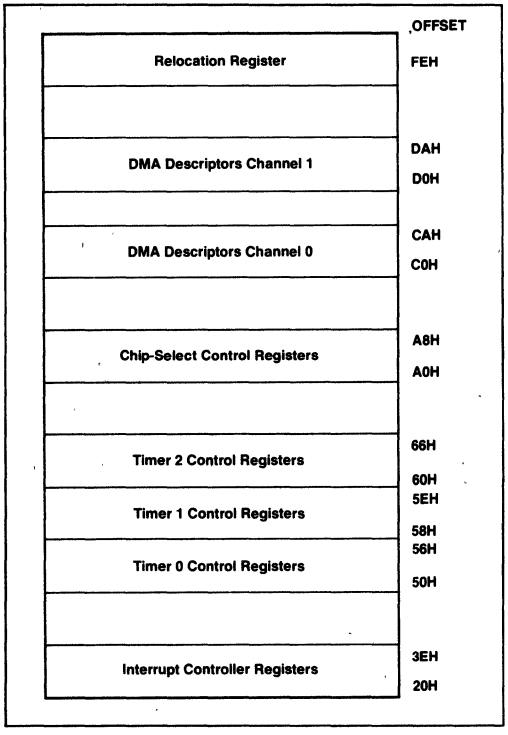


Figure 10. Internal Register Map

Upper Memory $\overline{CS}$

The iAPX 188 provides a chip select, called $\overline{UCS}$, for the top of memory. The top of memory is usually used as the system memory because after reset the iAPX 188 begins executing at memory location FFFF0H.

The upper limit of memory defined by this chip select is always FFFFFH, while the lower limit is programmable. By programming the lower limit, the size of the select block is also defined. Table 7 shows the relationship between the base address selected and the size of the memory block obtained.

Table 7. UMCS Programming Values

Starting Address (Base Address)	Memory Block Size	UMCS Value (Assuming R0=R1=R2=0)
FFC00	1K	FFF8H
FF800	2K	FFB8H
FF000	4K	FF38H
FE000	8K	FE38H
FC000	16K	FC38H
F8000	32K	F838H
F0000	64K	F038H
E0000	128K	E038H
C0000	256K	C038H

The lower limit of this memory block is defined in the UMCS register (see Figure 11). This register is at offset A0H in the internal control block. The legal values for bits 6–13 and the resulting starting address and memory block sizes are given in Table 7. Any combination of bits 6–13 not shown in Table 7 will result in undefined operation. After reset, the UMCS register is programmed for a 1K area. It must be reprogrammed if a larger upper memory area is desired.

Any internally generated 20-bit address whose upper 16 bits are greater than or equal to UMCS (with bits 0–5 "0") will cause UCS to be activated. UMCS bits R2–R0 are used to specify READY mode for the area of memory defined by this chip-select register, as explained below.

Lower Memory $\overline{CS}$

The iAPX 188 provides a chip select for low memory called $\overline{LCS}$. The bottom of memory contains the interrupt vector table, starting at location 00000H.

The lower limit of memory defined by this chip select is always 0H, while the upper limit is programmable. By programming the upper limit, the size of the memory block is also defined. Table 8 shows the relationship between the upper address selected and the size of the memory block obtained.

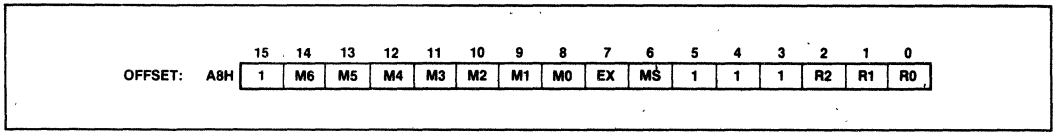


Figure 13. MPCS Register

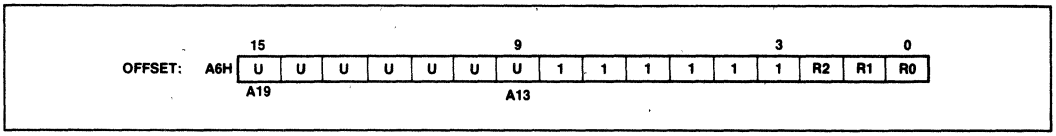


Figure 14. MMCS Register

MMCS bits R2–R0 specify READY mode of operation for all mid-range chip selects. All devices in mid-range memory must use the same number of WAIT states.

The 512K block size for the mid-range memory chip selects is a special case. When using 512K, the base address would have to be at either locations 00000H or 80000H. If it were to be programmed at 00000H when the $\overline{LCS}$ line was programmed, there would be an internal conflict between the $\overline{LCS}$ ready generation logic and the $\overline{MCS}$ ready generation logic. Likewise, if the base address were programmed at 80000H, there would be a conflict with the $\overline{UCS}$ ready generation logic. Since the $\overline{LCS}$ chip-select line does not become active until programmed, while the $\overline{UCS}$ line is active at reset, the memory base can be set only at 00000H. If this base address is selected, however, the $\overline{LCS}$ range must not be programmed.

Peripheral Chip Selects

The iAPX 188 can generate chip selects for up to seven peripheral devices. These chip selects are active for seven contiguous blocks of 128 bytes above a programmable base address. This base address may be located in either memory or I/O space.

Seven $\overline{CS}$ lines called $\overline{PCS0}$–6 are generated by the iAPX 188. The base address is user-programmable; however it can only be a multiple of 1K bytes, i.e., the least significant 10 bits of the starting address are always 0.

$\overline{PCS5}$ and $\overline{PCS6}$ can also be programmed to provide latched address bits A1, A2. If so programmed, they cannot be used as peripheral selects. These outputs can be connected directly to the A0, A1 pins used for selecting internal registers of 8-bit peripheral chips. This scheme simplifies the hardware interface because the 8-bit registers of peripherals are simply treated as 16-bit registers located on even boundaries in I/O space or memory space where only the lower 8-bits of the register are significant: the upper 8-bits are "don't cares."

The starting address of the peripheral chip-select block is defined by the PACS register (see Figure 15). This register is located at offset A4H in the internal control block. Bits 15–6 of this register correspond to bits 19–10 of the 20-bit Programmable Base Address (PBA) of the peripheral chip-select block. Bits 9–0 of the PBA of the peripheral chip-select block are all zeros. If the chip-select block is located in I/O space, bits 12–15 must be programmed zero, since the I/O address is only 16 bits wide. Table 10 shows the address range of each peripheral chip select with respect to the PBA contained in PACS register.

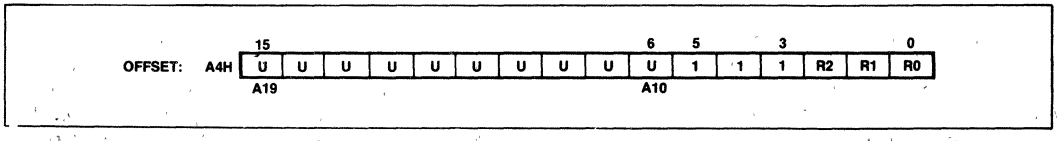


Figure 15. PACS Register

The user should program bits 15–6 to correspond to the desired peripheral base location. PACS bits 0–2 are used to specify READY mode for $\overline{PCS0}$–$\overline{PCS3}$.

Table 10. PCS Address Ranges

PCS Line	Active between Locations
PCS0	PBA — PBA+127
PCS1	PBA+128 — PBA+255
PCS2	PBA+256 — PBA+383
PCS3	PBA+384 — PBA+511
PCS4	PBA+512 — PBA+639
PCS5	PBA+640 — PBA+767
PCS6	PBA+768 — PBA+895

The mode of operation of the peripheral chip selects is defined by the MPCS register (which is also used to set the size of the mid-range memory chip-select block, see Figure 16). This register is located at offset A8H in the internal control block. Bit 7 is used to select the function of $\overline{PCS5}$ and $\overline{PCS6}$, while bit 6 is used to select whether the peripheral chip selects are mapped into memory or I/O space. Table 11 describes the programming of these bits. After reset, the contents of both the MPCS and the PACS registers are undefined, however none of the PCS lines will be active until both of the MPCS and PACS registers are accessed.

Table 11. MS, EX Programming Values

Bit	Description
MS	1 = Peripherals mapped into memory space. 0 = Peripherals mapped into I/O space.
EX	0 = 5 $\overline{PCS}$ lines. A1, A2 provided. 1 = 7 $\overline{PCS}$ lines. A1, A2 are not provided.

MPCS bits 0–2 are used to specify READY mode for $\overline{PCS4}$–$\overline{PCS6}$ as outlined below.

READY Generation Logic

The iAPX 188 can generate a "READY" signal internally for each of the memory or peripheral $\overline{CS}$ lines. The number of WAIT states to be inserted for each peripheral or memory is programmable to provide 0–3 wait states for all accesses to the area for which the chip select is active. In addition, the iAPX 188 may be programmed to either ignore external READY for each chip-select range individually or to factor external READY with the integrated ready generator.

READY control consists of 3 bits for each $\overline{CS}$ line or group of lines generated by the iAPX 188. The interpretation of the ready bits is shown in Table 12.

Table 12. READY Bits Programming

R2	R1	R0	Number of WAIT States Generated
0	0	0	0 wait states, external RDY also used.
0	0	1	1 wait state inserted, external RDY also used.
0	1	0	2 wait states inserted, external RDY also used.
0	1	1	3 wait states inserted, external RDY also used.
1	0	0	0 wait states, external RDY ignored.
1	0	1	1 wait state inserted, external RDY ignored.
1	1	0	2 wait states inserted, external RDY ignored.
1	1	1	3 wait states inserted, external RDY ignored.

The internal ready generator operates in parallel with external READY, not in series if the external READY is used (R2 = 0). This means, for example, if the internal generator is set to insert two wait states, but activity on the external READY lines will insert four wait states, the processor will only insert four wait states, not six. This is because the two wait states generated by the internal generator overlapped the first two wait states generated by the external ready signal. Note that the external ARDY and SRDY lines are always ignored during cycles accessing internal peripherals.

R2–R0 of each control word specifies the READY mode for the corresponding block, with the exception of the peripheral chip selects: R2–R0 of PACS set the $\overline{PCS0}$–3 READY mode, R2–R0 of MPCS set the $\overline{PCS4}$–6 READY mode.

Chip Select/Ready Logic and Reset

Upon reset, the Chip-Select/Ready Logic will perform the following actions:

- All chip-select outputs will be driven HIGH.
- Upon leaving RESET, the $\overline{UCS}$ line will be programmed to provide chip selects to a 1K block with the accompanying READY control bits set at 011 to

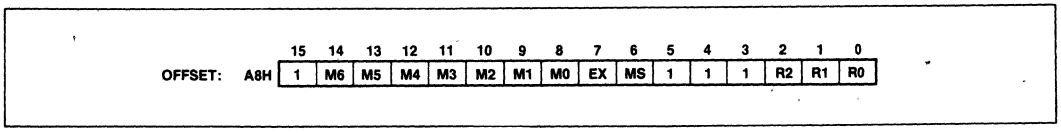


Figure 16. MPCS Register

AFN-01483A

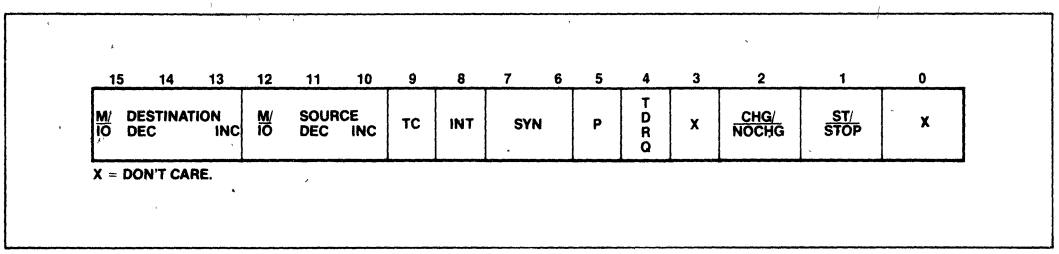


Figure 18. DMA Control Register

DMA Channel Control Word Register

Each DMA Channel Control Word determines the mode of operation for the particular 80188 DMA channel. This register specifies:

- the mode of synchronization;
- whether interrupts will be generated after the last transfer;
- whether DMA activity will cease after a programmed number of DMA cycles;
- the relative priority of the DMA channel with respect to the other DMA channel;
- whether the source pointer will be incremented, decremented, or maintained constant after each transfer;
- whether the source pointer addresses memory or I/O space;
- whether the destination pointer will be incremented, decremented, or maintained constant after each transfer; and
- whether the destination pointer will address memory or I/O space.

The DMA channel control registers may be changed while the channel is operating. However, any changes made during operation will affect the current DMA transfer.

DMA Control Word Bit Descriptions

ST/$\overline{\text{STOP}}$: Start/stop (1/0) Channel.

CHG/$\overline{\text{NOCHG}}$: Change/Do not change (1/0) ST/$\overline{\text{STOP}}$ bit. If this bit is set when writing to the control word, the ST/$\overline{\text{STOP}}$ bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/$\overline{\text{STOP}}$ bit will not be altered. This bit is not stored; it will always be a 0 on read.

INT: Enable Interrupts to CPU on byte count termination.

TC: If set, DMA will terminate when the contents of the Transfer Count register reach zero. The ST/$\overline{\text{STOP}}$ bit will also be reset at this point if TC is set. If this bit is cleared, the DMA unit will decrement the transfer count register for each DMA cycle, but the DMA transfer will not stop when the contents of the TC register reach zero.

SYN: (2 bits)
- 00 No synchronization.
  NOTE: The ST bit will be cleared automatically when the contents of the TC register reach zero regardless of the state of the TC bit.
- 01 Source synchronization.
- 10 Destination synchronization.
- 11 Unused.

SOURCE: INC Increment source pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 after each transfer.

DEST: INC Increment destination pointer by 1 after each transfer.

M/$\overline{\text{IO}}$ Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 after each transfer.

P Channel priority—relative to other channel.
- 0 low priority.
- 1 high priority.

Channels will alternate cycles if both set at same priority level.

allow the maximum number of internal wait states in conjunction with external Ready consideration (i.e., UMCS resets to FFFBH).

- No other chip select or READY control registers have any predefined values after RESET. They will not become active until the CPU accesses their control registers. Both the PACS and MPCS registers must be accessed before the $\overline{PCS}$ lines will become active.

DMA CHANNELS

The 80188 DMA controller provides two independent high-speed DMA channels. Data transfers can occur between memory and I/O spaces (e.g., Memory to I/O) or within the same space (e.g., Memory to Memory or I/O to I/O). Each DMA channel maintains both a 20-bit source and destination pointer which can be optionally incremented or decremented after each data transfer. Each data transfer consumes 2 bus cycles (a minimum of 8 clocks), one cycle to fetch data and the other to store data. This provides a maximum data transfer rate of one MByte/sec.

DMA Operation

Each channel has six registers in the control block which define each channel's specific operation. The control registers consist of a 20-bit Source pointer (2 words), a 20-bit Destination pointer (2 words), a 16-bit Transfer Counter, and a 16-bit Control Word. The format of the DMA Control Blocks is shown in Table 13. The Transfer Count Register (TC) specifies the number of DMA transfers to be performed. Up to 64K byte transfers can be performed with automatic termination. The Control Word defines the channel's operation (see Figure 18). All registers may be modified or altered during any DMA activity. Any changes made to these registers will be reflected immediately in DMA operation.

Table 13. DMA Control Block Format

Register Name	Register Address	
	Ch. 0	Ch. 1
Control Word	CAH	DAH
Transfer Count	C8H	D8H
Destination Pointer (upper 4 bits)	C6H	D6H
Destination Pointer	C4H	D4H
Source Pointer (upper 4 bits)	C2H	D2H
Source Pointer	C0H	D0H

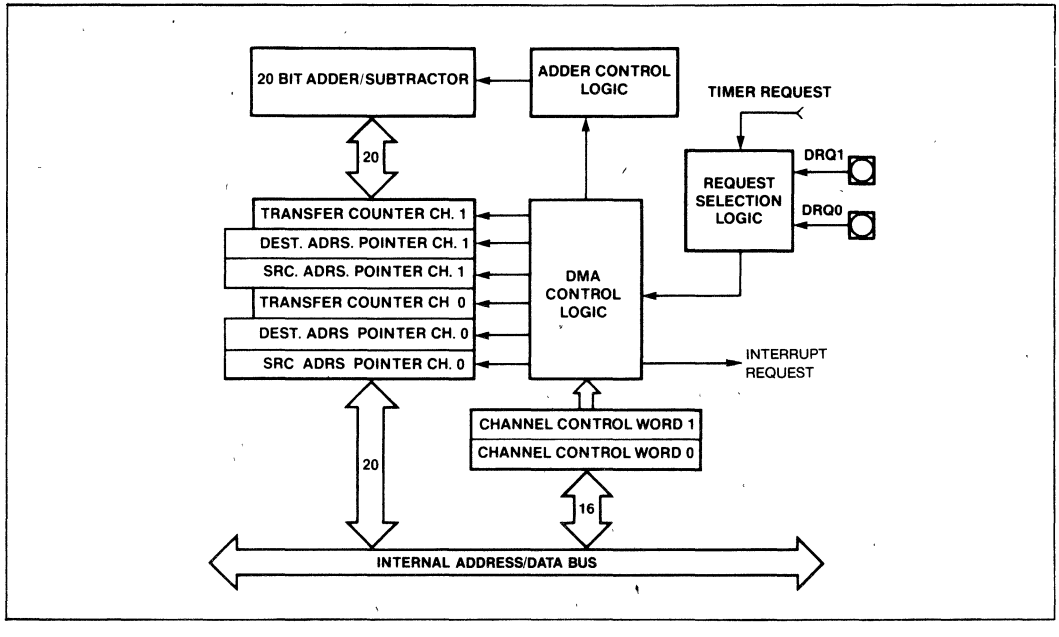


Figure 17. DMA Unit Block Diagram

TDRQ	0: Disable DMA requests from timer 2. 1: Enable DMA requests from timer 2.
Bit 3	Bit 3 is not used.

If both INC and DEC are specified for the same pointer, the pointer will remain constant after each cycle.

DMA Destination and Source Pointer Registers

Each DMA channel maintains a 20-bit source and a 20-bit destination pointer. Each of these pointers takes up two full 16-bit registers in the peripheral control block. The lower four bits of the upper register contain the upper four bits of the 20-bit physical address (see Figure 18a). These pointers may be individually incremented or decremented after each transfer. Each pointer may point into either memory or I/O space. Since the DMA channels can perform transfers to or from odd addresses, there is no restriction on values for the pointer registers.

DMA Transfer Count Register

Each DMA channel maintains a 16-bit transfer count register (TC). This register is decremented after every DMA cycle, regardless of the state of the TC bit int he DMA Control Register. If the TC bit in the DMA control word is set or unsynchronized transfers are programmed, DMA activity will terminate when the transfer count register reaches zero.

DMA Requests

Data transfers may be either source or destination synchronized, that is either the source of the data or the destination of the data may request the data transfer. In addition, DMA transfers may be unsynchronized; that is, the transfer will take place continually until the correct number of transfers has occurred. When source or unsynchronized transfers are performed, the DMA channel may begin another transfer immediately after the end of a previous DMA transfer. This allows a complete transfer to take place every 2 bus cycles or eight clock cycles (assuming no wait states). No prefetching occurs when destination synchronization is performed, however. Data will not be fetched from the source address until the destination device signals that it is ready to receive it. When destination synchronized transfers are requested, the DMA controller will relinquish control of the bus after every transfer. If no other bus activity is initiated, another DMA cycle will begin after two processor clocks. This is done to allow the destination device time to remove its request if another transfer is not desired. Since the DMA controller will relinquish the bus, the CPU can initiate a bus cycle. As a result, a complete bus cycle will often be inserted between destination synchronized transfers. These lead to the maximum DMA transfer rates shown in Table 14.

Table 14. Maximum DMA Transfer Rates

Type of Synchronization Selected	CPU Running	CPU Halted
Unsynchronized	1MBytes/sec	1MBytes/sec
Source Synch	1MBytes/sec	1MBytes/sec
Destination Synch	.65MBytes/sec	.75MBytes/sec

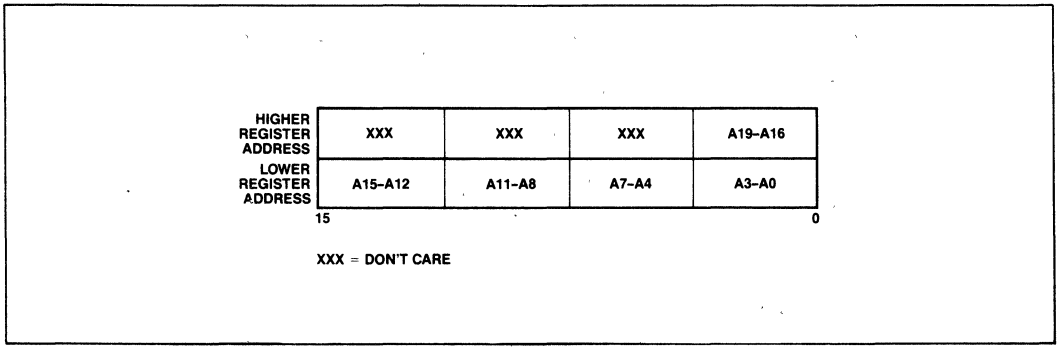


Figure 18a. DMA Memory Pointer Register Format

Table 8. LMCS Programming Values

Upper Address	Memory Block Size	LMCS Value (Assuming R0=R1=R2=0)
003FFH	1K	0038H
007FFH	2K	0078H
00FFFH	4K	00F8H
01FFFH	8K	01F8H
03FFFH	16K	03F8H
07FFFH	32K	07F8H
0FFFFH	64K	0FF8H
1FFFFH	128K	1FF8H
3FFFFH	256K	3FF8H

The upper limit of this memory block is defined in the LMCS register (see Figure 12). This register is at offset A2H in the internal control block. The legal values for bits 6–15 and the resulting upper address and memory block sizes are given in Table 8. Any combination of bits 6–15 not shown in Table 8 will result in undefined operation. After reset, the LMCS register value is undefined. However, the $\overline{LCS}$ chip-select line will not become active until the LMCS register is accessed.

Any internally generated 20-bit address whose upper 16 bits are less than or equal to LMCS (with bits 0–5 "1") will cause $\overline{LCS}$ to be active. LMCS register bits R2–R0 are used to specify the READY mode for the area of memory defined by this chip-select register.

Mid-Range Memory $\overline{CS}$

The iAPX 188 provides four $\overline{MCS}$ lines which are active within a user-locatable memory block. This block can be located anywhere within the iAPX 188 1M byte memory address space exclusive of the areas defined by $\overline{UCS}$ and $\overline{LCS}$. Both the base address and size of this memory block are programmable.

The size of the memory block defined by the mid-range select lines, as shown in Table 9, is determined by bits 8–14 of the MPCS register (see Figure 13). This register is at location A8H in the internal control block. One and only one of bits 8–14 must be set at a time. Unpredictable operation of the $\overline{MCS}$ lines will otherwise occur. Each of the four chip-select lines is active for one of the four equal contiguous divisions of the mid-range block. Thus, if the total block size is 32K, each chip select is active for 8K of memory with $\overline{MCS0}$ being active for the first range and $\overline{MCS3}$ being active for the last range.

The EX and MS in MPCS relate to peripheral functionality as descibed a later section.

Table 9. MPCS Programming Values

Total Block Size	Individual Select Size	MPCS Bits 14-8
8K	2K	0000001B
16K	4K	0000010B
32K	8K	0000100B
64K	16K	0001000B
128K	32K	0010000B
256K	64K	0100000B
512K	128K	1000000B

The base address of the mid-range memory block is defined by bits 15–9 of the MMCS register (see Figure 14). This register is at offset A6H in the internal control block. These bits correspond to bits A19–A13 of the 20-bit memory address. Bits A12–A0 of the base address are always 0. The base address may be set at any integer multiple of the size of the total memory block selected. For example, if the mid-range block size is 32K (or the size of the block for which each $\overline{MCS}$ line is active is 8K), the block could be located at 10000H or 18000H, but not at 14000H, since the first few integer multiples of a 32K memory block are 0H, 8000H, 10000H, 18000H, etc. After reset, the contents of both of these registers is undefined. However, none of the $\overline{MCS}$ lines will be active until both the MMCS and MPCS registers are accessed.

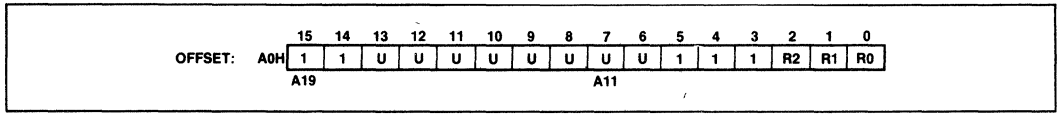


Figure 11. UMCS Register

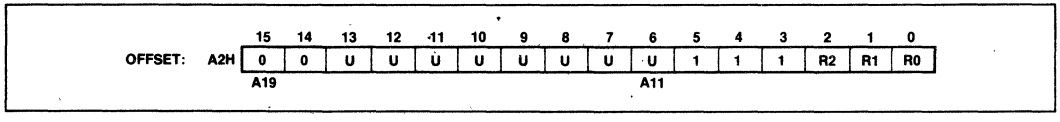


Figure 12. LMCS Register

DMA Acknowledge

No explicit DMA acknowledge pulse is provided. Since both source and destination pointers are maintained, a read from a requesting source, or a write to a requesting destination, should be used as the DMA acknowledge signal. Since the chip-select lines can be programmed to be active for a given block of memory or I/O space, and the DMA pointers can be programmed to point to the same given block, a chip-select line could be used to indicate a DMA acknowledge.

DMA Priority

The DMA channels may be programmed such that one channel is always given priority over the other, or they may be programmed such as to alternate cycles when both have DMA requests pending. DMA cycles always have priority over internal CPU cycles except between locked memory accesses or word accesses the odd memory locations; however, an external bus hold takes priority over an internal DMA cycle. Because an interrupt request cannot suspend a DMA operation and the CPU cannot access memory during a DMA cycle, interrupt latency time will suffer during sequences of continuous DMA cycles. An NMI request, however, will cause all internal DMA activity to halt. This allows the CPU to quickly respond to the NMI request.

DMA Programming

DMA cycles will occur whenever the ST/STOP bit of the Control Register is set. If synchronized transfers are programmed, a DRQ must also have been generated. Therefore, the source and destination transfer pointers, and the transfer count register (if used) must be programmed before this bit is set.

Each DMA register may be modified while the channel is operating. If the CHG/NOCHG bit is cleared when the control register is written, the ST/STOP bit of the control register will not be modified by the write. If multiple channel registers are modified, it is recommended that a LOCKED string transfer be used to prevent a DMA transfer from occurring between updates to the channel registers.

DMA Channels and Reset

Upon RESET, the DMA channels will perform the following actions:

- The Start/Stop bit for each channel will be reset to STOP.
- Any transfer in progress is aborted.

TIMERS

The 80188 provides three internal 16-bit programmable timers (see Figure 19). Two of these are highly flexible and are connected to four external pins (2 per timer). They can be used to count external events, time external events, generate nonrepetitive waveforms, etc. The third timer is not connected to any external pins, and is useful for real-time coding and time delay applications. In addition, this third timer can be used as a prescaler to the other two, or as a DMA request source.

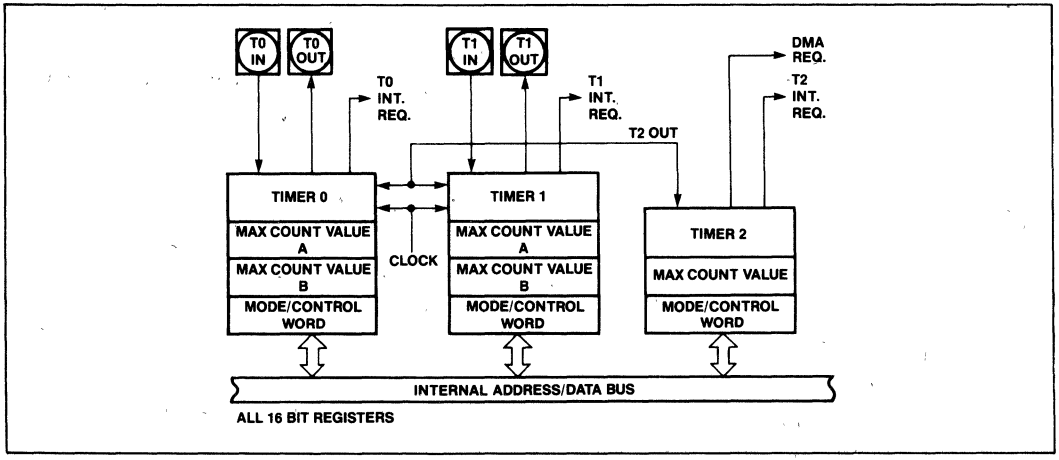


Figure 19. Timer Block Diagram

AFN-01483A

Timer Operation

The timers are controlled by 11 16-bit registers in the internal peripheral control block. The configuration of these registers is shown in Table 15. The count register contains the current value of the timer. It can be read or written at any time independent of whether the timer is running or not. The value of this register will be incremented for each timer event. Each of the timers is equipped with a MAX COUNT register, which defines the maximum count the timer will reach. After reaching the MAX COUNT register value, the timer count value will reset to zero during that same clock, i.e., the maximum count value is never stored in the count register itself. Timers 0 and 1 are, in addition, equipped with a second MAX COUNT register, which enables the timers to alternate their count between two different MAX COUNT values programmed by the user. If a single MAX COUNT register is used, the timer output pin will switch LOW for a single clock, 2 clocks after the maximum count value has been reached. In the dual MAX COUNT register mode, the output pin will indicate which MAX COUNT register is currently in use, thus allowing nearly complete freedom in selecting waveform duty cycles. For the timers with two MAX COUNT registers, the RIU bit in the control register determines which is used for the comparison.

Each timer gets serviced every fourth CPU-clock cycle, and thus can operate at speeds up to one-quarter the internal clock frequency (one-eighth the crystal rate). External clocking of the timers may be done at up to a rate of one-quarter of the internal CPU-clock rate (2 MHz for an 8 MHz CPU clock). Due to internal synchronization and pipelining of the timer circuitry, a timer output may take up to 6 clocks to respond to any individual clock or gate input.

Since the count registers and the maximum count registers are all 16 bits wide, 16 bits of resolution are provided. Any Read or Write access to the timers will add one wait state to the minimum four-clock bus cycle, however. This is needed to synchronize and coordinate the internal data flows between the internal timers and the internal bus.

The timers have several programmable options.

- All three timers can be set to halt or continue on a terminal count.
- Timers 0 and 1 can select between internal and external clocks, alternate between MAX COUNT registers and be set to retrigger on external events.
- The timers may be programmed to cause an interrupt on terminal count.

These options are selectable via the timer mode/control word.

Timer Mode/Control Register

The mode/control register (see Figure 20) allows the user to program the specific mode of operation or check the current programmed status for any of the three integrated timers.

Table 15. Timer Control Block Format

Register Name	Register Offset		
	Tmr. 0	Tmr. 1	Tmr. 2
Mode/Control Word	56H	5EH	66H
Max Count B	54H	5CH	not present
Max Count A	52H	5AH	62H
Count Register	50H	58H	60H

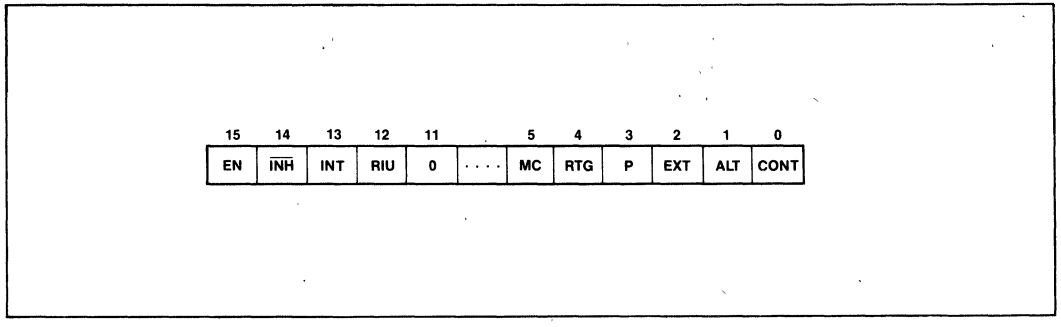


Figure 20. Timer Mode/Control Register

ALT:
The ALT bit determines which of two MAX COUNT registers is used for count comparison. If ALT = 0, register A for that timer is always used, while if ALT = 1, the comparison will alternate between register A and register B when each maximum count is reached. This alternation allows the user to change one MAX COUNT register while the other is being used, and thus provides a method of generating non-repetitive waveforms. Square waves and pulse outputs of any duty cycle are a subset of available signals obtained by not changing the final count registers. The ALT bit also determines the function of the timer output pin. If ALT is zero, the output pin will go LOW for one clock, the clock after the maximum count is reached. If ALT is one, the output pin will reflect the current MAX COUNT register being used (0/1 for B/A).

CONT:
Setting the CONT bit causes the associated timer to run continuously, while resetting it causes the timer to halt upon maximum count. If CONT = 0 and ALT =1, the timer will count to the MAX COUNT register A value, reset, count to the register B value, reset, and halt.

EXT:
The external bit selects between internal and external clocking for the timer. The external signal may be asynchronous with respect to the 80188 clock. If this bit is set, the timer will count LOW-to-HIGH transitions on the input pin. If cleared, it will count an internal clock while using the input pin for control. In this mode, the function of the external pin is defined by the RTG bit. The maximum input to output transition latency time may be as much as 6 clocks. However, clock inputs may be pipelined as closely together as every 4 clocks without losing clock pulses.

P:
The prescaler bit is ignored unless internal clocking has been selected (EXT = 0). If the P bit is a zero, the timer will count at one-fourth the internal CPU clock rate. If the P bit is a one, the output of timer 2 will be used as a clock for the timer. Note that the user must initialize and start timer 2 to obtain the prescaled clock.

RTG:
Retrigger bit is only active for internal clocking (EXT = 0). In this case it determines the control function provided by the input pin.

If RTG = 0, the input level gates the internal clock on and off. If the input pin is HIGH, the timer will count; if the input pin is LOW, the timer will hold its value. As indicated previously, the input signal may be asynchronous with respect to the 80188 clock.

When RTG = 1, the input pin detects LOW-to-HIGH transitions. The first such transition starts the timer running, clearing the timer value to zero on the first clock, and then incrementing thereafter. Further transitions on the input pin will again reset the timer to zero, from which it will start counting up again. If CONT = 0, when the timer has reached maximum count, the EN bit will be cleared, inhibiting further timer activity.

EN:
The enable bit provides programmer control over the timer's RUN/HALT status. When set, the timer is enabled to increment subject to the input pin constraints in the internal clock mode (discussed previously). When cleared, the timer will be inhibited from counting. All input pin transitions during the time EN is zero will be ignored. If CONT is zero, the EN bit is automatically cleared upon maximum count.

INH:
The inhibit bit allows for selective updating of the enable (EN) bit. If INH is a one during the write to the mode/control word, then the state of the EN bit will be modified by the write. If INH is a zero during the write, the EN bit will be unaffected by the operation. This bit is not stored; it will always be a 0 on a read.

INT:
When set, the INT bit enables interrupts from the timer, which will be generated on every terminal count. If the timer is configured in dual MAX COUNT register mode, an interrupt will be generated each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. If this enable bit is cleared after the interrupt request has been generated, but before a pending interrupt is serviced, the interrupt request will still be in force. (The request is latched in the Interrupt Controller.)

MC:
The Maximum Count bit is set whenever the timer reaches its final maximum count value. If the timer is configured in dual MAX COUNT register mode, this bit will be set each time the value in MAX COUNT register A is reached, and each time the value in MAX COUNT register B is reached. This bit is set regardless of the timer's interrupt-enable bit. The MC bit gives the user the ability to monitor timer status through software instead of through interrupts. Programmer intervention is required to clear this bit.

RIU:
The Register In Use bit indicates which MAX COUNT register is currently being used for comparison to the timer count value. A zero value indicates register A. The RIU bit cannot be written, i.e., its value is not affected when the control register is written. It is always cleared when the ALT bit is zero.

Not all mode bits are provided for timer 2. Certain bits are hardwired as indicated below:

ALT = 0, EXT = 0, P = 0, RTG = 0, RIU = 0

Count Registers

Each of the three timers has a 16-bit count register. The current contents of this register may be read or written by the processor at any time. If the register is written into while the timer is counting, the new value will take effect in the current count cycle.

Max Count Registers

Timers 0 and 1 have two MAX COUNT registers, while timer 2 has a single MAX COUNT register. These contain the number of events the timer will count. In timers 0 and 1, the MAX COUNT register used can alternate between the two max count values whenever the current maximum count is reached. The condition which causes a timer to reset is equivalent between the current count value and the max count being used. This means that if the count is changed to be above the max count value, or if the max count value is changed to be below the current value, the timer will not reset to zero, but rather will count to its maximum value, "wrap around" to zero, then count until the max count is reached.

Timers and Reset

Upon RESET, the Timers will perform the following actions:

- All EN (Enable) bits are reset preventing timer counting.
- All SEL (Select) bits are reset to zero. This selects MAX COUNT register A, resulting in the Timer Out pins going HIGH upon RESET.

INTERRUPT CONTROLLER

The 80188 can receive interrupts from a number of sources, both internal and external. The internal interrupt controller serves to merge these requests on a priority basis, for individual service by the CPU. Internal interrupt sources (Timers and DMA channels) can be disabled by their own control registers or by mask bits within the interrupt controller. The 80188 interrupt controller has its own control registers that set the mode of operation for the controller.

The interrupt controller will resolve priority among requests that are pending simultaneously. Nesting is provided so interrupt service routines for lower priority interrupts may themselves be interrupted by higher priority interrupts. A block diagram of the interrupt controller is shown in Figure 21.

The interrupt controller has a special iRMX 86 compatibility mode that allows the use of the 80188 within the iRMX 86 operating system interrupt structure. The controller is set in this mode by setting bit 14 in the peripheral control block relocation register (see iRMX 86 Compatibility Mode section). In this mode, the internal 80188 interrupt controller functions as a "slave" controller to an external "master" controller. Special initialization software must be included to properly set up the 80188 interrupt controller in iRMX 86 mode.

NON-iRMX MODE OPERATION

Interrupt Controller External Interface

For external interrupt sources, five dedicated pins are provided. One of these pins is dedicated to NMI, non-maskable interrupt. This is typically used for power-fail interrupts, etc. The other four pins may function either as four interrupt input lines with internally generated interrupt vectors, as an interrupt line and an interrupt acknowledge line (called the "cascade mode") along with two other input lines with internally generated interrupt vectors, or as two interrupt input lines and two dedicated interrupt acknowledge ouput lines. When the interrupt lines are configured in cascade mode, the 80188 interrupt controller will not generate internal interrupt vectors.

External sources in the cascade mode use externally generated interrupt vectors. When an interrupt is acknowledged, two $\overline{\text{INTA}}$ cycles are initiated and the vector is read into the 80188 on the second cycle. The capability to interface to external 8259A programmable interrupt controllers is thus provided when the inputs are configured in cascade mode.

Interrupt Controller Modes of Operation

The basic modes of operation of the interrupt controller in non-iRMX mode are similar to the 8259A. The interrupt controller responds identically to internal interrupts in all three modes: the difference is only in the interpretation of function of the four external interrupt pins. The interrupt controller is set into one of these three modes by programming the correct bits in the INT0 and INT1 control registers. The modes of interrupt controller operation are as follows:

Fully Nested Mode

When in the fully nested mode four pins are used as direct interrupt requests. The vectors for these four inputs are generated internally. An in-service bit is provided for every interrupt source. If a lower-priority device requests an interrupt while the in-service bit (IS) is set, no interrupt will be generated by the interrupt controller. In addition, if another interrupt request occurs from the same interrupt source while the inservice bit is set, no interrupt will be generated by the interrupt controller. This allows interrupt service routines to operate with interrupts enabled without being themselves interrupted by lower-priority interrupts. Since interrupts are enabled, higher-priority interrupts will be serviced.

When a service routine is completed, the proper IS bit must be reset by writing the proper pattern to the EOI register. This is required to allow subsequent interrupts from this interrupt source and to allow servicing of lower-priority interrupts. An EOI command is issued at the end of the service routine just before the issuance of the return from interrupt instruction. If the fully nested structure has been upheld, the next highest-priority source with its IS bit set is then serviced.

Cascade Mode

The 80188 has four interrupt pins and two of them have dual functions. In the fully nested mode the four pins are used as direct interrupt inputs and the corresponding vectors are generated internally. In the cascade mode, the four pins are configured into interrupt input-dedicated acknowledge signal pairs. The interconnection is shown in Figure 22. INT0 is an interrupt input interfaced to an 8259A, while INT2/$\overline{\text{INTA0}}$ serves as the dedicated interrupt acknowledge signal to that peripheral. The same is true for INT1 and INT3/$\overline{\text{INTA1}}$. Each pair can selectively be placed in the cascade or non-cascade mode by programming the proper value into INT0 and INT1 control registers. The use of the dedicated acknowledge signals eliminates the need for the use of external logic to generate $\overline{\text{INTA}}$ and device select signals.

The primary cascade mode allows the capability to serve up to 128 external interrupt sources through the use of external master and slave 8259As. Three levels of priority are created, requiring priority resolution in the 80188 interrupt controller, the master 8259As, and the slave 8259As. If an external interrupt is serviced, one IS bit is set at each of these levels. When the interrupt service routine is completed, up to three end-of-interrupt commands must be issued by the programmer.

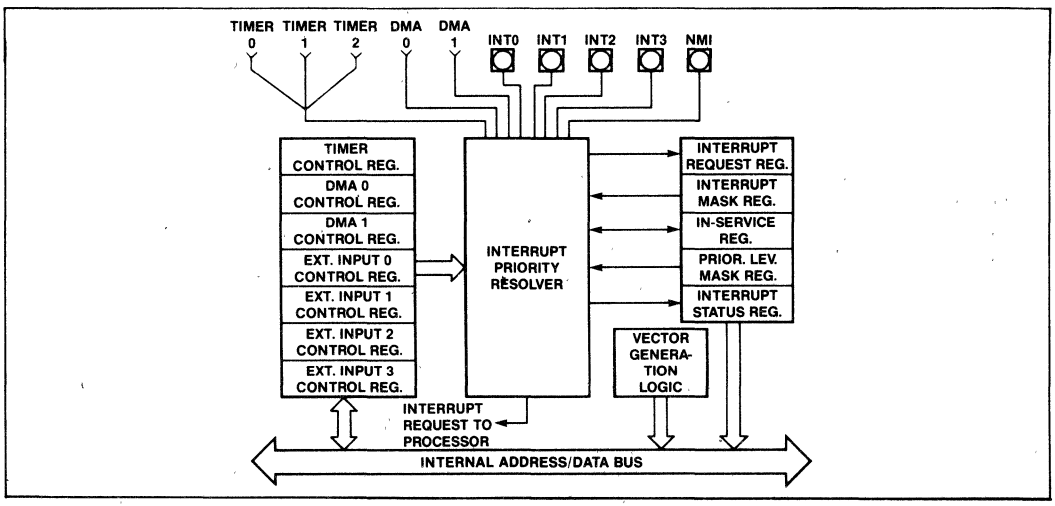


Figure 21. Interrupt Controller Block Diagram

Special Fully Nested Mode
This mode is entered by setting the SFNM bit in INT0 or INT1 control register. It enables complete nestability with external 8259A masters. Normally, an interrupt request from an interrupt source will not be recognized unless the in-service bit for that source is reset. If more than one interrupt source is connected to an external interrupt controller, all of the interrupts will be funneled through the same 80188 interrupt request pin. As a result, if the external interrupt controller receives a higher-priority interrupt, its interrupt will not be recognized by the 80188 controller until the 80188 in-service bit is reset. In special fully nested mode, the 80188 interrupt controller will allow interrupts from an external pin regardless of the state of the in-service bit for an interrupt source in order to allow multiple interrupts from a single pin. An in-service bit will continue to be set, however, to inhibit interrupts from other lower-priority 80188 interrupt sources.

Special procedures should be followed when resetting IS bits at the end of interrupt service routines. Software polling of the external master's IS register is required to determine if there is more than one bit set. If so, the IS bit in the 80188 remains active and the next interrupt service routine is entered.

Operation in a Polled Environment

The controller may be used in a polled mode if interrupts are undesirable. When polling, the processor disables interrupts and then polls the interrupt controller whenever it is convenient. Polling the interrupt controller is accomplished by reading the Poll Word (Figure 31). Bit 15 in the poll word indicates to the processor that an interrupt of high enought priority is requesting service. Bits 0-4 indicate to the processor the type vector of the highest-priority source requesting service. Reading the Poll Word causes the In-Service bit of the highest-priority source to be set.

It is desirable to be able to read the Poll Word information without guaranteeing service of any pending interrupt, i.e., not set the indicated in-service bit. The 80188 provides a Poll Status Word in addition to the conventional Poll Word to allow this to be done. Poll Word information is duplicated in the Poll Status Word, but reading the Poll Status Word does not set the associated in-service bit. These words are located in two adjacent memory locations in the register file.

Non-iRMX Mode Features

Programmable Priority
The user can program the interrupt sources into any of eight different priority levels. The programming is done by placing a 3-bit priority level (0–7) in the control register of each interrupt source. (A source with a priority level of 4 has higher priority over all priority levels from 5 to 7. Priority registers containing values lower than 4 have greater priority.) All interrupt sources have preprogrammed default priority levels (see Table 4).

If two requests with the same programmed priority level are pending at once, the priority ordering scheme shown in Table 4 is used. If the serviced interrupt routine reenables interrupts, it allows other requests to be serviced.

End-of-Interrupt Command
The end-of-interrupt (EOI) command is used by the programmer to reset the In-Service (IS) bit when an interrupt service routine is completed. The EOI command is issued by writing the proper pattern to the EOI register. There are two types of EOI commands, specific and nonspecific. The nonspecific command does not specify which IS bit is reset. When issued, the interrupt controller automatically resets the IS bit of the highest priority source with an active service routine. A specific EOI command requires that the programmer send the interrupt vector type to the interrupt controller indicating which source's IS bit is to be reset. This command is used when the fully nested structure has been disturbed or the highest priority IS bit that was set does not belong to the service routine in progress.

Trigger Mode
The four external interrupt pins can be programmed in either edge- or level-trigger mode. The control register for each external source has a level-trigger mode (LTM) bit. All interrupt inputs are active HIGH. In the edge sense mode or the level-trigger mode, the interrupt request must remain active (HIGH) until the interrupt request is acknowledged by the 80188 CPU. In the edge-sense mode, if the level remains high after the interrupt is acknowledged, the input is disabled and no further requests will be generated. The input level must go LOW for at least one clock cycle to reenable the input. In the level-trigger mode, no such provision is made: holding the interrupt input HIGH will cause continuous interrupt requests.

Interrupt Vectoring
The 80188 Interrupt Controller will generate interrupt vectors for the integrated DMA channels and the integrated Timers. In addition, the Interrupt Controller will generate interrupt vectors for the external interrupt lines if they are not configured in Cascade or Special Fully Nested Mode. The interrupt vectors generated are fixed and cannot be changed (see Table 4).

Interrupt Controller Registers

The Interrupt Controller register model is shown in Figure 23. It contains 15 registers. All registers can both be read or written unless specified otherwise.

In-Service Register
This register can be read from or written into. The format is shown in Figure 24. It contains the In-Service bit for each of the interrupt sources. The In-Service bit is set to indicate that a source's service routine is in progress. When an In-Service bit is set, the interrupt controller will not generate interrupts to the CPU when it receives interrupt requests from devices with a lower programmed priority level. The TMR bit is the In-Service bit for all three timers; the D0 and D1 bits are the In-Service bits for the two DMA channels; the I0–I3 are the In-Service bits for the external interrupt pins. The IS bit is set when the processor acknowledges an interrupt request either by an interrupt acknowledge or by reading the poll register. The IS bit is reset at the end of the interrupt service routine by an end-of-interrupt command issued by the CPU.

Interrupt Request Register
The internal interrupt sources have interrupt request bits inside the interrupt controller. The format of this register is shown in Figure 24. A read from this register yields the status of these bits. The TMR bit is the logical OR of all timer interrupt requests. D0 and D1 are the interrupt request bits for the DMA channels.

The state of the external interrupt input pins is also indicated. The state of the external interrupt pins is not a stored condition inside the interrupt controller, therefore the external interrupt bits cannot be written. The external interrupt request bits show exactly when an interrupt request is given to the interrupt controller, so if edge-triggered mode is selected, the bit in the register will be HIGH only after an inactive-to-active transition. For internal interrupt sources, the register bits are set when a request arrives and are reset when the processor acknowledges the requests.

Mask Register
This is a 16-bit register that contains a mask bit for each interrupt source. The format for this register is shown in Figure 24. A one in a bit position corresponding to a particular source serves to mask the source from generating interrupts. These mask bits are the exact same bits which are used in the individual control registers; programming a mask bit using the mask register will also change this bit in the individual control registers, and vice versa.

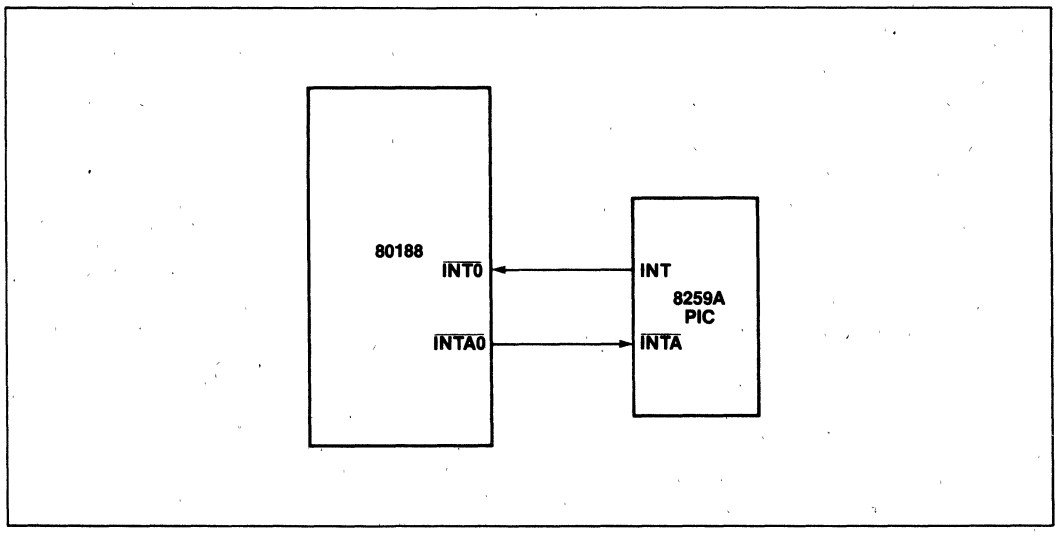


Figure 22. Cascade Mode Interrupt Connection

	OFFSET
INT3 CONTROL REGISTER	3EH
INT2 CONTROL REGISTER	3CH
INT1 CONTROL REGISTER	3AH
INT0 CONTROL REGISTER	38H
DMA 1 CONTROL REGISTER	36H
DMA 0 CONTROL REGISTER	34H
TIMER CONTROL REGISTER	32H
INTERRUPT STATUS REGISTER	30H
INTERRUPT REQUEST REGISTER	2EH
IN-SERVICE REGISTER	2CH
PRIORITY MASK REGISTER	2AH
MASK REGISTER	28H
POLL STATUS REGISTER	26H
POLL REGISTER	24H
EOI REGISTER	22H

Figure 23. Interrupt Controller Registers (Non-iRMX 86 Mode)

Priority Mask Register

This register is used to mask all interrupts below particular interrupt priority levels. The format of this register is shown in Figure 25. The code in the lower three bits of this register inhibits interrupts of priority lower (a higher priority number) than the code specified. For example, 100 written into this register masks interrupts of level five (101), six (110), and seven (111). The register is reset to seven (111) upon RESET so all interrupts are unmasked.

Interrupt Status Register

This register contains general interrupt controller status information. The format of this register is shown in Figure 26. The bits in the status register have the following functions:

DHLT: DMA Halt Transfer; setting this bit halts all DMA transfers. It is automatically set whenever a non-maskable interrupt occurs, and it is reset when an IRET instruction is executed. The purpose of this bit is to allow prompt service of all non-maskable interrupts. This bit may also be set by the CPU.

IRTx: These three bits represent the individual timer interrupt request bits. These bits are used to differentiate the timer interrupts, since the timer IR bit in the interrupt request register is the "OR" function of all timer interrupt requests. Note that setting any one of these three bits initiates an interrupt request to the interrupt controller.

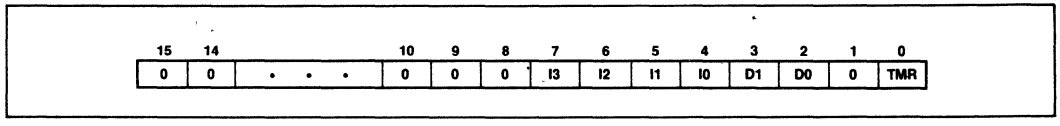


Figure 24. In-Service, Interrupt Request, and Mask Register Formats

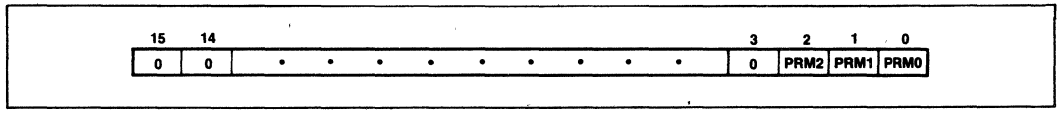


Figure 25. Priority Mask Register Format

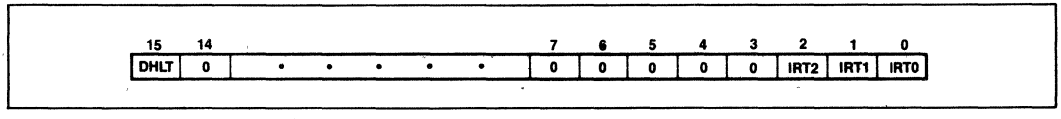


Figure 26. Interrupt Status Register Format

Timer, DMA 0, 1; Control Registers
These registers are the control words for all the internal interrupt sources. The format for these registers is shown in Figure 27. The three bit positions PR0, PR1, and PR2 represent the programmable priority level of the interrupt source. The MSK bit inhibits interrupt requests from the interrupt source. The MSK bits in the individual control registers are the exact same bits as are in the Mask Register; modifying them in the individual control registers will also modify them in the Mask Register, and vice versa.

INT0–INT3 Control Registers
These registers are the control words for the four external input pins. Figure 28 shows the format of the INT0 and INT1 Control registers; Figure 29 shows the format of the INT2 and INT3 Control registers. In cascade mode or special fully nested mode, the control words for INT2 and INT3 are not used.

The bits in the various control registers are encoded as follows:

PRO-2: Priority programming information. Highest priority = 000, lowest priority = 111.

LTM: Level-trigger mode bit. 1 = level-triggered; 0 = edge-triggered. Interrupt Input levels are active high. In level-triggered mode, an interrupt is generated whenever the external line is high. In edge-triggered mode, an interrupt will be generated only when this level is preceded by an inactive-to-active transition on the line. In both cases, the level must remain active until the interrupt is acknowledged.

MSK: Mask bit, 1 = mask; 0 = nonmask.

C: Cascade mode bit, 1 = cascade; 0 = direct

SFNM: Special fully nested mode bit, 1 = SFNM

EOI Register
The end of the interrupt register is a command register which can only be written into. The format of this register is shown in Figure 30. It initiates an EOI command when written to by the 80188 CPU.

The bits in the EOI register are encoded as follows:

S_x: Encoded information that specifies an interrupt source vector type as shown in Table 4. For example, to reset the In-Service bit for DMA channel 0, these bits should be set to 01010, since the vector type for DMA channel 0 is 10. Note that to reset the single In-Service bit for any of the three timers, the vector type for timer 0 (8) should be written in this register.

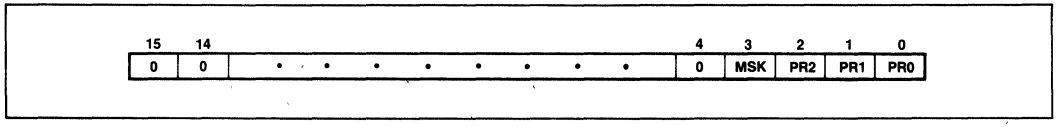


Figure 27. Timer/DMA Control Register Formats

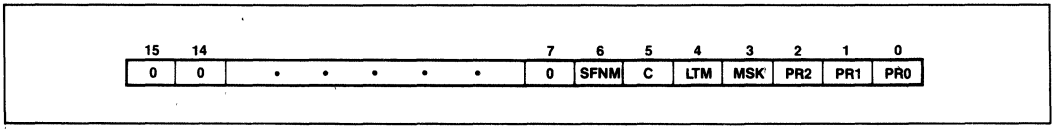


Figure 28. INT0/INT1 Control Register Formats

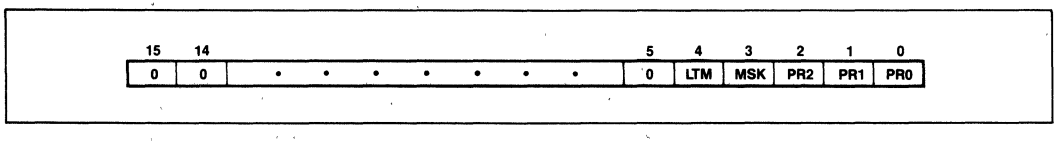


Figure 29. INT2/INT3 Control Register Formats

NSPEC/SPEC: A bit that determines the type of EOI command. Nonspecific = 1, Specific = 0.

Poll and Poll Status Registers

These registers contain polling information. The format of these registers is shown in Figure 31. They can only be read. Reading the Poll register constitutes a software poll. This will set the IS bit of the highest priority pending interrupt. Reading the poll status register will not set the IS bit of the highest priority pending interrupt; only the status of pending interrupts will be provided.

Encoding of the Poll and Poll Status register bits are as follows:

S_x: Encoded information that indicates the vector type of the highest priority interrupting source. Valid only when INTREQ = 1.

INTREQ: This bit determines if an interrupt request is present. Interrupt Request = 1; no Interrupt Request = 0.

iRMX 86 COMPATIBILITY MODE

This mode allows iRMX 86-80188 compatibility. The interrupt model of iRMX 86 requires one master and multiple slave 8259As in cascaded fashion. When iRMX mode is used, the internal 80188 interrupt controller will be used as a slave controller to an external master interrupt controller. The internal 80188 resources will be monitored through the internal interrupt controller, while the external controller functions as the system master interrupt controller.

Upon reset, the 80188 interrupt controller will be in the non-iRMX 86 mode of operation. To set the controller in the iRMX 86 mode, bit 14 of the Relocation Register should be set.

Because of pin limitations caused by the need to interface to an external 8259A master, the internal interrupt controller will no longer accept external inputs. There are however, enough 80188 interrupt controller inputs (internally) to dedicate one to each timer. In this mode, each timer interrupt source has its own mask bit, IS bit, and control word.

The iRMX 86 operating system requires peripherals to be assigned fixed priority levels. This is incompatible with the normal operation of the 80188 interrupt controller. Therefore, the initialization software must program the proper priority levels for each source. The required priority levels for the internal interrupt sources in iRMX mode are shown in Table 16.

Table 16. Internal Source Priority Level

Priority Level	Interrupt Source
0	Timer 0
1	(reserved)
2	DMA 0
3	DMA 1
4	Timer 1
5	Timer 2

These level assignments must remain fixed in the iRMX 86 mode of operation.

iRMX 86 Mode External Interface

The configuration of the 80188 with respect to an external 8259A master is shown in Figure 32. The INT0 input is used as the 80188 CPU interrupt input. INT3 functions as an output to send the 80188 slave-interrupt-request to one of the 8 master-PIC-inputs.

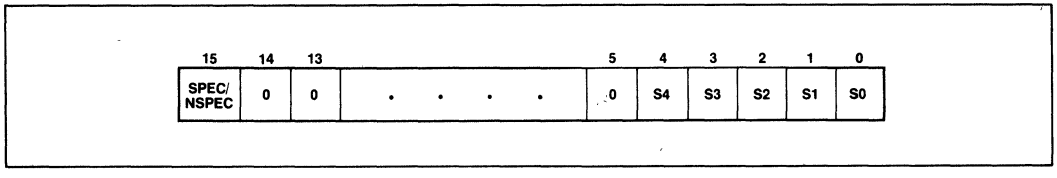


Figure 30. EOI Register Format

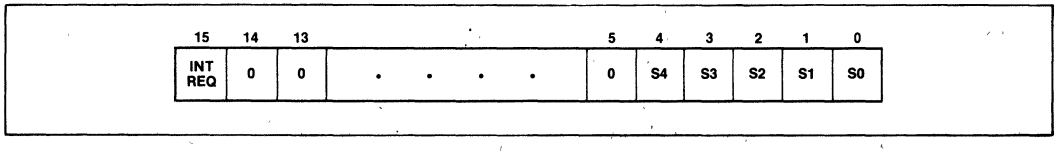


Figure 31. Poll Register Format

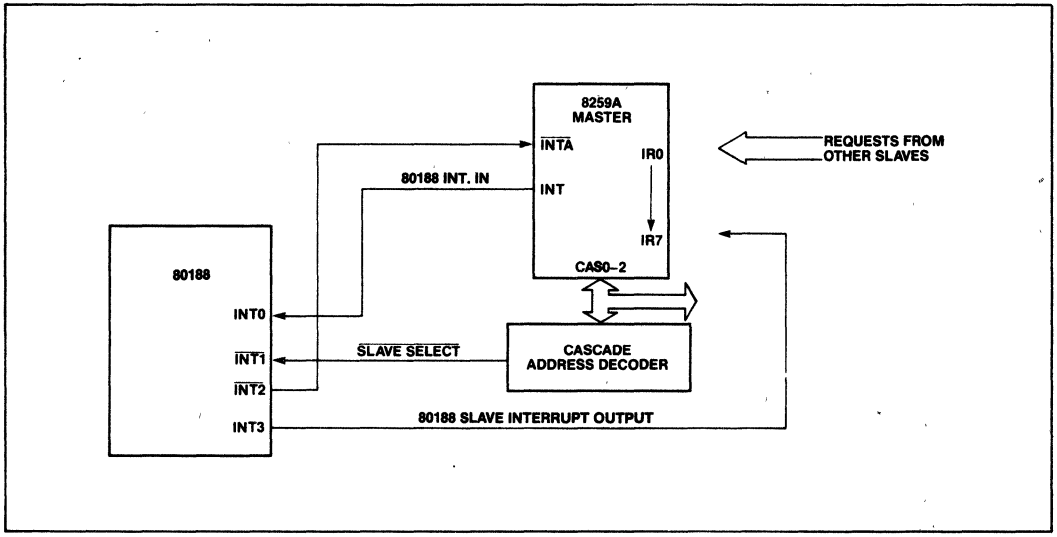


Figure 32. iRMX 86 Interrupt Controller Interconnection

Correct master-slave interface requires decoding of the slave addresses (CAS0–2). Slave 8259As do this internally. Because of pin limitations, the 80188 slave address will have to be decoded externally. $\overline{INT1}$ is used as a slave-select input. Note that the slave vector address is transferred internally, but the READY input must be supplied externally.

$\overline{INT2}$ is used as an acknowledge output, suitable to drive the $\overline{INTA}$ input of an 8259A.

Interrupt Nesting

iRMX 86 mode operation allows nesting of interrupt requests. When an interrupt is acknowledged, the priority logic masks off all priority levels except those with equal or higher priority.

Vector Generation in the iRMX 86 Mode

Vector generation in iRMX mode is exactly like that of an 8259A slave. The interrupt controller generates an 8-bit vector which the CPU multiplies by four and uses as an address into a vector table. The significant five bits of the vector are user-programmable while the lower three bits are generated by the priority logic. These bits represent the encoding of the priority level requesting service. The significant five bits of the vector are programmed by writing to the Interrupt Vector register at offset 20H.

Specific End-of-Interrupt

In iRMX mode the specific EOI command operates to reset an in-service bit of a specific priority. The user supplies a 3-bit priority-level value that points to an in-service bit to be reset. The command is executed by writing the correct value in the Specific EOI register at offset 22H.

Interrupt Controller Registers in the iRMX 86 Mode

All control and command registers are located inside the internal peripheral control block. Figure 33 shows the offsets of these registers.

End-of-Interrupt Register

The end-of-interrupt register is a command register which can only be written. The format of this register is shown in Figure 34. It initiates an EOI command when written by the 80188 CPU.

The bits in the EOI register are encoded as follows:

L_X: Encoded value indicating the priority of the IS bit to be reset.

In-Service Register

This register can be read from or written into. It contains the in-service bit for each of the internal

interrupt sources. The format for this register is shown in Figure 35. Bit positions 2 and 3 correspond to the DMA channels; positions 0, 4, and 5 correspond to the integral timers. The source's IS bit is set when the processor acknowledges its interrupt request.

Interrupt Request Register
This register indicates which internal peripherals have interrupt requests pending. The format of this register is shown in Figure 35. The interrupt request bits are set when a request arrives from an internal source, and are reset when the processor acknowledges the request.

Mask Register
This register contains a mask bit for each interrupt source. The format for this register is shown in Figure 35. If the bit in this register corresponding to a particular interrupt source is set, any interrupts from that source will be masked. These mask bits are exactly the same bits which are used in the individual control registers, i.e., changing the state of a mask bit in this register will also change the state of the mask bit in the individual interrupt control register corresponding to the bit.

Control Registers
These registers are the control words for all the internal interrupt sources. The format of these registers is shown in Figure 36. Each of the timers and both of the DMA channels have their own Control Register.

The bits of the Control Registers are encoded as follows:

pr_x: 3-bit encoded field indicating a priority level for the source; note that each source must be programmed at specified levels.

msk: mask bit for the priority level indicated by pr_x bits.

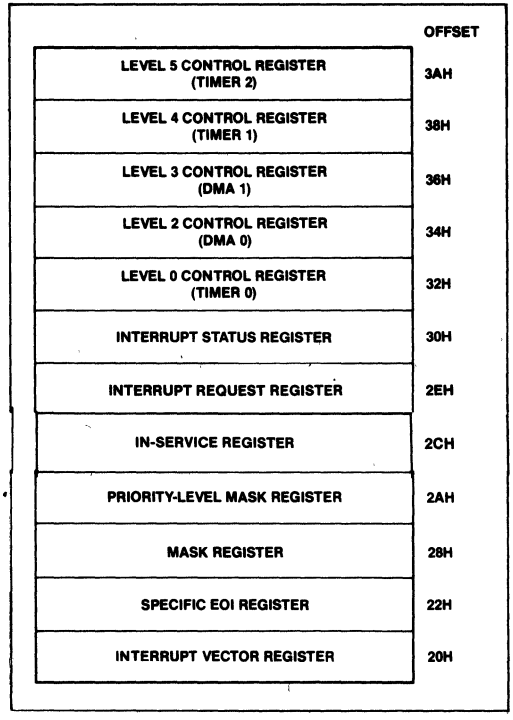


Figure 33. Interrupt Controller Registers (iRMX 86 Mode)

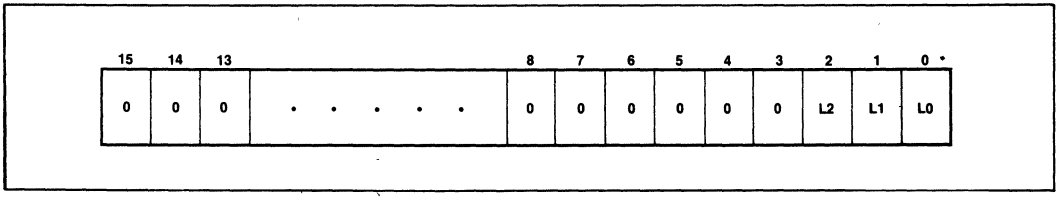


Figure 34. Specific EOI Register Format

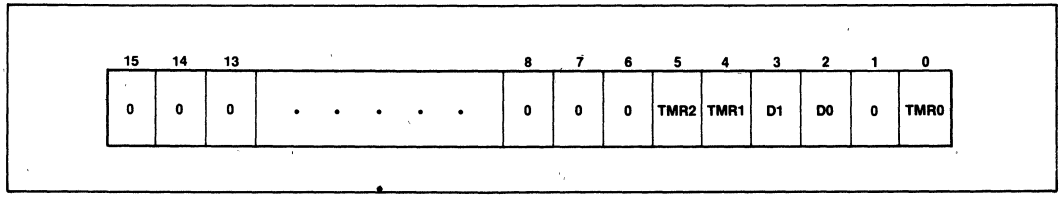


Figure 35. In-Service, Interrupt Request, and Mask Register Format

Interrupt Vector Register
This register provides the upper five bits of the interrupt vector address. The format of this register is shown in Figure 37. The interrupt controller itself provides the lower three bits of the interrupt vector as determined by the priority level of the interrupt request.

The format of the bits in this register is:

t_x: 5-bit field indicating the upper five bits of the vector address.

Priority-Level Mask Register
This register indicates the lowest priority-level interrupt which will be serviced.

The encoding of the bits in this register is:

m_x: 3-bit encoded field indication priority-level value. All levels of lower priority will be masked.

Interrupt Status Register
This register is defined exactly as in non-iRMX mode (see Figure 26).

Interrupt Controller and Reset

Upon RESET, the interrupt controller will perform the following actions:

- All SFNM bits reset to 0, implying Fully Nested Mode.
- All PR bits in the various control registers set to 1. This places all sources at lowest priority (level 111).
- All LTM bits reset to 0, resulting in edge-sense mode.
- All Interrupt Service bits reset to 0.
- All Interrupt Request bits reset to 0.
- All MSK (Interrupt Mask) bits set to 1 (mask).
- All C (Cascade) bits reset to 0 (non-cascade).
- All PRM (Priority Mask) bits set to 1, implying no levels masked.
- Initialized to non-iRMX 86 mode.

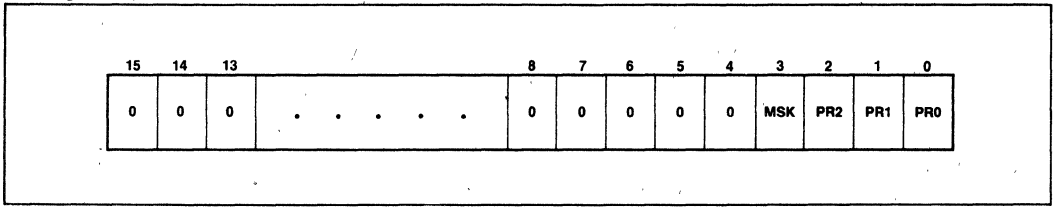


Figure 36. Control Word Format

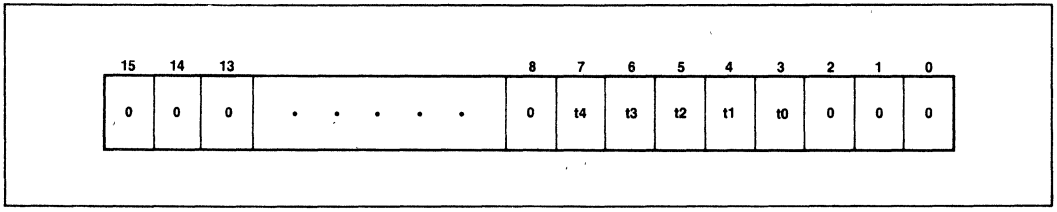


Figure 37. Interrupt Vector Register Format

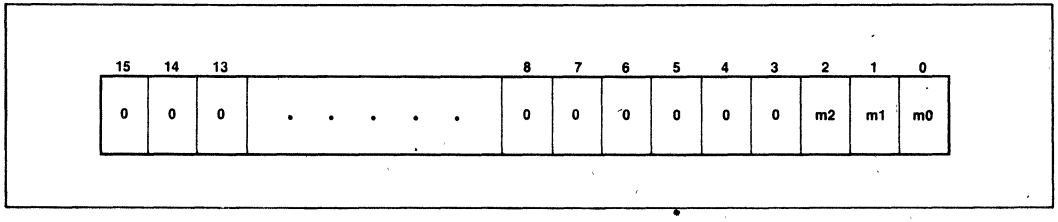


Figure 38. Priority Level Mask Register

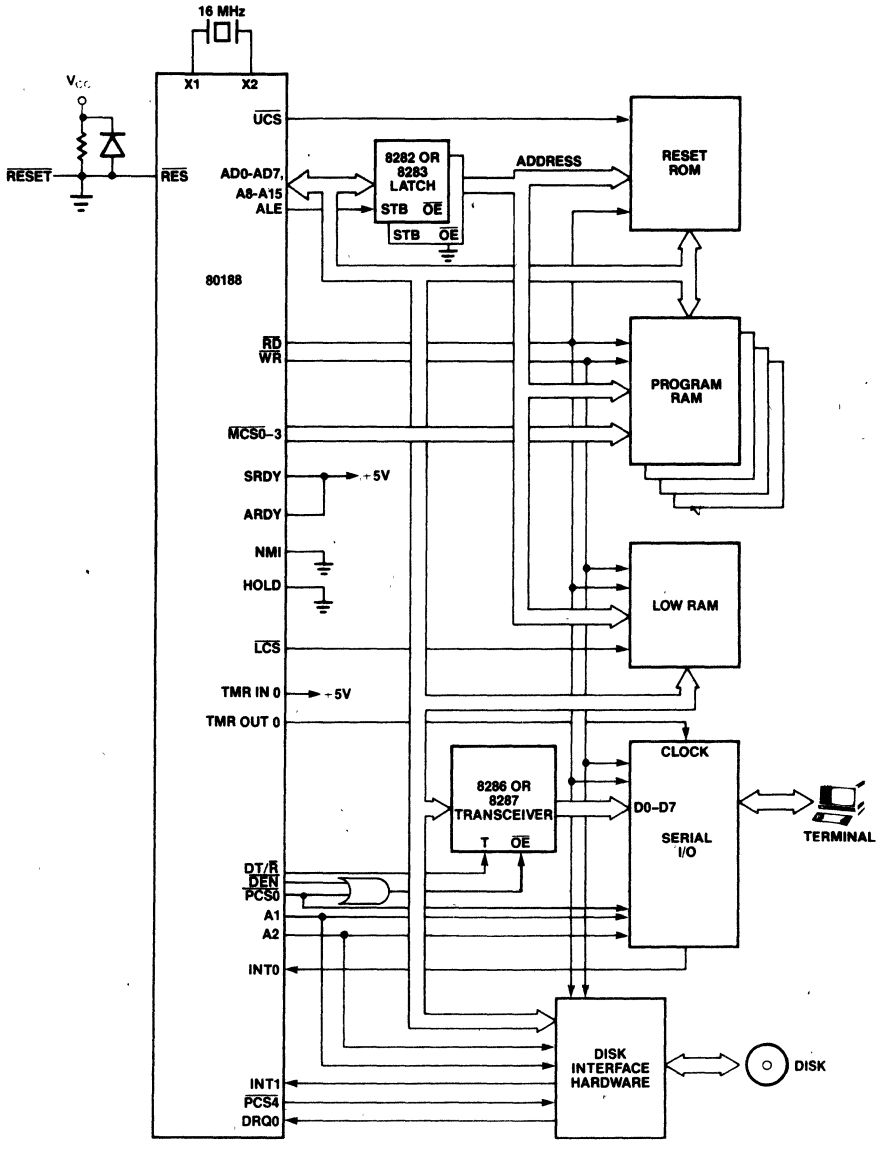


Figure 39. Typical iAPX 188 Computer

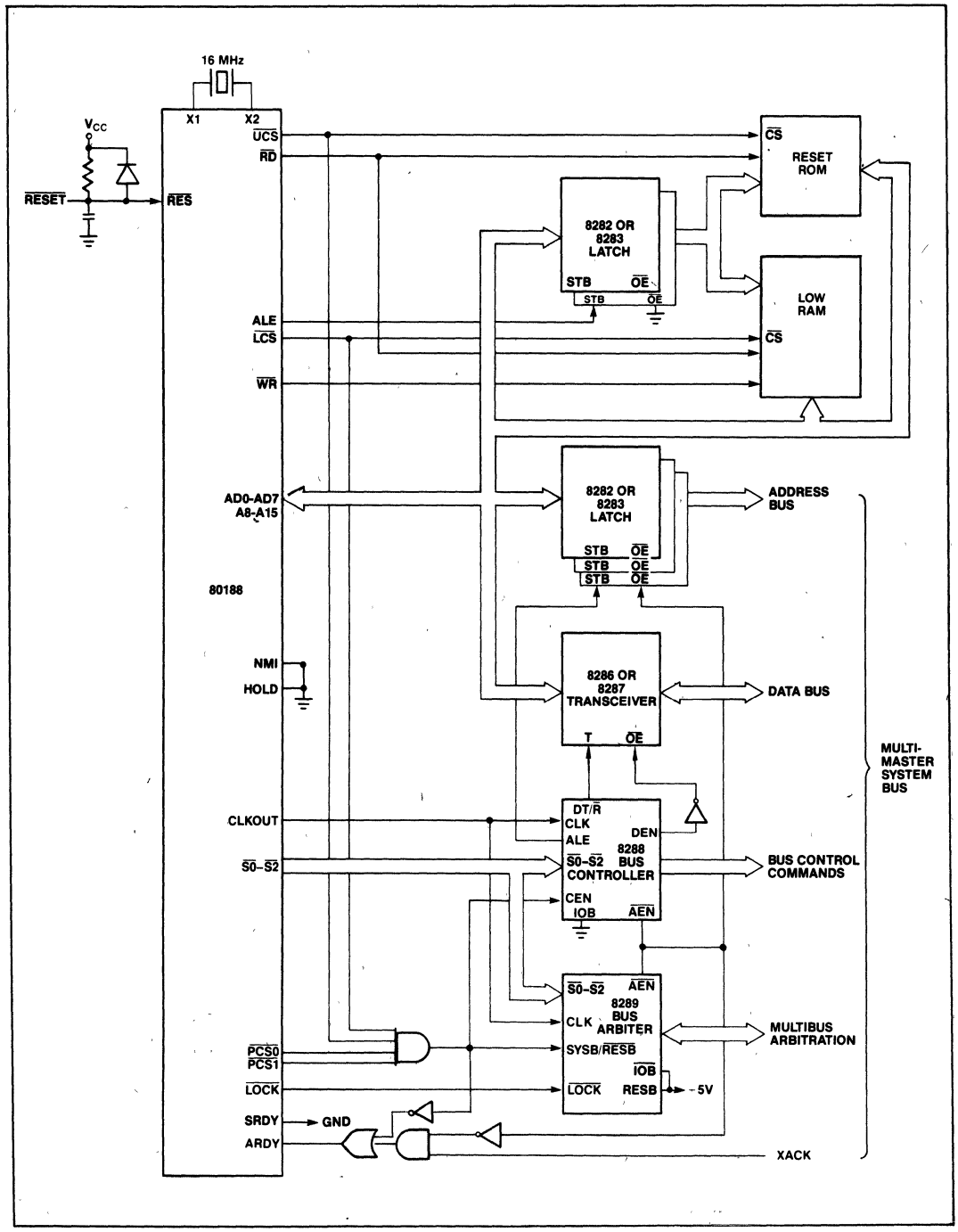


Figure 40. Typical iAPX 188 Multi-Master Bus Interface

AFN-01483A

PACKAGE

The 80188 is housed in a 68-pin, leadless JEDEC type A hermetic chip carrier. Figure 41 illustrates the package dimensions.

NOTE: The IDT 3M Textool 68-pin JEDEC Socket is required for ICE™ operation. See Figure 42 for details.

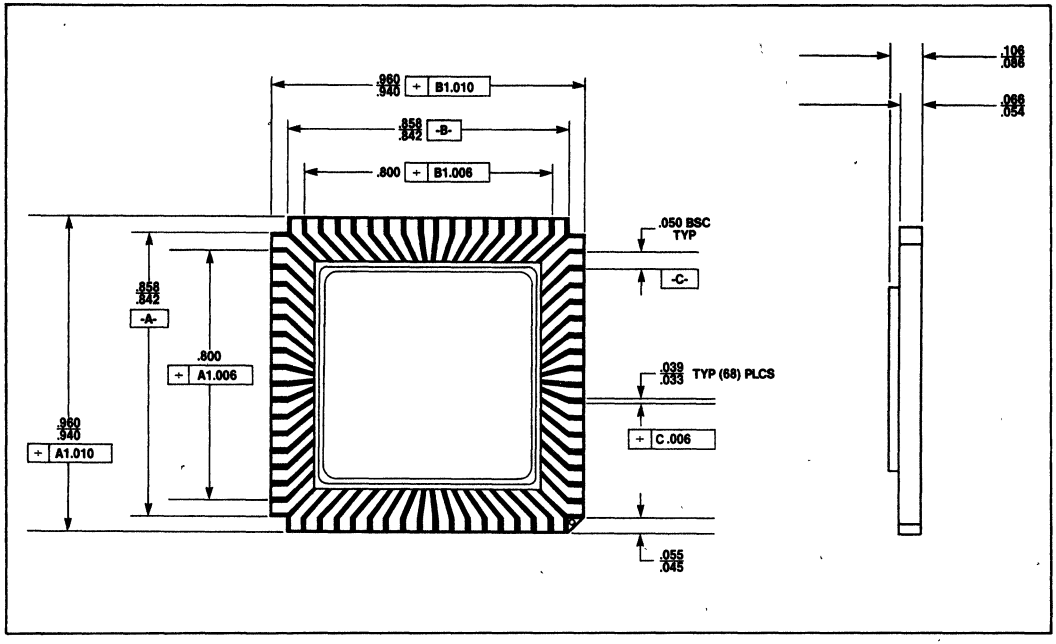
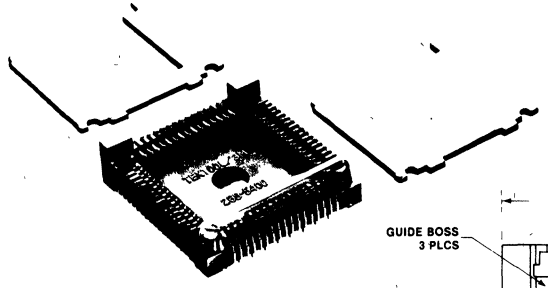
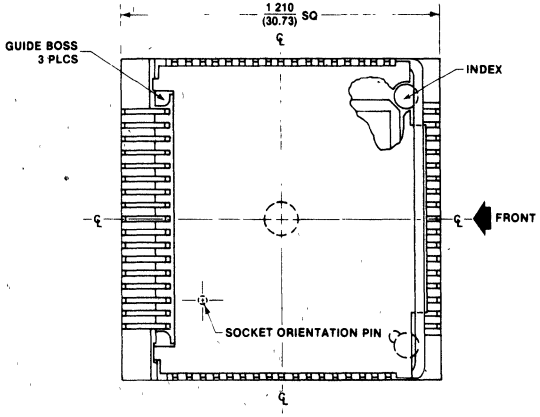


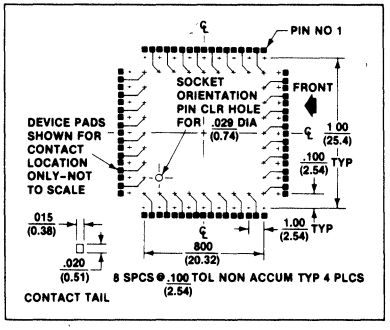
Figure 41. 80188 JEDEC Type A Package

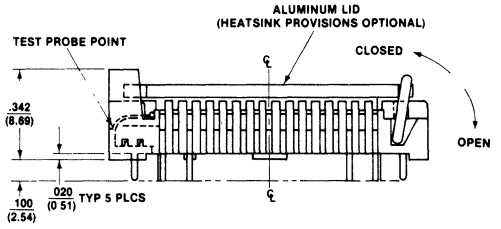
AFN-01483A





PC BOARD PATTERN





NOTE: Physical dimensions shown are for reference only. Please consult 3M Textool for complete information on the socket.

Figure 42. Textool 68 Lead Chip Carrier Socket

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 3 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°-70°C, V_{CC} = 5V < 10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	− 0.5	+ 0.8	Volts	
V_{IH}	Input High Voltage (All except X1 and $\overline{RES}$)	2.0	V_{CC} + 0.5	Volts	
V_{IH1}	Input High Voltage ($\overline{RES}$)		V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage	3.0	0.45	Volts	I_a = 2.5 mA for $\overline{S0}$-$\overline{S2}$ I_a = 2.0 mA for all other outputs
V_{OH}	Output High Voltage		2.4	Volts	I_{oa} = −400 μA
I_{CC}	Power Supply Current		550 450	mA	Max measured at T_A = 0°C T_A = 70°C
I_{LI}	Input Leakage Current		± 10	μA	OV < V_{IN} < V_{CC}
I_{LO}	Output Leakage Current		± 10	μA	0.45V < V_{OUT} < V_{CC}
V_{CLO}	Clock Output Low		0.6	Volts	I_a = 4.0 mA
V_{CHO}	Clock Output High	4.0		Volts	I_{oa} = −200 μA
V_{CLI}	Clock Input Low Voltage	−0.5	0.6	Volts	
V_{CHI}	Clock Input High Voltage	3.9	V_{CC} + 1.0	Volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

PIN TIMINGS

A.C. CHARACTERISTICS (T_A = 0°–70°C, V_{CC} = 5V ± 10%)

80188 Timing Requirements All Timings Measured At 1.5 Volts Unless Otherwise Noted.

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TDVCL	Data in Setup (A/D)	20		ns	
TCLDX	Data in Hold (A/D)	10		ns	
TARYHCH	Asynchronous Ready (AREADY) active setup time*	20		ns	
TARYLCL	AREADY inactive setup time	35		ns	
TCHARYX	AREADY hold time	15		ns	
TSRYCL	Synchronous Ready (SREADY) transition setup time	35		ns	
TCLSRY	SREADY transition hold time	15		ns	
THVCL	HOLD Setup*	25		ns	
TINVCH	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		ns	
TINVCL	DRQ0, DRQ1, Setup*	25		ns	

*To guarantee recognition at next clock.

AFN-01483A

A.C. CHARACTERISTICS (Continued)

80188 Master Interface Timing Responses

		80188 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLAV}	Address Valid Delay	5	44	5	63	ns	C_L = 20-200 pF all outputs
T_{CLAX}	Address Hold	10		10		ns	
T_{CLAZ}	Address Float Delay	T_{CLAX}	35	T_{CLAX}	44	ns	
T_{CHCZ}	Command Lines Float Delay		45		56	ns	
T_{CHCV}	Command Lines Valid Delay (after float)		55		76	ns	
T_{LHLL}	ALE Width	$T_{CLCL-35}$		$T_{CLCL-35}$		ns	
T_{CHLH}	ALE Active Delay		35		44	ns	
T_{CHLL}	ALE Inactive Delay		35		44	ns	
T_{LLAX}	Address Hold to ALE Inactive	$T_{CHCL-25}$		$T_{CHCL-30}$		ns	
T_{CLDV}	Data Valid Delay	10	44	10	55	ns	
T_{CLDOX}	Data Hold Time	10		10		ns	
T_{WHDX}	Data Hold after WR	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CVCTV}	Control Active Delay 1	5	70	5	87	ns	
T_{CHCTV}	Control Active Delay 2	10	55	10	76	ns	
T_{CVCTX}	Control Inactive Delay	5	55	5	76	ns	
T_{CVDEX}	$\overline{DEN}$ Inactive Delay (Non-Write Cycle)		70		87	ns	
T_{AZRL}	Address Float to $\overline{RD}$ Active	0		0		ns	
T_{CLRL}	$\overline{RD}$ Active Delay	10	70	10	87	ns	
T_{CLRH}	$\overline{RD}$ Inactive Delay	10	55	10	76	ns	
T_{RHAV}	$\overline{RD}$ Inactive to Address Active	$T_{CLCL-40}$		$T_{CLCL-50}$		ns	
T_{CLHAV}	HLDA Valid Delay	10	50	10	67	ns	
T_{RLRH}	$\overline{RD}$ Width	$2T_{CLCL-50}$		$2T_{CLCL-50}$		ns	
T_{WLWH}	$\overline{WR}$ Width	$2T_{CLCL-40}$		$2T_{CLCL-40}$		ns	
T_{AVAL}	Address Valid to ALE Low	$T_{CLCH-25}$		$T_{CLCH-45}$		ns	
T_{CHSV}	Status Active Delay	10	55	10	76	ns	
T_{CLSH}	Status Inactive Delay	10	55	10	76	ns	
T_{CLTMV}	Timer Output Delay		60		75	ns	100 pF max
T_{CLRO}	Reset Delay		60		75	ns	
T_{CHQSV}	Queue Status Delay		35		44	ns	

80188 Chip-Select Timing Responses

Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLCSV}	Chip-Select Active Delay		66		80	ns	
T_{CXCSX}	Chip-Selct Hold from Command Inactive	35		35		ns	
T_{CHCSX}	Chip-Select Inactive Delay	5	35	5	47	ns	

A.C. CHARACTERISTICS (Continued)

80188 CLKIN Requirements

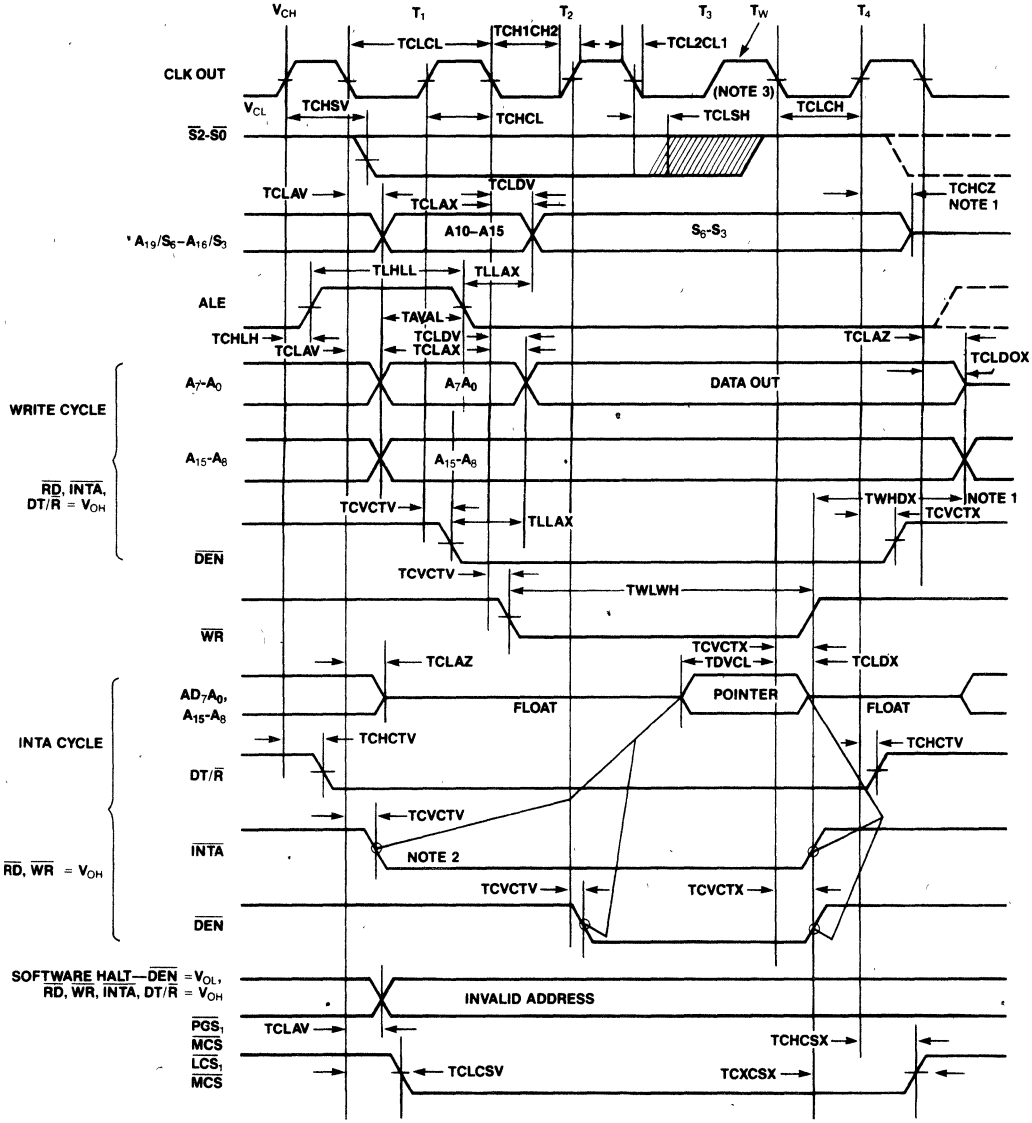
		80188 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CKIN}	CLKIN Period	62.5	250	83	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0 volts
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5 volts
T_{CLCK}	CLKIN Low Time	25		33		ns	1.5 volts
T_{CHCK}	CLKIN High Time	25		33		ns	1.5 volts

80188 CLKOUT Timing (200 pF load)

Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CICO}	CLKIN to CLKOUT Skew		50		62.5	ns	
T_{CLCL}	CLKOUT Period	125	500	167	500	ns	
T_{CLCH}	CLKOUT Low Time	$\frac{1}{2} T_{CLCL-7.5}$		$\frac{1}{2} T_{CLCL-7.5}$		ns	1.5 volts
T_{CHCL}	CLKOUT High Time	$\frac{1}{2} T_{CLCL-7.5}$		$\frac{1}{2} T_{CLCL-7.5}$		ns	1.5 volts
T_{CH1CH2}	CLKOUT Rise Time		15		15	ns	1.0 to 3.5 volts
T_{CL2CL1}	CLKOUT Fall Time		15		15	ns	3.5 to 1. volts

WAVEFORMS

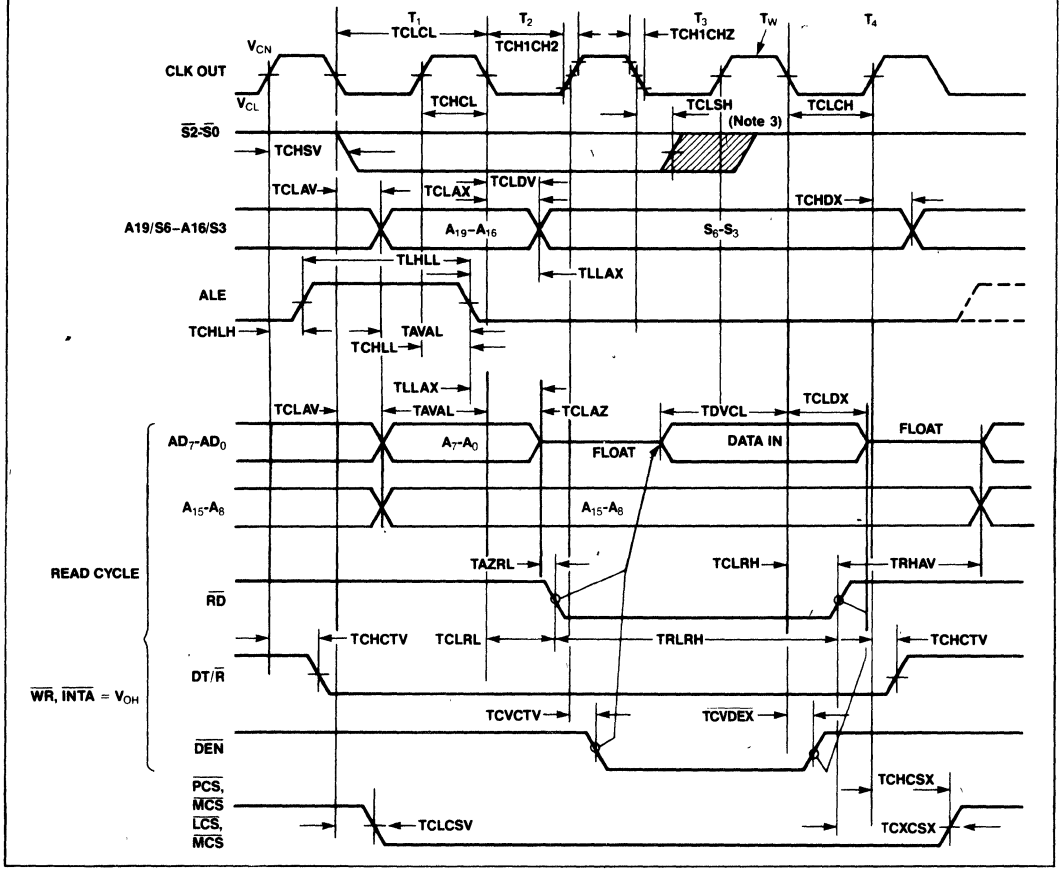
MAJOR CYCLE TIMING



AFN-01483A

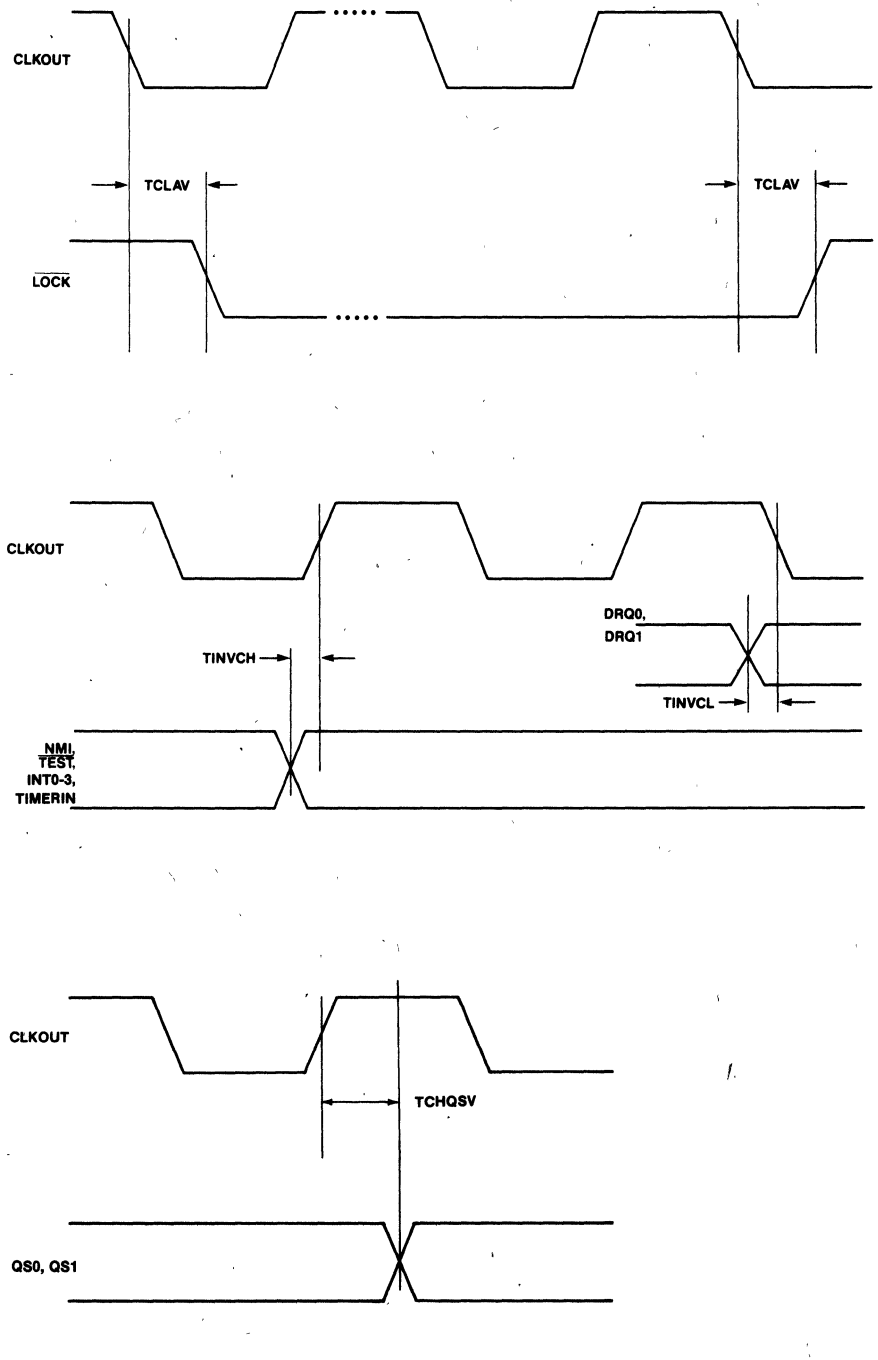
WAVEFORMS (Continued)

MAJOR CYCLE TIMING (Continued)



NOTES
1 Following a Write cycle, the Local Bus is floated by the 80188 only when the 80188 enters a "Hold Acknowledge" state
2 INTA occurs one clock later in RMX-mode
3 Status inactive just prior to T_4

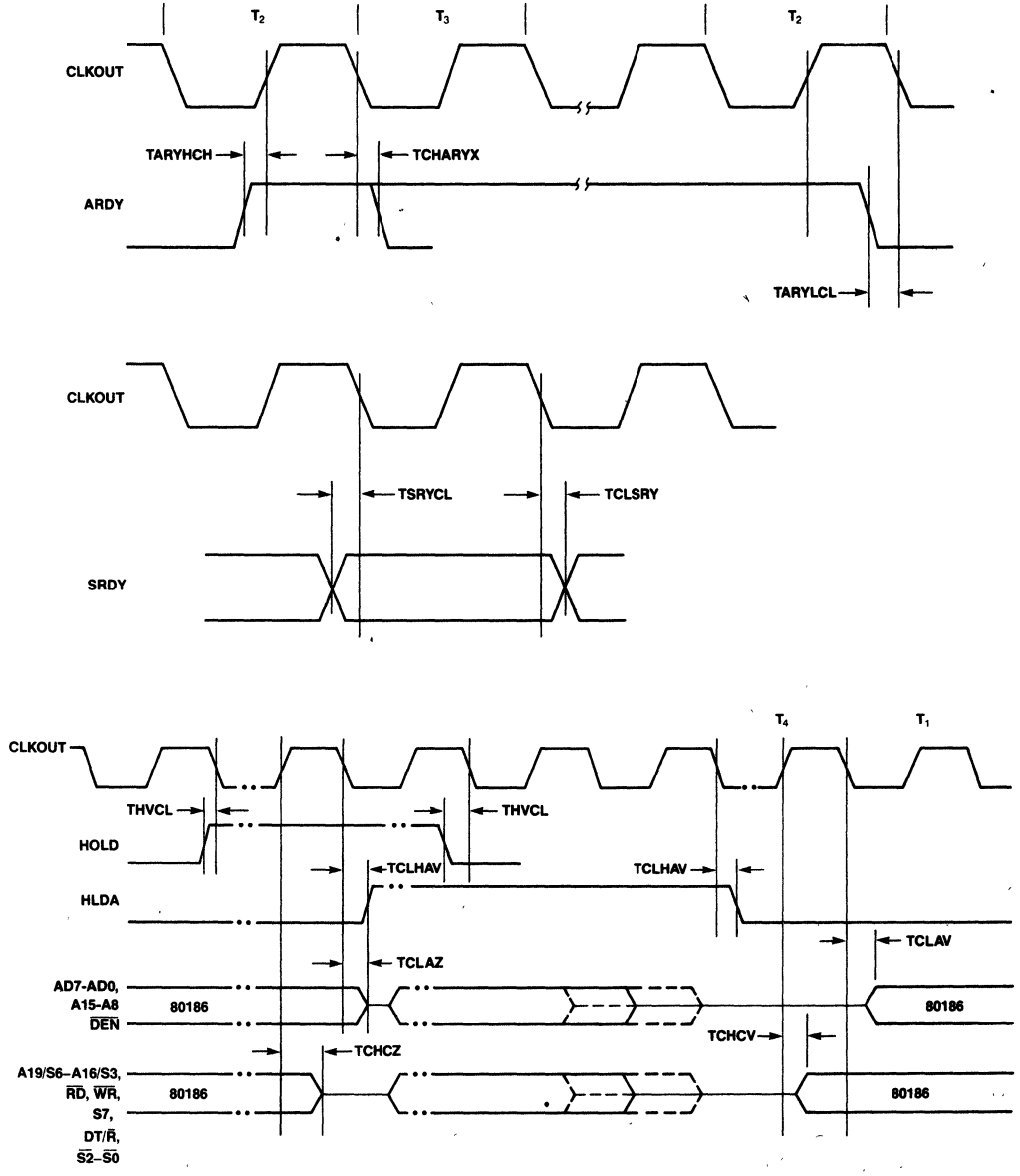
WAVEFORMS (Continued)



AFN-01483A

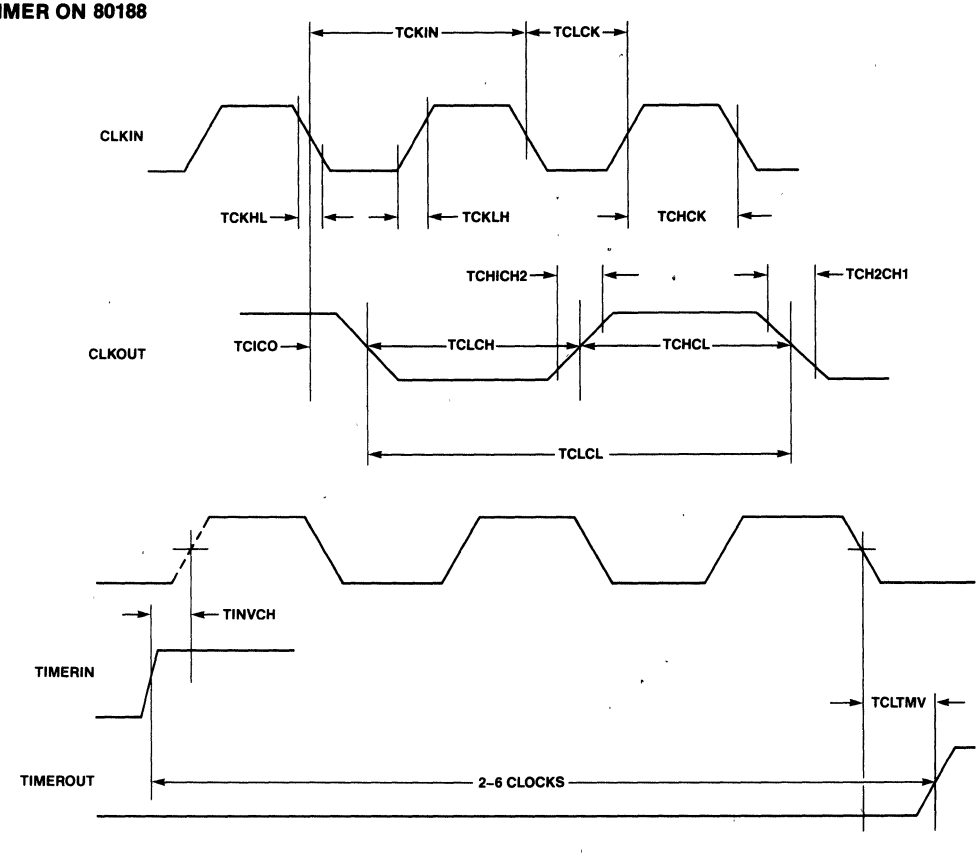
WAVEFORMS (Continued)

HOLD-HLDA TIMING



WAVEFORMS (Continued)

TIMER ON 80188



80188 INSTRUCTION TIMINGS

The following instruction timings represent the minimum execution time in clock cycles for each instruction. The timings given are based on the following assumptions:

- The opcode, along with any data or displacement required for execution of a particular instruction, has been prefetched and resides in the queue at the time it is needed.
- No wait states or bus HOLDS occur.
- All word-data is located on even-address boundaries.

All jumps and calls include the time required to fetch the opcode of the next instruction at the destination address.

All instructions which involve memory reference can require one (and in some cases, two) additional clocks above the minimum timings shown. This is due to the asynchronous nature of the handshake between the BIU and the Execution unit.

INSTRUCTION SET SUMMARY

FUNCTION	FORMAT	Clock Cycles	Comments
DATA TRANSFER			
MOV = Move:			
Register to Register/Memory	1 0 0 0 1 0 0 w | mod reg r/m	2/12*	
Register/memory to register	1 0 0 0 1 0 1 w | mod reg r/m	2/9*	
Immediate to register/memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	12–13*	8/16-bit
Immediate to register	1 0 1 1 w reg | data | data if w = 1	3–4	8/16-bit
Memory to accumulator	1 0 1 0 0 0 0 w | addr-low | addr-high	9*	
Accumulator to memory	1 0 1 0 0 0 1 w | addr-low | addr-high	8*	
Register/memory to segment register	1 0 0 0 1 1 1 0 | mod 0 reg r/m	2/13	
Segment register to register/memory	1 0 0 0 1 1 0 0 | mod 0 reg r/m	2/15	
PUSH = Push:			
Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	20	
Register	0 1 0 1 0 reg	14	
Segment register	0 0 0 reg 1 1 0	13	
Immediate	0 1 1 0 1 0 s 0 | data | data if s = 0	14	
PUSHA = Push All	0 1 1 0 0 0 0 0	68	
POP = Pop:			
Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	24	
Register	0 1 0 1 1 reg	14	
Segment register	0 0 0 reg 1 1 1 | (reg ≠ 01)	12	
POPA = Pop All	0 1 1 0 0 0 0 1	83	
XCHG = Exchange:			
Register/memory with register	1 0 0 0 0 1 1 w | mod reg r/m	4/17*	
Register with accumulator	1 0 0 1 0 reg	3	
IN = Input from:			
Fixed port	1 1 1 0 0 1 0 w | port	10*	
Variable port	1 1 1 0 1 1 0 w	8*	
OUT = Output to:			
Fixed port	1 1 1 0 0 1 1 w | port	9*	
Variable port	1 1 1 0 1 1 1 w	7*	
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	15	
LEA = Load EA to register	1 0 0 0 1 1 0 1 | mod reg r/m	6	
LDS = Load pointer to DS	1 1 0 0 0 1 0 1 | mod reg r/m | (mod ≠ 11)	26	
LES = Load pointer to ES	1 1 0 0 0 1 0 0 | mod reg r/m | (mod ≠ 11)	26	
LAHF = Load AH with flags	1 0 0 1 1 1 1 1	2	
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	3	
PUSHF = Push flags	1 0 0 1 1 1 0 0	13	
POPF = Pop flags	1 0 0 1 1 1 0 1	12	
SEGMENT = Segment Override			
CS	0 0 1 0 1 1 1 0	2	
SS	0 0 1 1 0 1 1 0	2	
DS	0 0 1 1 1 1 1 0	2	
ES	0 0 1 0 0 1 1 0	2	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
ARITHMETIC			
ADD = Add:			
Reg/memory with register to either	0 0 0 0 0 0 d w | mod reg r/m	3/10*	
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | data | data if s w = 01	4/16*	
Immediate to accumulator	0 0 0 0 0 1 0 w | data | data if w = 1	3/4	8/16-bit
ADC = Add with carry:			
Reg/memory with register to either	0 0 0 1 0 0 d w | mod reg r/m	3/10*	
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | data | data if s w = 01	4/16*	
Immediate to accumulator	0 0 0 1 0 1 0 w | data | data if w = 1	3/4	8/16-bit
INC = Increment:			
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 0 r/m	3/15*	
Register	0 1 0 0 0 reg	3	
SUB = Subtract:			
Reg/memory and register to either	0 0 1 0 1 0 d w | mod reg r/m	3/10*	
Immediate from register/memory	1 0 0 0 0 0 s w | mod 1 0 1 r/m | data | data if s w = 01	4/16*	
Immediate from accumulator	0 0 1 0 1 1 0 w | data | data if w = 1	3/4	8/16-bit
SBB = Subtract with borrow:			
Reg/memory and register to either	0 0 0 1 1 0 d w | mod reg r/m	3/10*	
Immediate from register/memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | data | data if s w = 01	4/16*	
Immediate from accumulator	0 0 0 1 1 1 0 w | data | data if w = 1	3/4	8/16-bit
DEC = Decrement:			
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 1 r/m	3/15*	
Register	0 1 0 0 1 reg	3	
CMP = Compare:			
Register/memory with register	0 0 1 1 1 0 1 w | mod reg r/m	3/10*	
Register with register/memory	0 0 1 1 1 0 0 w | mod reg r/m	3/10*	
Immediate with register/memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | data | data if s w = 01	3/10*	
Immediate with accumulator	0 0 1 1 1 1 0 w | data | data if w = 1	3/4	8/16-bit
NEG = Change sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	3	
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	8	
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	4	
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	7	
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	4	
MUL = Multiply (unsigned)	1 1 1 1 0 1 1 w | mod 1 0 0 r/m		
Register-Byte		26–28	
Register-Word		35–37	
Memory-Byte		32–34	
Memory-Word		41–43*	
IMUL = Integer multiply (signed)	1 1 1 1 0 1 1 w | mod 1 0 1 r/m		
Register-Byte		25–28	
Register-Word		34–37	
Memory-Byte		31–34	
Memory-Word		40–43*	
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 | mod reg r/m | data | data if s = 0	22–25/29–32*	
DIV = Divide (unsigned)	1 1 1 1 0 1 1 w | mod 1 1 0 r/m		
Register-Byte		29	
Register-Word		38	
Memory-Byte		35	
Memory-Word		44*	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
ARITHMETIC (Continued):			
IDIV = Integer divide (signed)	1 1 1 1 0 1 1 w | mod 1 1 1 r/m		
Register-Byte		44–52	
Register-Word		53–61	
Memory-Byte		50–58	
Memory-Word		59–67*	
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	19	
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	15	
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	
CWD = Convert word to double word	1 0 0 1 1 0 0 1	4	
LOGIC			
Shift/Rotate Instructions:			
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	2/15*	
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	5+n/17+n*	
Register/Memory by Count	1 1 0 0 0 0 0 w | mod TTT r/m | count	5+n/17+n*	
	TTT Instruction 0 0 0 ROL 0 0 1 ROR 0 1 0 RCL 0 1 1 RCR 1 0 0 SHL/SAL 1 0 1 SHR 1 1 1 SAR		
AND = And:			
Reg/memory and register to either	0 0 1 0 0 0 d w | mod reg r/m	3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | data | data if w = 1	4/16*	
Immediate to accumulator	0 0 1 0 0 1 0 w | data | data if w = 1	3/4	8/16-bit
TEST = And function to flags, no result:			
Register/memory and register	1 0 0 0 0 1 0 w | mod reg r/m	3/10*	
Immediate data and register/memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	4/10*	
Immediate data and accumulator	1 0 1 0 1 0 0 w | data | data if w = 1	3/4	8/16-bit
OR = Or:			
Reg/memory and register to either	0 0 0 0 1 0 d w | mod reg r/m	3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | data | data if w = 1	4/16*	
Immediate to accumulator	0 0 0 0 1 1 0 w | data | data if w = 1	3/4	8/16-bit
XOR = Exclusive or:			
Reg/memory and register to either	0 0 1 1 0 0 d w | mod reg r/m	3/10*	
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | data | data if w = 1	4/16*	
Immediate to accumulator	0 0 1 1 0 1 0 w | data | data if w = 1	3/4	8/16-bit
NOT = Invert register/memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	3	
STRING MANIPULATION:			
MOVS = Move byte/word	1 0 1 0 0 1 0 w	14*	
CMPS = Compare byte/word	1 0 1 0 0 1 1 w	22*	
SCAS = Scan byte/word	1 0 1 0 1 1 1 w	15*	
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w	12*	
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w	10*	
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	14*	
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	14*	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
STRING MANIPULATION (Continued)			
Repeated by count in CX			
MOVS Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	8+8n*	
CMPS Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+22n*	
SCAS Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+15n*	
LODS Load string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	6+11n*	
STOS Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	6+9n*	
INS Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	8+8n*	
OUTS Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	8+8n*	
CONTROL TRANSFER			
CALL = Call:			
Direct within segment	1 1 1 0 1 0 0 0 | disp-low | disp-high	18	
Register memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r m	17/27	
Direct intersegment	1 0 0 1 1 0 1 0 | segment offset | segment selector	31	
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r m (mod = 11)	54	
JMP = Unconditional jump:			
Short long	1 1 1 0 1 0 1 1 | disp-low	13	
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	13	
Register memory indirect within segment	1 1 1 1 1 1 1 1 | mod 1 0 0 r m	11/21	
Direct intersegment	1 1 1 0 1 0 1 0 | segment offset | segment selector	13	
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r m (mod = 11)	34	
RET = Return from CALL:			
Within segment	1 1 0 0 0 0 1 1	20	
Within seg adding immed to SP	1 1 0 0 0 0 1 0 | data-low | data-high	22	
Intersegment	1 1 0 0 1 0 1 1	30	
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0 | data-low | data-high	33	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

*Note: Clock cycles shown for byte transfer. For word operations, add 4 clock cycles for all memory transfers.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
CONTROL TRANSFER (Continued):			
JE/JZ = Jump on equal zero	0 1 1 1 0 1 0 0 | disp	4/13	JMP not taken/JMP taken
JL/JNGE = Jump on less not greater or equal	0 1 1 1 1 1 0 0 | disp	4/13	
JLE/JNG = Jump on less or equal not greater	0 1 1 1 1 1 1 0 | disp	4/13	
JB/JNAE = Jump on below not above or equal	0 1 1 1 0 0 1 0 | disp	4/13	
JBE/JNA = Jump on below or equal not above	0 1 1 1 0 1 1 0 | disp	4/13	
JP/JPE = Jump on panty panty even	0 1 1 1 1 0 1 0 | disp	4/13	
JO = Jump on overflow	0 1 1 1 0 0 0 0 | disp	4/13	
JS = Jump on sign	0 1 1 1 1 0 0 0 | disp	4/13	
JNE/JNZ = Jump on not equal not zero	0 1 1 1 0 1 0 1 | disp	4/13	
JNL/JGE = Jump on not less greater or equal	0 1 1 1 1 1 0 1 | disp	4/13	
JNLE/JG = Jump on not less or equal greater	0 1 1 1 1 1 1 1 | disp	4/13	
JNB/JAE = Jump on not below above or equal	0 1 1 1 0 0 1 1 | disp	4/13	
JNBE/JA = Jump on not below or equal above	0 1 1 1 0 1 1 1 | disp	4/13	
JNP/JPO = Jump on not par par odd	0 1 1 1 1 0 1 1 | disp	4/13	
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 | disp	4/13	
JNS = Jump on not sign	0 1 1 1 1 0 0 1 | disp	4/13	
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1 | disp	5/15	
LOOP = Loop CX times	1 1 1 0 0 0 1 0 | disp	6/16	LOOP not taken/LOOP taken
LOOPZ/LOOPE = Loop while zero equal	1 1 1 0 0 0 0 1 | disp	6/16	
LOOPNZ/LOOPNE = Loop while not zero equal	1 1 1 0 0 0 0 0 | disp	6/16	
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 | data-low | data-high | L		
L = 0		15	
L = 1		25	
L > 1		22 + 16(n − 1)	
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	8	
INT = Interrupt:			
Type specified	1 1 0 0 1 1 0 1 | type	47	
Type 3	1 1 0 0 1 1 0 0	45	if INT. taken/ if INT. not taken
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	48/4	
IRET = Interrupt return	1 1 0 0 1 1 1 1	28	
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r/m	33–35	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	Clock Cycles	Comments
PROCESSOR CONTROL			
CLC = Clear carry	1 1 1 1 1 0 0 0	2	
CMC = Complement carry	1 1 1 1 0 1 0 1	2	
STC = Set carry	1 1 1 1 1 0 0 1	2	
CLD = Clear direction	1 1 1 1 1 1 0 0	2	
STD = Set direction	1 1 1 1 1 1 0 1	2	
CLI = Clear interrupt	1 1 1 1 1 0 1 0	2	
STI = Set interrupt	1 1 1 1 1 0 1 1	2	
HLT = Halt	1 1 1 1 0 1 0 0	2	
WAIT = Wait	1 0 0 1 1 0 1 1	6	if $\overline{\text{test}}$ = 0
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0	2	
ESC = Processor Extension Escape	1 0 0 1 1 T T T | mod LLL r m (TTT LLL are opcode to processor extension)	6	

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

FOOTNOTES

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

EA calculation time is 4 clock cycles for all modes, and is included in the execution times given whenever appropriate.

SEGMENT OVERRIDE PREFIX

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

8089
8 & 16-BIT HMOS I/O PROCESSOR

- **High Speed DMA Capabilities Including I/O to Memory, Memory to I/O, Memory to Memory, and I/O to I/O**
- **iAPX 86, 88 Compatible: Removes I/O Overhead from CPU in iAPX 86/11 or 88/11 Configuration**
- **Allows Mixed Interface of 8- & 16-Bit Peripherals, to 8- & 16-Bit Processor Busses**
- **1 Mbyte Addressability**
- **Memory Based Communication with CPU**
- **Supports LOCAL or REMOTE I/O Processing**
- **Flexible, Intelligent DMA Functions Including Translation, Search, Word Assembly/Disassembly**
- **MULTIBUS™ Compatible System Interface**
- **Available in EXPRESS - Standard Temperature Range**

The Intel® 8089 is a revolutionary concept in microprocessor input/output processing. Packaged in a 40-pin DIP package, the 8089 is a high performance processor implemented in N-channel, depletion load silicon gate technology (HMOS). The 8089's instruction set and capabilities are optimized for high speed, flexible and efficient I/O handling. It allows easy interface of Intel's 16-bit iAPX 86 and 8-bit iAPX 88 microprocessors with 8- and 16-bit peripherals. In the REMOTE configuration, the 8089 bus is user definable allowing it to be compatible with any 8/16-bit Intel microprocessor, interfacing easily to the Intel multiprocessor system bus standard MULTIBUS™.

The 8089 performs the function of an intelligent DMA controller for the Intel iAPX 86, 88 family and with its processing power, can remove I/O overhead from the iAPX 86 or iAPX 88. It may operate completely in parallel with a CPU, giving dramatically improved performance in I/O intensive applications. The 8089 provides two I/O channels, each supporting a transfer rate up to 1.25 mbyte/sec at the standard clock frequency of 5 MHz. Memory based communication between the IOP and CPU enhances system flexibility and encourages software modularity, yielding more reliable, easier to develop systems.

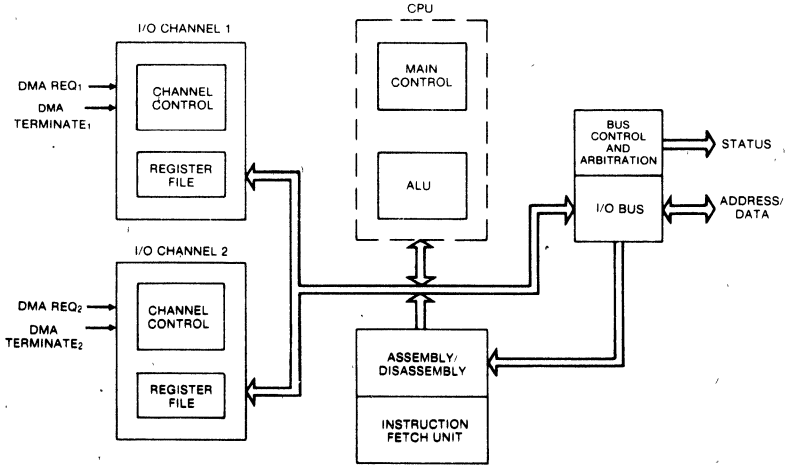


Figure 1. 8089 I/O Processor Block Diagram

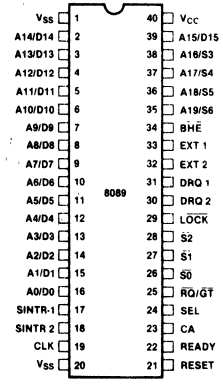


Figure 2. 8089 Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
A0–A15/ D0–D15	I/O	**Multiplexed Address and Data Bus:** The function of these lines are defined by the state of $\overline{S0}$, $\overline{S1}$ and $\overline{S2}$ lines. The pins are floated after reset and when the bus is not acquired. A8–A15 are stable on transfers to a physical 8-bit data bus (same bus as 8088), and are multiplexed with data on transfers to a 16-bit physical bus.
A16–A19/ S3–S6	O	**Address and Status:** Multiplexed most significant address lines and status information. The address lines are active only when addressing memory. Otherwise, the status lines are active and are encoded as shown below. The pins are floated after reset and when the bus is not acquired. **S6 S5 S4 S3** 1 1 0 0 DMA cycle on CH1 1 1 0 1 DMA cycle on CH2 1 1 1 0 Non-DMA cycle on CH1 1 1 1 1 Non-DMA cycle on CH2
$\overline{BHE}$	O	**Bus High Enable:** The Bus High Enable is used to enable data operations on the most significant half of the data bus (D8–D15). The signal is active low when a byte is to be transferred on the upper half of the data bus. The pin is floated after reset and when the bus is not acquired. $\overline{BHE}$ does not have to be latched.
$\overline{S0}$, $\overline{S1}$, $\overline{S2}$	O	**Status:** These are the status pins that define the IOP activity during any given cycle. They are encoded as shown below: **$\overline{S2}$ $\overline{S1}$ $\overline{S0}$** 0 0 0 Instruction fetch; I/O space 0 0 1 Data fetch; I/O space 0 1 0 Data store; I/O space 0 1 1 Not used 1 0 0 Instruction fetch; System Memory 1 0 1 Data fetch; System Memory 1 1 0 Data store; System Memroy 1 1 1 Passive The status lines are utilized by the bus controller and bus arbiter to generate all memory and I/O control signals. The signals change during T4 if a new cycle is to be entered while the return to passive state in T3 or T_W indicates the end of a cycle. The pins are floated after system reset and when the bus is not acquired.
READY	I	**Ready:** The ready signal received from the addressed device indicates that the device is ready for data transfer. The signal is active high and is synchronized by the 8284 clock generator.

Symbol	Type	Name and Function
$\overline{LOCK}$	O	**Lock:** The lock output signal indicates to the bus controller that the bus is needed for more than one contiguous cycle. It is set via the channel control register, and during the TSL instruction. The pin floats after reset and when the bus is not acquired. This output is active low.
RESET	I	**Reset:** The receipt of a reset signal causes the IOP to suspend all its activities and enter an idle state until a channel attention is received. The signal must be active for at least four clock cycles.
CLK	I	**Clock:** Clock provides all timing needed for internal IOP operation.
CA	I	**Channel Attention:** Gets the attention of the IOP. Upon the falling edge of this signal, the SEL input pin is examined to determine Master/Slave or CH1/CH2 information. This input is active high.
SEL	I	**Select:** The first CA received after system reset informs the IOP via the SEL line, whether it is a Master or Slave (0/1 for Master/Slave respectively) and starts the initialization sequence. During any other CA the SEL line signifies the selection of CH1/CH2. (0/1 respectively.)
DRQ1–2	I	**Data Request:** DMA request inputs which signal the IOP that a peripheral is ready to transfer/receive data using channels 1 or 2 respectively. The signals must be held active high until the appropriate fetch/stroke is initiated.
$\overline{RQ}/\overline{GT}$	I/O	**Request Grant:** Request Grant implements the communication dialogue required to arbitrate the use of the system bus (between IOP and CPU, LOCAL mode) or I/O bus when two IOPs share the same bus (REMOTE mode). The $\overline{RQ}/\overline{GT}$ signal is active low. An internal pull-up permits $\overline{RQ}/\overline{GT}$ to be left floating if not used.
SINTR1–2	O	**Signal Interrupt:** Signal Interrupt outputs from channels 1 and 2 respectively. The interrupts may be sent directly to the CPU or through the 8295A interrupt controller. They are used to indicate to the system the occurrence of user defined events.
EXT1–2	I	**External Terminate:** External terminate inputs for channels 1 and 2 respectively. The EXT signals will cause the termination of the current DMA transfer operation if the channel is so programmed by the channel control register. The signal must be held active high until termination is complete.
V_{CC}		**Voltage:** +5 volt power input.
V_{SS}		**Ground.**

FUNCTIONAL DESCRIPTION

The 8089 IOP has been designed to remove I/O processing, control and high speed transfers from the central processing unit. Its major capabilities include that of initializing and maintaining peripheral components and supporting versatile DMA. This DMA function boasts flexible termination conditions (such as external terminate, mask compare, single transfer and byte count expired). The DMA function of the 8089 IOP uses a two cycle approach where the information actually flows through the 8089 IOP. This approach to DMA vastly simplifies the bus timings and enhances compatibility with memory and peripherals, in addition to allowing operations to be performed on the data as it is transferred. Operations can include such constructs as translate, where the 8089 automatically vectors through a lookup table and mask compare, both on the "fly".

The 8089 is functionally compatible with Intel's iAPX 86, 88 family. It supports any combination of 8/16-bit busses. In the REMOTE mode it can be used to complement other Intel processor families. Hardware and communication architecture are designed to provide simple mechanisms for system upgrade.

The only direct communication between the IOP and CPU is handled by the Channel Attention and Interrupt lines. Status information, parameters and task programs are passed via blocks of shared memory, simplifying hardware interface and encouraging structured programming.

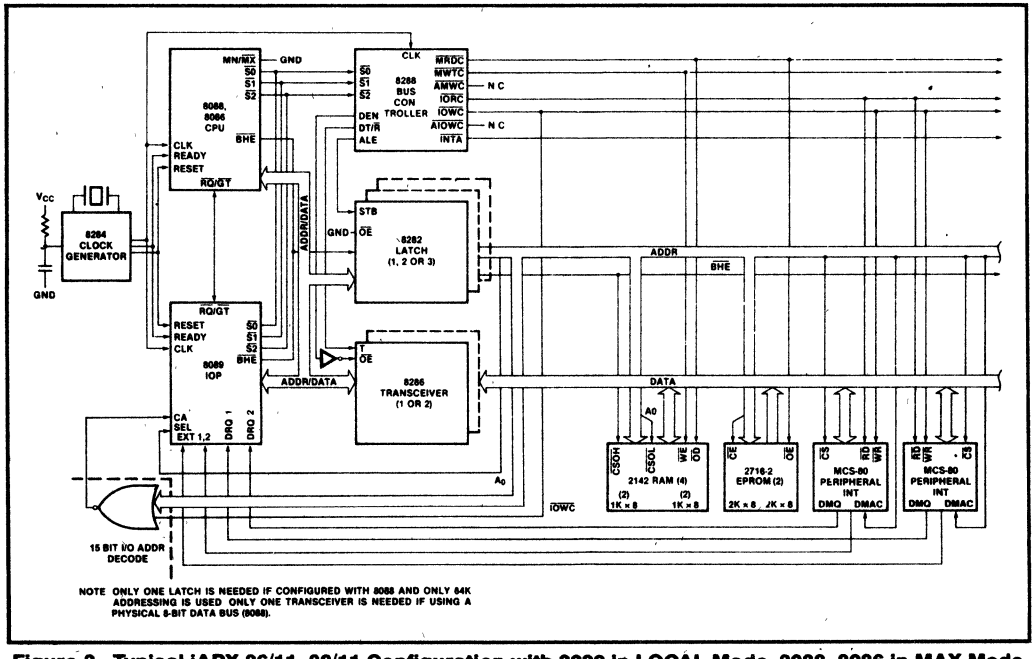
The 8089 can be used in applications such as file and buffer management in hard disk or floppy disk control. It can also provide for soft error recovery routines and scan control. CRT control, such as cursor control and auto scrolling, is simplified with the 8089. Keyboard control, communication control and general I/O are just a few of the typical applications for the 8089.

Remote and Local Modes

Shown in Figure 3 is the 8089 in a LOCAL configuration. The iAPX 86 (or iAPX 88) is used in its maximum mode. The 8089 and iAPX 86 reside on the same local bus, sharing the same set of system buffers. Peripherals located on the system bus can be addressed by either the iAPX 86 or the 8089. The 8089 requests the use of the LOCAL bus by means of the RQ/GT line. This performs a similar function to that of HOLD and HLDA on the Intel 8085A, 8080A and iAPX 86 minimum mode, but is implemented on one physical line. When the iAPX 86 relinquishes the system bus, the 8089 uses the same bus control, latches and transceiver components to generate the system address, control and data lines. This mode allows a more economical system configuration at the expense of reduced CPU thruput due to IOP bus utilization.

A typical REMOTE configuration is shown in Figure 4. In this mode, the IOP's bus is physically separated from the system bus by means of system buffers/latches. The IOP maintains its own local bus and can operate out of local or system memory. The system bus interface contains the following components:

- Up to three 8282 buffer/latches to latch the address to the system bus.
- Up to two 8286 devices bidirectionally buffer the system data bus.



NOTE ONLY ONE LATCH IS NEEDED IF CONFIGURED WITH 8088 AND ONLY 64K ADDRESSING IS USED ONLY ONE TRANSCEIVER IS NEEDED IF USING A PHYSICAL 8-BIT DATA BUS (8088).

Figure 3. Typical iAPX 86/11, 88/11 Configuration with 8089 in LOCAL Mode, 8088, 8086 in MAX Mode

- An 8288 bus controller supplies the control signals necessary for buffer operation as well as MRDC (Memory Read) and MWTC (Memory Write) signals.
- An 8289 bus arbiter performs all the functions necessary to arbitrate the use of the system bus. This is used in place of the $\overline{RQ}/\overline{GT}$ logic in the LOCAL mode. This arbiter decodes type of cycle information from the 8089 status lines to determine if the IOP desires to perform a transfer over the "common" or system bus.

The peripheral devices PER1 and PER2 are supported on their own data and address bus. the 8089 communicates with the peripherals without affecting system bus operation. Optional buffers may be used on the local bus when capacitive loading conditions so dictate. I/O programs and RAM buffers may also reside on the local bus to further reduce system bus utilization.

COMMUNICATION MECHANISM

Fundamentally, communication between the CPU and IOP is performed through messages prepared in shared memory. The CPU can cause the 8089 to execute a program by placing it in the 8089's memory space and/or directing the 8089's attention to it by asserting a hardware Channel Attention (CA) signal to the IOP, activating the proper I/O channel. The SEL Pin indicates to the IOP which channel is being addressed. Communication from the IOP to the processor can be performed in a similar manner via a system interrupt (SINTR 1,2), if the CPU has enabled interrupts for this purpose. Additionally, the 8089 can store messages in memory regarding its status and the status of any peripherals. This communication mechanism is supported by a hierarchial data structure to provide a maximum amount of flexibility of memory use with the added capability of handling multiple IOP's.

Illustrated in Figure 5 is an overview of the communication data structure hierarchy that exists for the 8089 I/O processor. Upon the first CA from RESET, if the IOP is initialized as the BUS MASTER, 5 bytes of information are read into the 8089 starting at location FFFF6 (FFFF6, FFFF8-FFFFB) where the type of system bus (16-bit or 8-bit) and pointers to the system configuration block are obtained. This is the only fixed location the 8089 accesses. The remaining addresses are obtained via the data structure hierarchy. The 8089 determines addresses in the same manner as does the iAPX 86; i.e., a 16-bit relocation pointer is offset left 4 bits and added to the 16-bit address offset, obtaining a 20-bit address. Once these 20-bit addresses are formed, they are stored as such, as all the 8089 address registers are 20 bits long. After the system configuration pointer address is formed, the 8089 IOP accesses the system configuration block.

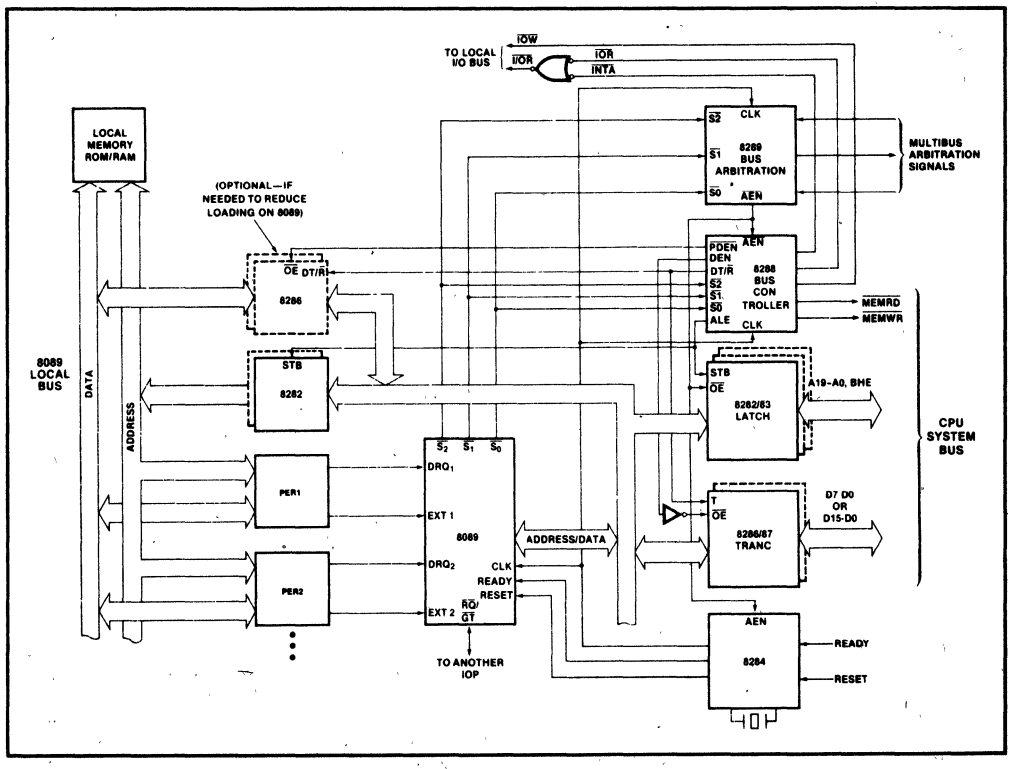


Figure 4. Typical REMOTE Configuration

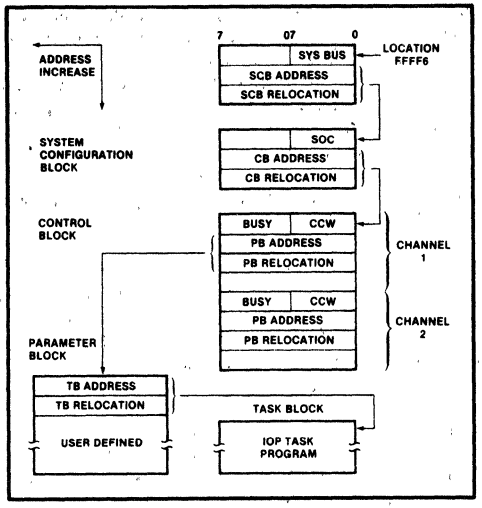


Figure 5. Communication Data Structure Hierarchy

The System Configuration Block (SCB), used only during startup, points to the Control Block (CB) and provides IOP system configuration data via the SOC byte. The SOC byte initializes IOP I/O bus width to 8/16, and defines one of two IOP $\overline{RQ}/\overline{GT}$ operating modes. For $\overline{RQ}/\overline{GT}$ mode 0, the IOP is typically initialized as SLAVE and has its $\overline{RQ}/\overline{GT}$ line tied to a MASTER CPU (typical LOCAL configuration). In this mode, the CPU normally has control of the bus, grants control to the IOP as needed, and has the bus restored to it upon IOP task completion (IOP request—CPU grant—IOP done). For $\overline{RQ}/\overline{GT}$ mode 1, useful only in remote mode between two IOPs, MASTER/SLAVE designation is used only to initialize bus control: from then on, each IOP requests and grants as the bus is needed (IOP1 request—IOP2 grant—IOP2 request—IOP1 grant). Thus, each IOP retains bus control until the other requests it. The completion of initialization is signalled by the IOP clearing the BUSY flag in the CB. This type of startup allows the user to have the startup pointers in ROM with the SCB in RAM. Allowing the SCB to be in RAM gives the user the flexibility of being able to initialize multiple IOPs.

The Control Block furnishes bus control Initialization for the IOP operation (CCW or Channel Control Word) and provides pointers to the Parameter Block or "data" memory for both channels 1 and 2. The CCW is retrieved and analyzed upon all CA's other than the first after a reset. The CCW byte is decoded to determine channel operation.

The Parameter Block contains the address of the Task Block and acts as a messge center between the IOP and CPU. Parameters or variable information is passed from the CPU to its IOP in this block to customize the software interface to the peripheral device. It is also used for transferring data and status information between the IOP and CPU.

The Task Block contains the instructions for the respective channel. This block can reside on the local bus of the IOP, allowing the IOP to operate concurrently with the CPU, or reside in system memory.

The advantage of this type of communication between the processor, IOP and peripheral, is that it allows for a very clean method for the operating system to handle I/O routines. Canned programs or "Task Blocks" allow for execution of general purpose I/O routines with the status and peripheral command information being passed via the Parameter Block ("data" memory). Task Blocks (or "program" memory) can be terminated or restarted by the CPU, if need be. Clearly, the flexibility of this communication lends itself to modularity and applicability to a large number of peripheral devices and upward compatibility to future end user systems and microprocessor families.

Register Set

The 8089 maintains separate registers for its two I/O channels as well as some common registers (see Figure 6). There are sufficient registers for each channel to sustain its own DMA transfers, and process its own instruction stream. The basic DMA pointer registers (GA, GB—20 bits each), can point to either the system bus or local bus, DMA source or destination, and can be autoincremented. A third register set (GC) can be used to allow translation during the DMA process through a lookup table it points to. The channel control register, which may be accessed only by a MOV, or MOVI instruction, determines the mode of the channel operation. Additionally, registers are provided for a masked compare during the data transfer and can be set up to act as one of the termination conditions. Other registers are also provided. Many of these registers can be used as general purpose registers during program execution, when the IOP is not performing DMA cycles.

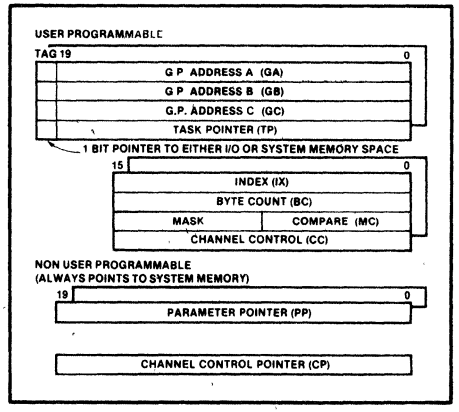


Figure 6. Register Model

Bus Operation

The 8089 utilizes the same bus structure as the iAPX 86, 88 in their maximum mode configurations (see Figure 7). The address is time multiplexed with the data on the first 16/8 lines. A16 through A19 are time multiplexed with four status lines S3-S6. For 8089 cycles, S4 and S3 determine what type of cycle (DMA versus non-DMA) is being performed on channels 1 or 2. S5 and S6

are a unique code assigned to the 8089 IOP, enabling the user to detect which processor is performing a bus cycle in a multiprocessing environment.

The first three status lines, S0-S2, are used with an 8288 bus controller to determine if an instruction fetch or data transfer is being performed in I/O or system memory space.

DMA transfers require at least two bus cycles with each bus cycle requiring a minimum of four clock cycles. Additional clock cycles are added if wait states are required. This two cycle approach simplifies considerably the bus timings in burst DMA. The 8089 optimizes the transfer between two different bus widths by using three bus cycles versus four to transfer 1 word. More than one read (write) is performed when mapping an 8-bit bus onto a 16-bit bus (vice versa). For example, a data transfer from an 8-bit peripheral to a 16-bit physical location in memory is performed by first doing two reads, with word assembly within the IOP assembly register file and then one write.

As can be expected, the data bandwidth of the IOP is a function of the physical bus width of the system and I/O busses. Table 2 gives the bandwidth, latency and bus utilization of the 8089. The system bus is assumed to be 16-bits wide with either an 8-bit peripheral (under byte column) or 16-bit peripheral (word column) being shown.

The latency refers to the worst case response time by the IOP to a DMA request, without the bus arbitration times. Notice that the word transfer allows 50% more bandwidth. This occurs since three bus cycles are required to map 8-bit data into a 16-bit location, versus two for a 16-bit to 16-bit transfer. Note that it is possible to fully saturate the system bus in the LOCAL mode whereas in the REMOTE mode this is reduced to a maximum of 50%.

Table 2. Achievable 5 MHz 8089 Operations with a 16-Bit System Bus

	Local		Remote	
	Byte	Word	Byte	Word
Bandwidth	830 KB/S	1250 KB/S	830 KB/S	1250 KB/S
Latency	1.0/2.4 μsec*	1.0/2.4 μsec*	1.0/2.4 μsec*	1.0/2.4 μsec*
System Bus Utilization	2.4 μsec PER TRANSFER	1.6 μsec PER TRANSFER	0.8 μsec PER TRANSFER	0.8 μsec PER TRANSFER

*2.4 μsec if interleaving with other channel and no wait states. 1μsec if channel is waiting for request.

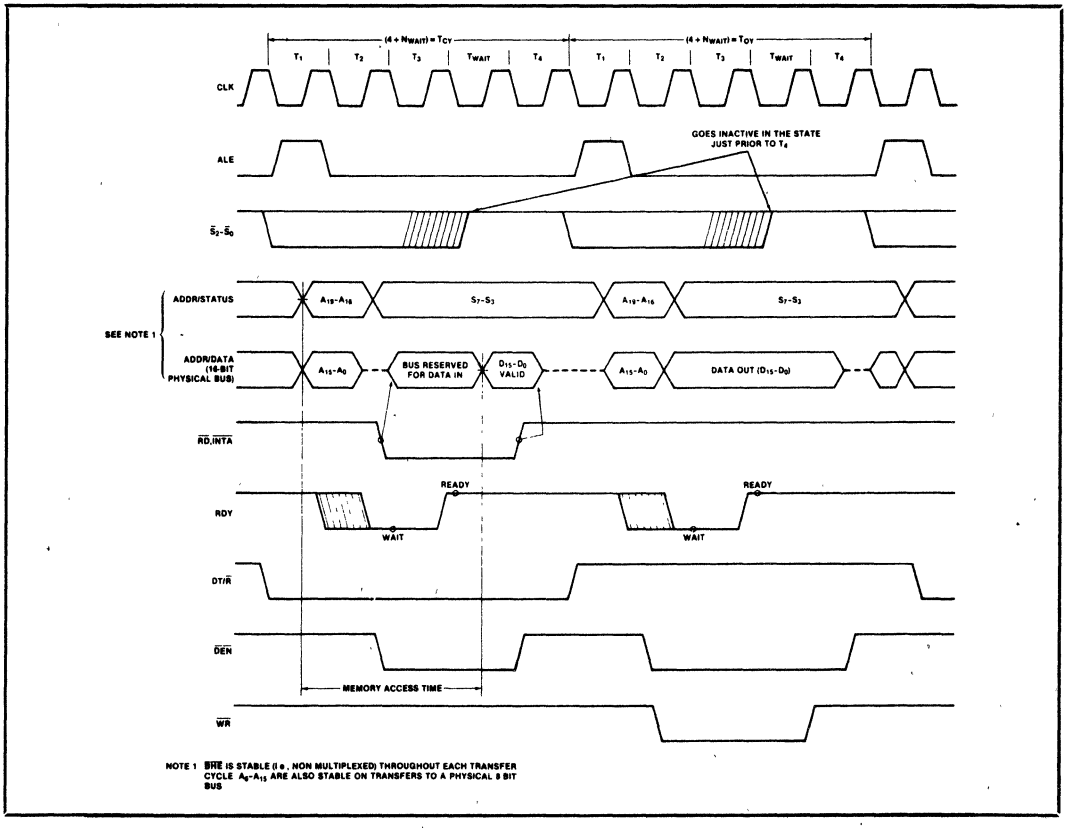


Figure 7. 8089 Bus Operation

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to Ground . . . . . −1.0 to +7V
Power Dissipation . . . . . 2.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+1.0	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL}=2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current		350	mA	T_A=25°C
I_{LI}	Input Leakage Current(1)		±10	μA	0V < VIN < V_{CC}
I_{LO}	Output Leakage Current		±10	μA	0.45V ≤ V_{OUT} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC}+1.0	V	
C_{IN}	Capacitance of Input Buffer (All input except AD_0−AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD_0−AD_{15}, $\overline{RQ}/\overline{GT}$)		15	pF	fc = 1 MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

8089/8086 MAX MODE SYSTEM (USING 8288 BUS CONTROLLER) TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	ns	
TCLCH	CLK Low Time	(⅔TCLCL)−15		ns	
TCHCL	CLK High Time	(⅓TCLCL)+2		ns	
TCH1CH2	CLK Rise Time		10	ns	From 1.0V to 3.5V
TCL2CL1	CLK Fall Time		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		ns	
TCLDX	Data In Hold Time	10		ns	
TR1VCL	RDY Setup Time into 8284 (See Notes 1, 2)	35		ns	
TCLR1X	RDY Hold Time into 8284 (See Notes 1, 2)	0		ns	
TRYHCH	READY Setup Time into 8089	(⅔TCLCL)−15		ns	
TCHRYX	READY Hold Time into 8089	30		ns	
TRYLCL	READY Inactive to CLK (See Note 4)	−8		ns	
TINVCH	Setup Time Recognition (DRQ 1,2 RESET, Ext 1,2) (See Note 2)	30		ns	
TGVCH	$\overline{RQ}/\overline{GT}$ Setup Time	30		ns	
TCAHCAL	CA Width	95		ns	
TSLVCAL	SEL Setup Time	75		ns	
TCALSLX	SEL Hold Time	0		ns	
TCHGX	GT Hold Time into 8089	40		ns	
TILIH	Input Rise Time (Except CLK)		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12	ns	From 2.0V to 0.8V

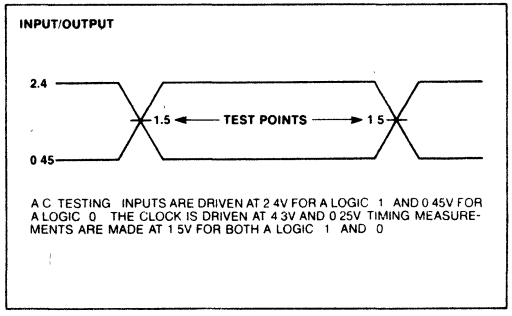
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TCLML	Command Active Delay (See Note 1)	10	35	ns	C_L = 80 pF
TCLMH	Command Inactive Delay (See Note 1)	10	35	ns	
TRYHSH	READY Active to Status Passive (See Note 3)		110	ns	
TCHSV	Status Active Delay	10	110	ns	
TCLSH	Status Inactive Delay	10	130	ns	
TCLAV	Address Valid Delay	10	110	ns	
TCLAX	Address Hold Time	10		ns	
TCLAZ	Address Float Delay	TCLAX	80	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15	ns	C_L = 150 pF
TCHLL	ALE Inactive Delay (See Note 1)		15	ns	
TCLDV	Data Valid Delay	10	110	ns	
TCHDX	Data Hold Time	10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30	ns	
TCLGL	$\overline{RQ}$ Active Delay	0	85	ns	C_L = 100 pF
TCLGH	$\overline{RQ}$ Inactive Delay		85	ns	Note 5: C_L = 30 pF
TCLSRV	SINTR Valid Delay		150	ns	C_L = 100 pF
TOLOH	Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12	ns	From 2.0V to 0.8V

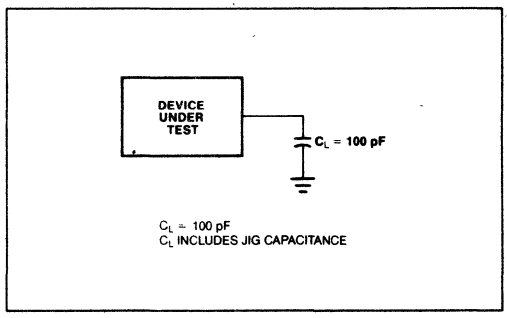
NOTES: 1 Signal at 8284 or 8288 shown for reference only
2 Setup requirement for asynchronous signal only to guarantee recognition at next CLK
3 Aplies only to T3 and TW states
4 Applies only to T2 state
5 Applies only if RQ/GT Mode 1 C_L=30pf, 2 7 KΩ pull up to Vcc

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 THE CLOCK IS DRIVEN AT 4 3V AND 0 25V TIMING MEASUREMENTS ARE MADE AT 1 5V FOR BOTH A LOGIC 1 AND 0

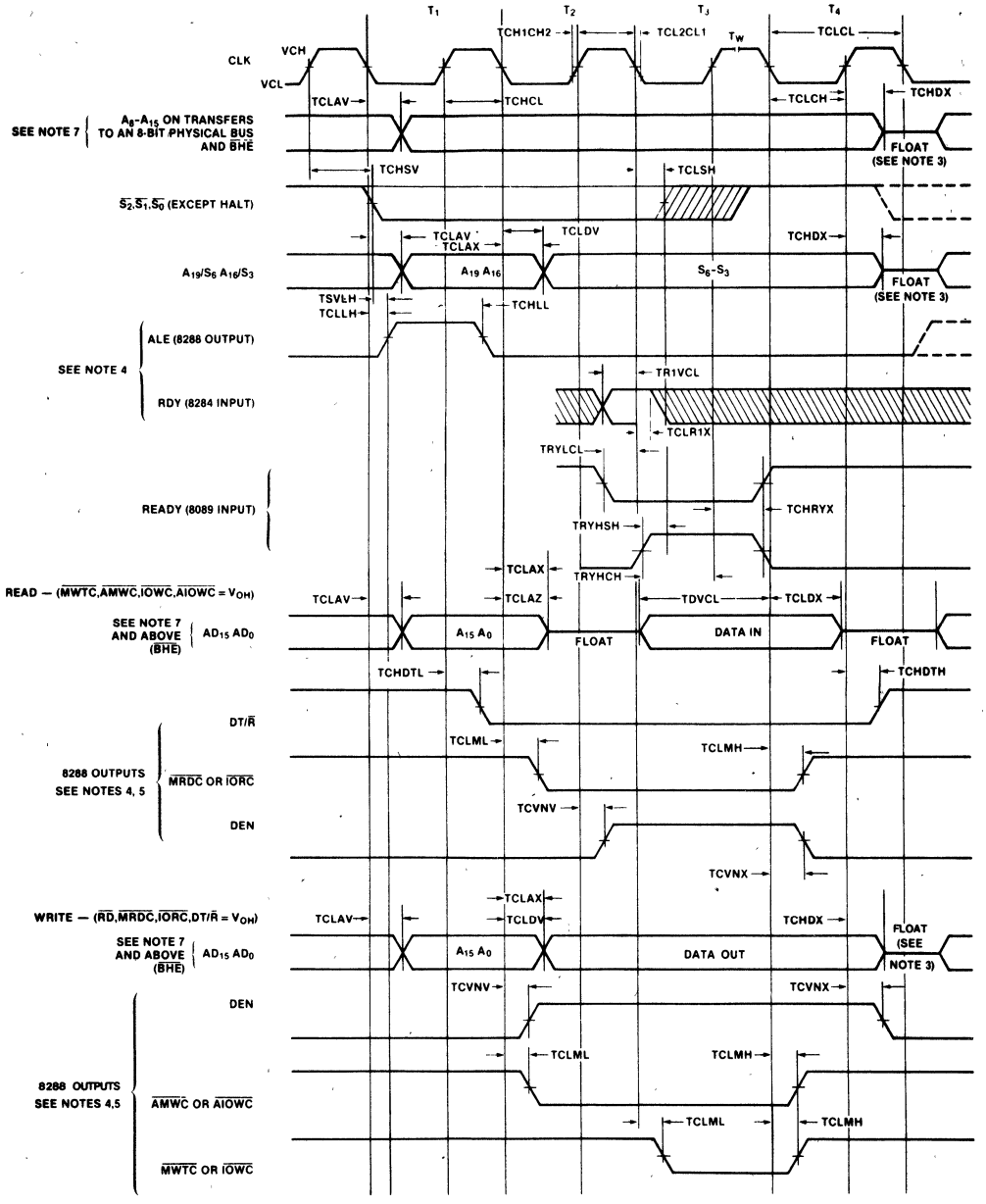
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

8089 BUS TIMING USING 8288

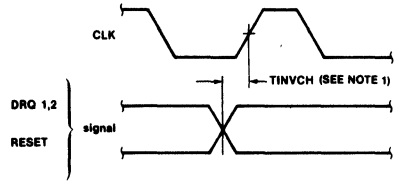


NOTES

1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED
2. RDY IS SAMPLED NEAR THE END OF T_2, T_3, T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED
3. FOLLOWING A WRITE CYCLE DATA REMAINS VALID ON THE 8089 LOCAL BUS UNTIL A LOCAL BUS MASTER DECIDES TO RUN ANOTHER BUS CYCLE. THE LOCAL BUS IS FLOATED BY THE 8089 WHEN THE 8089 ENTERS A REQUEST BUS ACKNOWLEDGE STATE
4. SIGNALS AT 8284 OR 8288 ARE SHOWN FOR REFERENCE ONLY
5. THE ISSUANCE OF THE 8288 COMMAND AND CONTROL SIGNALS (MRDC, MWTC, AMWC, IORC, IOWC, AIOWC, INTA AND DEN) LAGS THE ACTIVE HIGH 8288 CEN
6. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED
7. A_8-A_{15} ARE STABLE ON TRANSFERS TO AN 8 BIT PHYSICAL DATA BUS (i.e. A_8-A_{15} DON'T FLOAT ON A READ FROM AN 8 BIT PHYSICAL BUS OR MULTIPLEX WITH DATA ON A WRITE TO AN 8 BIT PHYSICAL BUS. $\overline{BHE}$ IS STABLE (NON MULTIPLEXED) FOR ALL TRANSFERS

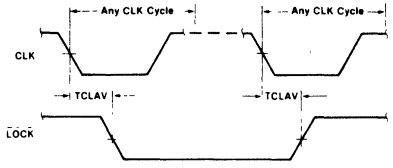
WAVEFORMS (Continued)

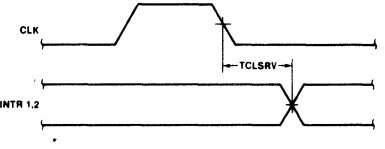
ASYNCHRONOUS SIGNAL RECOGNITION



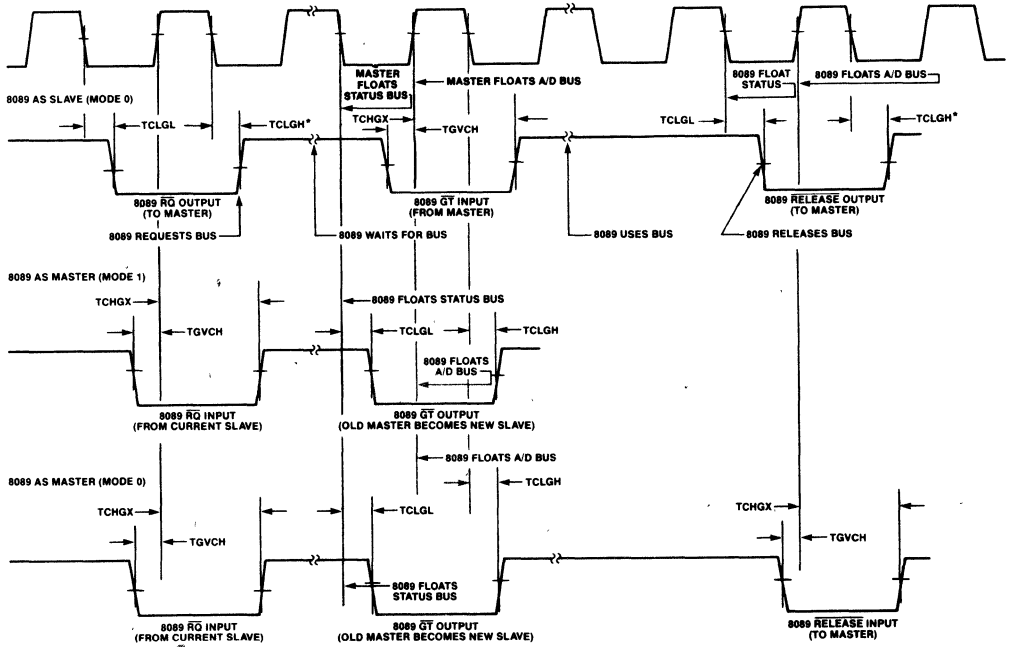
NOTES:
1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK
2 ALL INPUTS EXCEPT CA ARE LATCHED ON A CLK EDGE THE CA INPUT IS NEGATIVE EDGE TRIGGERED
3 DRQ BECOMING ACTIVE GREATER THAN 30 ns AFTER THE RISING EDGE OF CLK WILL GUARANTEE NON RECOGNITION UNTIL THE NEXT RISING CLOCK EDGE

BUS LOCK SIGNAL TIMING AND SINTR TIMING





REQUEST/GRANT SEQUENCE TIMINGS

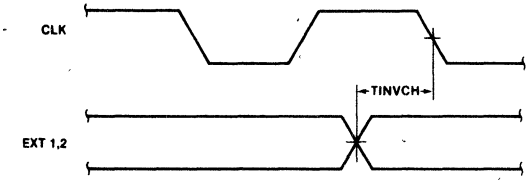


*CPU provides active pull-up

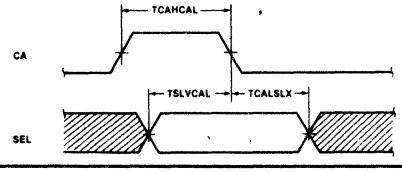
AFN-00840E

WAVEFORMS (Continued)

EXTERNAL TERMINATE SETUP TIMING



SEL SETUP AND TIMING



8089 INSTRUCTION SET SUMMARY

Data Transfers

	POINTER INSTRUCTIONS	OPCODE 7 0 7 0
LPD P,M	Load Pointer PPP from Addressed Location	P P P 0 0 A A 1 | 1 0 0 0 1 0 M M
LPDI P,I	Load Pointer PPP Immediate 4 Bytes	P P P 1 0 0 0 1 | 0 0 0 0 1 0 0 0
MOVP M,P	Store Contents of Pointer PPP in Addressed Location	P P P 0 0 A A 1 | 1 0 0 1 1 0 M M
MOVP P,M	Restore Pointer	P P P 0 0 A A 1 | 1 0 0 0 1 1 M M

	MOVE DATA	OPCODE
MOV M,M	Move from Source to Destination Source— Destination—	0 0 0 0 0 A A W | 1 0 0 1 0 0 M M 0 0 0 0 0 A A W | 1 1 0 0 1 1 M M
MOV R,M	Load Register RRR from Addressed Location	R R R 0 0 A A W | 1 0 0 0 0 0 M M
MOV M,R	Store Contents of Register RRR in Addressed Location	R R R 0 0 A A W | 1 0 0 0 0 1 M M
MOVI R	Load Register RRR Immediate (Byte) Sign Extend	R R R wb 0 0 W | 0 0 1 1 0 0 0 0
MOVI M	Move Immediate to Addressed Location	0 0 0 wb A A W | 0 1 0 0 1 1 M M

Control Transfer

	CALLS	OPCODE 7 0 7 0
*CALL	Call Unconditional	1 0 0 dd A A W | 1 0 0 1 1 1 M M

	JUMP	OPCODE
JMP	Unconditional	1 0 0 dd 0 0 W | 0 0 1 0 0 0 0 0
JZ M	Jump on Zero Memory	0 0 0 dd A A W | 1 1 1 0 0 1 M M
JZ R	Jump on Zero Register	R R R dd 0 0 0 | 0 1 0 0 0 1 0 0
JNZ M	Jump on Non-Zero Memory	0 0 0 dd A A W | 1 1 1 0 0 0 M M
JNZ R	Jump on Non-Zero Register	R R R dd 0 0 0 | 0 1 0 0 0 0 0 0
JBT	Test Bit and Jump if True	B B B dd A A 0 | 1 0 1 1 1 1 M M
JNBT	Test Bit and Jump if Not True	B B B dd A A 0 | 1 0 1 1 1 0 M M
JMCE	Mask/Compare and Jump on Equal	0 0 0 dd A A 0 | 1 0 1 1 0 0 M M
JMCNE	Mask/Compare and Jump on Non-Equal	0 0 0 dd A A 0 | 1 0 1 1 0 1 M M

Arithmetic and Logic Instructions

	INCREMENT, DECREMENT	OPCODE 7 0 7 0
INC M	Increment Addressed Location	0 0 0 0 0 A A W | 1 1 1 0 1 0 M M
INC R	Increment Register	R R R 0 0 0 0 0 | 0 0 1 1 1 0 0 0
DEC M	Decrement Addressed Location	0 0 0 0 0 A A W | 1 1 1 0 1 1 M M
DEC R	Decrement Register	R R R 0 0 0 0 0 | 0 0 1 1 1 1 0 0

Arithmetic and Logic Instructions

	ADD	OPCODE
		7 0 7 0
ADDI M,I	ADD Immediate to Memory	0 0 0 wb A A W 1 1 0 0 0 0 M M
ADDI R,I	ADD Immediate to Register	R R R wb 0 0 W 0 0 1 0 0 0 0 0
ADD M,R	ADD Register to Memory	R R R 0 0 A A W 1 1 0 1 0 0 M M
ADD R,M	ADD Memory to Register	R R R 0 0 A A W 1 0 1 0 0 0 M M
	AND	**OPCODE**
ANDI M,I	AND Memory with Immediate	0 0 0 wb A A W 1 1 0 0 1 0 M M
ANDI R,I	AND Register with Immediate	R R R wb 0 0 W 0 0 1 0 1 0 0 0
AND M,R	AND Memory with Register	R R R 0 0 A A W 1 1 0 1 1 0 M M
AND R,M	AND Register with Memory	R R R 0 0 A A W 1 0 1 0 1 0 M M
	OR	**OPCODE**
ORI M,I	OR Memory with Immediate	0 0 0 wb A A W 1 1 0 0 0 1 M M
ORI R,I	OR Register with Immediate	R R R wb A A W 0 0 1 0 0 1 0 0
OR M,R	OR Memory with Register	R R R 0 0 A A W 1 1 0 1 0 1 M M
OR R,M	OR Register with Memory	R R R 0 0 A A W 1 0 1 0 0 1 M M
	NOT	**OPCODE**
NOT R	Complement Register	R R R 0 0 0 0 0 0 0 1 0 1 1 0 0
NOT M	Complement Memory	0 0 0 0 0 A A W 1 1 0 1 1 1 M M
NOT R,M	Complement Memory, Place in Register	R R R 0 0 A A W 1 0 1 0 1 1 M M

Bit Manipulation and Test Instructions

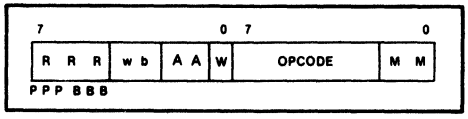
	BIT MANIPULATION	OPCODE
		7 0 7 0
SET	Set the Selected Bit	B B B 0 0 A A 0 1 1 1 1 0 1 M M
CLR	Clear the Selected Bit	B B B 0 0 A A 0 1 1 1 1 1 0 M M
	TEST	**OPCODE**
TSL	Test and Set Lock	0 0 0 1 1 A A 0 1 0 0 1 0 1 M M

Control

	Control	OPCODE
		7 0 7 0
HLT	Halt Channel Execution	0 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0
SINTR	Set Interrupt Service Flip Flop	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
NOP	No Operation	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
XFER	Enter DMA Transfer	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
WID	Set Source, Destination Bus Width; S,D 0 = 8, 1 = 16	1 S D 0 0 0 0 0 0 0 0 0 0 0 0 0

*AAField in call instruction can be 00, 01, 10 only.
**OPCODE is second byte fetched.

All instructions consist of at least 2 bytes, while some instructions may use up to 3 additional bytes to specify literals and displacement data. The definition of the various fields within each instruction is given below:



MM	Base Pointer Select
00	GA
01	GB
10	GC
11	PP

RRR Register Field

The RRR field specifies a 16-bit register to be used in the instruction. If GA, GB, GC or TP, are referenced by the RRR field, the upper 4 bits of the registers are loaded with the sign bit (Bit 15). PPP registers are used as 20-bit address pointers.

RRR			
000	r0	GA	
001	r1	GB	
010	r2	GC	
011	r3	BC	; byte count
100	r4	TP	; task block
101	r5	IX	; index register
110	r6	CC	; channel control (mode)
111	r7	MC	; mask/compare

See Notes 1, 2

PPP			
000	p0	GA	;
001	p1	GB	;
010	p2	GC	;
100	p4	TP	; task block pointer

Note 1. Logical and arithmetic instructions should not be used to update the CC register (i.e.—only MOV and MOVI instructions should be used.)
2. A 20-bit register (GA, GB, GC or TP) that is initialized as a 16-bit I/O space pointer must be saved at even addresses when using MOVP or CALL instructions.

NOTES:

BBB Bit Select Field

The bit select field replaces the RRR field in bit manipulation instructions and is used to select a bit to be operated on by those instructions. Bit 0 is the least significant bit.

wb

01 1 byte literal
10 2 byte (word) literal

dd

01 1 byte displacement
10 2 byte (word) displacement.

AA Field

00 The selected pointer contains the operand address.
01 The operand address is formed by adding an 8-bit, unsigned, offset contained in the instruction to the selected pointer. The contents of the pointer are unchanged.
10 The operand address is formed by adding the contents of the Index register to the selected pointer. Both registers remain unchanged.
11 Same as 10 except the Index register is post auto-incremented (by 1 for 8-bit transfer, by 2 for 16-bit transfer).

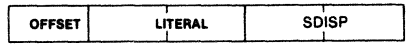
W Width Field

0 The selected operand is 1 byte long.
1 The selected operand is 2 bytes long.

Additional Bytes

OFFSET : 8-bit unsigned offset.
SDISP : 8/16-bit signed displacement.
LITERAL : 8/16-bit literal. (32 bits for LDPI).

The order in which the above optional bytes appear in IOP instructions is given below:



Offsets are treated as unsigned numbers. Literals and displacements are sign extended (2's complement).

8087
NUMERIC DATA COPROCESSOR

- **High Performance Numeric Data Coprocessor**
- **Adds Arithmetic, Trigonometric, Exponential, and Logarithmic Instructions to the Standard iAPX 86 and iAPX 186 Instruction Set For All Data Types**
- **All 24 Addressing Modes Available with 8086, 8088/80186, 80188 CPUs.**
- **Conforms To Proposed IEEE Floating Point Standard**
- **CPU/8087 System Supports 8 Data Types: 8-, 16-, 32-, 64-Bit Integers, 32-, 64-, 80-Bit Floating Point, and 18-Digit BCD Operands**
- **Adds 8 x 80-Bit Individually Addressable Register Stack**
- **7 Built-in Exception Handling Functions**
- **MULTIBUS System Compatible Interface**

The 8087 Numeric Data Coprocessor provides the instructions and data types needed for high performance numeric applications, providing up to 100 times the performance of a CPU alone. The 8087 is implemented in N-channel, depletion load, silicon gate technology (HMOS), housed in a 40-pin package. Sixty-eight numeric processing instructions are added to the iAPX 86, 186 instruction sets, and eight 80-bit registers are added to the register set. The 8087 conforms to the proposed IEEE Floating Point Standard.

The two-chip numeric data processing systems are refered to as follows;

iAPX 86/20—16-bit 8086 CPU with 8087
iAPX 88/20—8-bit 8088 CPU with 8087
iAPX 186—16-bit 80186 CPU with 8087
iAPX 188—8-bit 80188 CPU with 8087

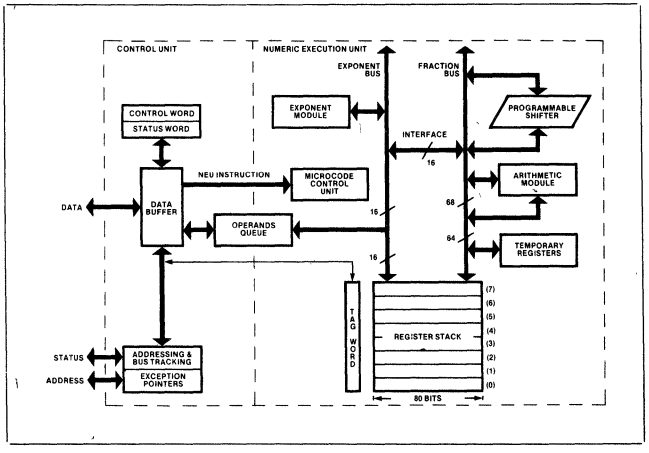


Figure 1. 8087 Block Diagram

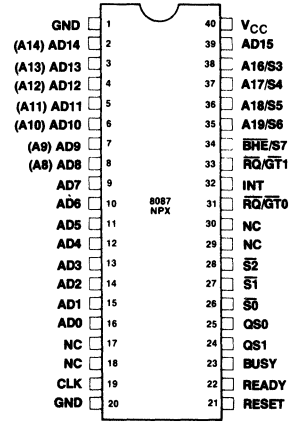


Figure 2. 8087 Pin Configuration

Table 1. 8087 Pin Description

Symbol	Type	Name and Function
AD15-AD0	I/O	**Address Data:** These lines constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. A0 is analogous to $\overline{BHE}$ for the lower byte of the data bus, pins D7-D0. It is LOW during T_1 when a byte is to be transterred on the lower portion of the bus in memory operations. Eight-bit oriented devices tied to the lower half of the bus would normally use A0 to condition chip select functions. These lines are active HIGH. They are input/output lines for 8087-driven bus cycles and are inputs which the 8087 monitors when the CPU is in control of the bus. A15-A8 do not require an address latch in an iAPX 88/20. The 8087 will supply an address for the T_1-T_4 period.
A19/S6, A18/S5, A17/S4, A16/S3	I/O	**Address Memory:** During T_1 these are the four most significant address lines for memory operations. During memory operations, status information is available on these lines during T_2, T_3, T_W, and T_4. For 8087 controlled bus cycles, S6, S4, and S3 are reserved and currently one (HIGH), while S5 is always LOW. These lines are inputs which the 8087 monitors when the CPU is in control of the bus.
BHE/S7	I/O	**Bus High Enable:** During T_1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D15-D8. Eight-bit-oriented devices tied to the upper half of the bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is LOW during T_1 for read and write cycles when a byte is to be transferred on the high portion of the bus. The S7 status information is available during T_2, T_3, T_W, and T_4. The signal is active LOW. S7 is an input which the 8087 monitors during the CPU-controlled bus cycles.
$\overline{S2}$, $\overline{S1}$, $\overline{S0}$	I/O	**Status:** For 8087-driven bus cycles, these status lines are encoded as follows: $\overline{S2}$ $\overline{S1}$ $\overline{S0}$ 0 (LOW) X X Unused 1 (HIGH) 0 0 Unused 1 0 1 Read Memory 1 1 0 Write Memory 1 1 1 Passive Status is driven active during T_4, remains valid during T_1 and T_2, and is returned to the passive state (1, 1, 1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller (or the 82188 Advanced Bus Controller with a 186/188 CPU) to generate all memory access control signals. Any change in $\overline{S2}$, $\overline{S1}$, or $\overline{S0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle. These signals are monitored by the 8087 when the CPU is in control of the bus.
$\overline{RQ}/\overline{GT0}$	I/O	**Request/Grant:** This request/grant pin is used by the 8087 to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins. The request grant sequence on this pin is as follows: 1. A pulse one clock wide is passed to the CPU to indicate a local bus request by either the 8087 or the master connected to the 8087 $\overline{RQ}/\overline{GT1}$ pin. 2. The 8087 waits for the grant pulse and when it is received will either initiate bus transfer activity in the clock cycle following the grant or pass the grant out on the $\overline{RQ}/\overline{GT1}$ pin in this clock if the initial request was for another bus master. 3. The 8087 will generate a release pulse to the CPU one clock cycle after the completion of the last 8087 bus cycle or on receipt of the release pulse from the bus master on $\overline{RQ}/\overline{GT1}$. For iAPX 186 systems, the same sequence applies except $\overline{RQ}/\overline{GT}$ signals are converted to appropriate HOLD, HLDA signals by the 82188 Advanced Bus Controller. This is to conform with iAPX 186's HOLD, HLDA bus exchange protocol.

Table 1. 8087 Pin Description (Continued)

Symbol	Type	Name and Function
$\overline{RQ}/\overline{GT}1$	I/O	**Request/Grant:** This request/grant pin is used by another local bus master to force the 8087 to request the local bus. If the 8087 is not in control of the bus when the request is made the request/grant sequence is passed through the 8087 on the $\overline{RQ}/\overline{GT}0$ pin one cycle later. Subsequent grant and release pulses are also passed through the 8087 with a two and one clock delay, respectively, for resynchronization. $\overline{RQ}/\overline{GT}1$ has an internal pullup resistor, and so may be left unconnected. If the 8087 has control of the bus the request/grant sequence is as follows: 1. A pulse 1 CLK wide from another local bus master indicates a local bus request to the 8087 (pulse 1). 2. During the 8087's next T_4 or T_1 a pulse 1 CLK wide from the 8087 to the requesting master (pulse 2) indicates that the 8087 has allowed the local bus to float and that it will enter the "RQ/GT acknowledge" state at the next CLK. The 8087's control unit is disconnected logically from the local bus during "RQ/GT acknowledge." 3. A pulse 1 CLK wide from the requesting master indicates to the 8087 (pulse 3) that the "RQ/GT" request is about to end and that the 8087 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW. For iAPX 186 systems, the $\overline{RQ}/\overline{GT}1$ line may be connected to the 82188 Advanced Bus Controller. In this case, a third processor with a HOLD, HLDA bus exchange system may acquire the bus from the 8087. For this configuration, $\overline{RQ}/\overline{GT}1$ will only be used if the 8087 is the bus master.
QS1, QS0	I	**QS1, QS0:** QS1 and QS0 provide the 8087 with status to allow tracking of the CPU instruction queue. **QS1** **QS0** 0 (LOW) 0 No Operation 0 1 First Byte of Op Code from Queue 1 (HIGH) 0 Empty the Queue 1 1 Subsequent Byte from Queue
INT	O	**Interrupt:** This line is used to indicate that an unmasked exception has occurred during numeric instruction execution when 8087 interrupts are enabled. This signal is typically routed to an 8259A for 8086 systems and to INTO for iAPX 186 systems. INT is active HIGH.
BUSY	O	**Busy:** This signal indicates that the 8087 NEU is executing a numeric instruction. It is connected to the CPU's $\overline{TEST}$ pin to provide synchronization. In the case of an unmasked exception BUSY remains active until the exception is cleared. BUSY is active HIGH.
READY	I	**Ready:** READY is the acknowledgment from the addressed memory device that it will complete the data transfer. The RDY signal from memory is synchronized by the 8284A Clock Generator to form READY for 8086 systems. For iAPX 186 systems, RDY is synchronized by the 82188 Advanced Bus Controller to form READY. This signal is active HIGH.
RESET	I	**Reset:** RESET causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. RESET is internally synchronized.
CLK	I	**Clock:** The clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.
V_{CC}		**Power:** V_{CC} is the +5V power supply pin.
GND		**Ground:** GND are the ground pins.

NOTE:
For the pin descriptions of the 8086, 8088, 80186 and 80188 CPU's, reference the respective data sheets (iAPX 86/10, iAPX 88/10, iAPX 186, iAPX 188).

APPLICATION AREAS

The 8087 provides functions meant specifically for high performance numeric processing requirements. Trigonometric, logarithmic, and exponential functions are built into the coprocessor hardware. These functions are essential in scientific, engineering, navigational, or military applications.

The 8087 also has capabilities meant for business or commercial computing. An 8087 can process Binary Coded Decimal (BCD) numbers up to 18 digits without roundoff errors. It can also perform arithmetic on integers as large as 64 bits $\pm 10^{18}$).

PROGRAMMING LANGUAGE SUPPORT

Programs for the 8087 can be written in Intel's high-level languages for iAPX 86, 88 and iAPX 186, 188 Systems; ASM-86 (the iAPX 86, 88 assembly language), PL/M-86, FORTRAN-86, and PASCAL-86.

RELATED INFORMATION

For iAPX 86/10, iAPX 88/10, iAPX 186 or iAPX 188 details, refer to the respective data sheets.

FUNCTIONAL DESCRIPTION

The 8087 Numeric Data Processor's architecture is designed for high performance numeric computing in conjunction with general purpose processing.

The 8087 is a numeric processor extension that provides arithmetic and logical instruction support for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 8087 executes instructions as a coprocessor to a maximum mode CPU. It effectively extends the register and instruction set of the system and adds several new data types as well. Figure 3 presents the registers of the CPU-8087. Table 2 shows the range of data types supported by the 8087. The 8087 is treated as an extension to the CPU, providing register, data types, control, and instruction capabilities at the hardware level. At the programmers level the CPU and the 8087 are viewed as a single unified processor.

System Configuration

As a coprocessor to an 8086 or 8088, the 8087 is wired in parallel with the CPU as shown in Figure 4. Figure 5 shows the iAPX 186 system configuration. The CPU's status (S0-S2) and queue status lines (QS0-QS1) enable the 8087 to monitor and decode instructions in synchronization with the CPU and without any CPU overhead. For iAPX 186 systems, the queue status signals of the iAPX 186 are synchronized to 8087 requirements by the 82188 Advanced Bus Controller. Once started, the 8087 can process in parallel with, and independent of, the host CPU. For resynchronization, the 8087's BUSY signal informs the CPU that the 8087 is executing an instruction and the CPU WAIT instruction tests this signal to insure that the 8087 is ready to execute subsequent instructions. The 8087

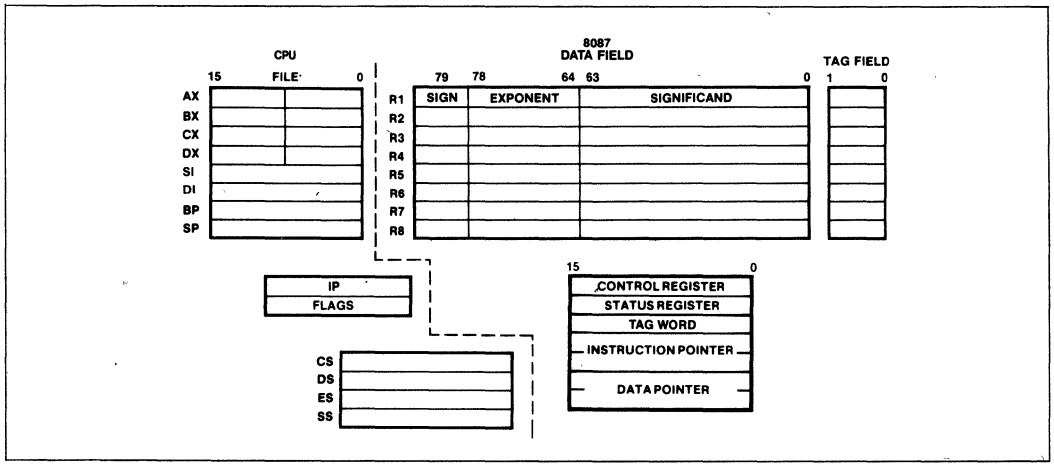


Figure 3. CPU-8087 Architecture

can interrupt the CPU when it detects an error or exception. The 8087's interrupt request line is typically routed to the CPU through an 8259A Programmable Interrupt Controller for 8086, 8088 systems and INTO for iAPX, 188.

The 8087 uses one of the request/grant lines of the iAPX 86, 88 architecture (typically $\overline{RQ}/\overline{GT}1$)to obtain control of the local bus for data transfers. The other request/grant line is available for general system use (for instance by an I/O processor in LOCAL mode). A bus master can also be connected to the 8087's $\overline{RQ}/\overline{GT}1$ line. In this configuration the 8087 will pass the request/grant handshake signals between the CPU and the attached master when the 8087 is not in control of the bus and will relinquish the bus to the master directly when the 8087 is in control. In this way two additional masters can be configured in an iAPX 86, 88/20 system; one will share the 8086 bus with the 8087 on a first come first served basis, and the second will be guaranteed to be higher in priority than the 8087.

For iAPX 186 systems, $\overline{RQ}/\overline{GT}0$ and $\overline{RQ}/\overline{GT}1$ are connected to the corresponding inputs of the 82188 Advanced Bus Controller. Because the iAPX has a HOLD, HLDA bus exchange protocol, an interface is needed which will translate $\overline{RQ}/\overline{GT}$ signals to corresponding HOLD, HDLA signals and visa versa. One of the funtions of the 82188 ABC is to provide this translation. $\overline{RQ}/\overline{GT}0$ is translated to HOLD, HLDA signals which are then directly connected to the iAPX 186. The $\overline{RQ}/\overline{GT}1$ line is also translated into HOLD, HLDA signals (referred to as SYSHOLD, SYSHLDA signals) by the 82188 ABC. This allows a third processor (using a HOLD, HLDA bus exchange protocol) to gain control of the bus. Unlike an iAPX 86/20 system, $\overline{RQ}/\overline{GT}1$ is only used when the 8087 has bus control. If the third processor requests the bus when the current bus master is the 186, the 82188 ABC will directly pass the request onto the 186 without going through the 8087. The third processor has the highest bus priority in the system. If the 8087 requests the bus while the third processor has bus control, the grant pulse will not be issued until the third processor releases the bus (using SYSHOLD). In this configuration, the third processor has the highest priority, the 8087 has the next highest, and the 186 has the lowest bus priority.

Table 2. 8087 Data Types

Data Formats	Range	Precision	Most Significant Byte (7 0 7 0 7 0 7 0 7 0 7 0 7 0 7 0)
Byte Integer	10^2	8 Bits	I_7 I_0 Two's Complement
Word Integer	10^4	16 Bits	I_{15} I_0 Two's Complement
Short Integer	10^9	32 Bits	I_{31} I_0 Two's Complement
Long Integer	10^{18}	64 Bits	I_{63} I_0 Two's Complement
Packed BCD	10^{18}	18 Digits	S — $D_{17}D_{16}$ D_1 D_0
Short Real	$10^{\pm 38}$	24 Bits	S E_7 E_0 F_1 F_{23} F_0 Implicit
Long Real	$10^{\pm 308}$	53 Bits	S E_{10} E_0 F_1 F_{52} F_0 Implicit
Temporary Real	$10^{\pm 4932}$	64 Bits	S E_{14} E_0 F_0 F_{63}

Integer: I

Packed BCD: $(-1)^S(D_{17}\ldots D_0)$

Real: $(-1)^S(2^{E-BIAS})(F_0 \cdot F_1 \ldots)$

Bias = 127 for Short Real
1023 for Long Real
16383 for Temp Real

Bus Operation

The 8087 bus structure, operation and timing are identical to all other processors in the iAPX 86, 88 series (maximum mode configuration). The address is time multiplexed with the data on the first 16/8 lines of the address/data bus. A16 through A19 are time multiplexed with four status lines S3-S6. S3, S4 and S6 are always one (HIGH) for 8087-driven bus cycles while S5 is always zero (LOW). When the 8087 is monitoring CPU bus cycles (passive mode) S6 is also monitored by the 8087 to differentiate 8086/8088 activity from that of a local I/O processor or any other local bus master. (The 8086/8088 must be the only processor on the local bus to drive S6 LOW.) S7 is multiplexed with and has the same value as $\overline{BHE}$ for all 8087 bus cycles.

The first three status lines, $\overline{S0}$-$\overline{S2}$, are used with an 8288 bus controller or 82188 Advanced Bus Controller to determine the type of bus cycle being run:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	
0	X	X	Unused
1	0	0	Unused
1	0	1	Memory Data Read
1	1	0	Memory Data Write
1	1	1	Passive (no bus cycle)

Programming Interface

The 8087 includes the standard iAPX 86/10, 88/10 instruction set for general data manipulation and program control. It also includes 68 numeric instructions for extended precision integer, floating point, trigonometric, logarithmic, and exponential functions. Sample execution times for several 8087 functions are shown in Table 3. Overall system performance is 100 times that of an iAPX 86/10 class processor for numeric instructions.

Any instruction executed by the 8087 is the combined result of the CPU and 8087 activity. The CPU and the 8087 have specialized functions and registers providing fast concurrent operation. The CPU controls overall program execution while the 8087 uses the coprocessor interface to recognize and perform numeric operations.

Table 2 lists the eight data types the 8087 supports and presents the format for each type. Internally, the 8087 holds all numbers in the temporary real format. Load and store instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point numbers or 18-digit packed BCD numbers into temporary real format and vice versa. The 8087 also provides the capability to control round off, underflow, and overflow errors in each calculation.

Computations in the 8087 use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. The 8087 register set can be accessed as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 5 lists the 8087's instructions by class. All appear as ESCAPE instructions to the host. Assembly language programs are written in ASM-86, the iAPX 86, 88 assembly language.

Table 3. Execution Times for Selected iAPX 86/20 Numeric Instructions and Corresponding iAPX 86/10 Emulation

Floating Point Instruction	Approximate Execution Time (μs)	
	iAPX 86/20 (5 MHz Clock)	iAPX 86/10 Emulation
Add/Subtract	17	1,600
Multiply (single precision)	19	1,600
Multiply (extended precision)	27	2,100
Divide	39	3,200
Compare	9	1,300
Load (double precision)	10	1,700
Store (double precision)	21	1,200
Square Root	36	19,600
Tangent	90	13,000
Exponentiation	100	17,100

NUMERIC PROCESSOR EXTENSION ARCHITECTURE

As Shown in Figure 5, the 8087 is internally divided into two processing elements, the control unit (CU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the CU receives and decodes instructions, reads and writes memory operands and executes 8087 control instructions. The two elements are able to operate independently of one another, allowing the CU to maintain synchronization with the CPU while the NEU is busy processing a numeric instruction.

Control Unit

The CU keeps the 8087 operating in synchronization with its host CPU. 8087 instructions are intermixed with CPU instructions in a single instruction stream. The CPU fetches all instructions from memory; by monitoring the status ($\overline{S0}$-$\overline{S2}$, S6) emitted by the CPU, the control unit determines when an instruction is being fetched. The

Figure 4. iAPX 86/20, 88/20 System Configuration

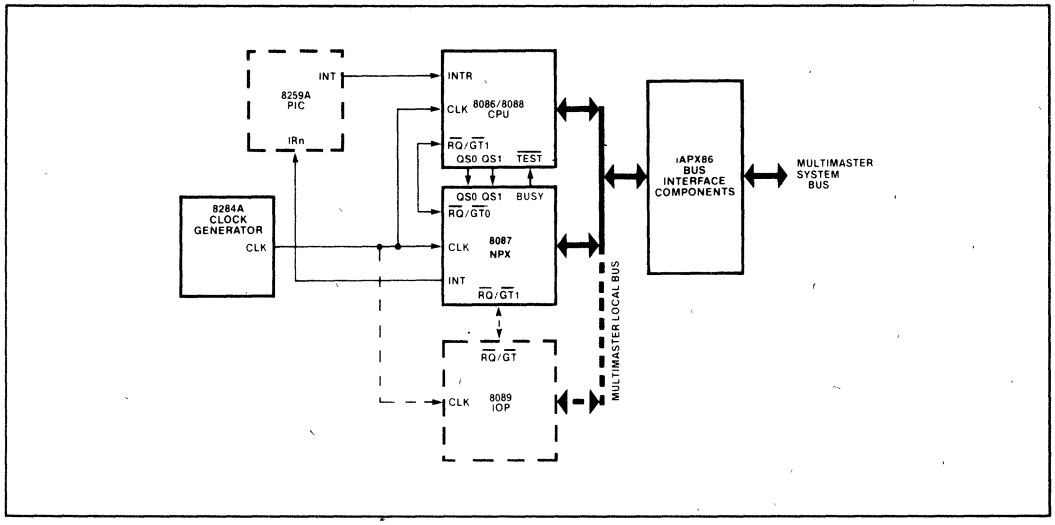
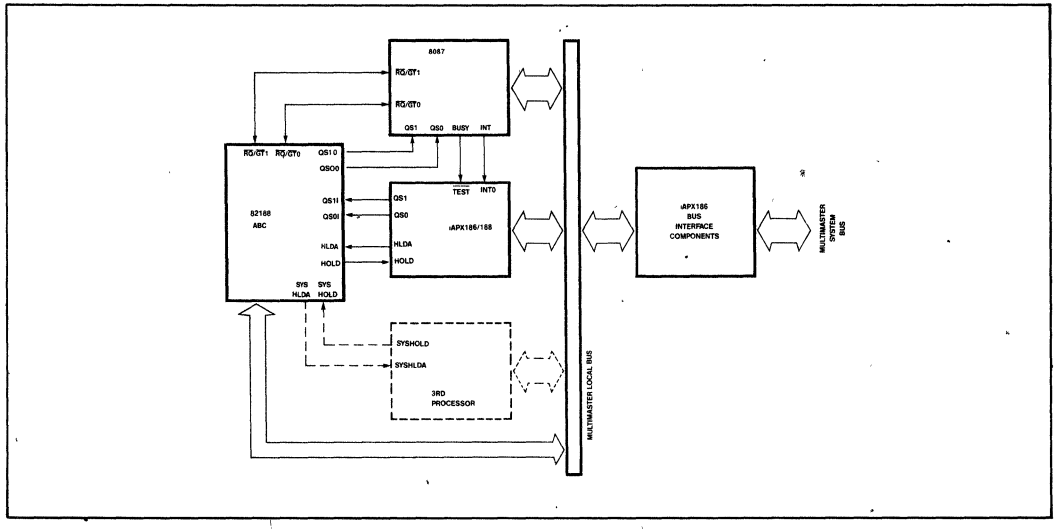


Figure 5. iAPX 186, 188 System Configuration



CU monitors the Data bus in parallel with the CPU to obtain instructions that pertain to the 8087.

The CU maintains an instruction queue that is identical to the queue in the host CPU. The CU automatically determines if the CPU is an 8086/186 or an 8088/188 immediately after reset (by monitoring the $\overline{BHE}$/S7 line) and matches its queue length accordingly. By monitoring the CPU's queue status lines (QS0, QS1), the CU obtains and decodes instructions from the queue in synchronization with the CPU.

A numeric instruction appears as an ESCAPE instruction to the CPU. Both the CPU and 8087 decode and execute the ESCAPE instruction together. The 8087 only recognizes the numeric instructions shown in Table 5. The start of a numeric operation is acomplished when the CPU executes the ESCAPE instruction. The instruction may or may not identify a memory operand.

The CPU does, however, distinguish between ESC instructions that reference memory and those that do not. If the instruction refers to a memory operand, the CPU calculates the operand's address using any one of its available addressing modes, and then performs a "dummy read" of the word at that location. (Any location within the 1M byte address space is allowed.) This is a normal read cycle except that the CPU ignores the data it receives. If the ESC instruction does not contain a memory reference (e.g. an 8087 stack operation), the CPU simply proceeds to the next instruction.

An 8087 Instruction can have one of three memory reference options; (1) not reference memory; (2) load an operand word from memory into the 8087; or (3) store an operand word from the 8087 into memory. If no memory reference is required, the 8087 simply executes its instruction. If a memory reference is required, the CU uses a "dummy read" cycle initiated by the CPU to capture and save the address that the CPU places on the bus. If the instruction is a load, the CU additionally captures the data word when it becomes available on the local data bus. If data required is longer than one word, the CU immediately obtains the bus from the CPU using the request/grant protocol and reads the rest of the information in consecutive bus cycles. In a store operation, the CU captures and saves the store address as in a load, and ignores the data word that follows in the "dummy read" cycle. When the 8087 is ready to perform the store, the CU obtains the bus from the CPU and writes the operand starting at the specified address.

Numeric Execution Unit

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 fraction bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the 8087 BUSY signal. This signal can be used in conjunction with the CPU WAIT instruction to resynchronize both processors when the NEU has completed its current instruction.

Register Set

The iAPX 86/20 register set is shown in Figure 3. Each of the eight data registers in the 8087's register stack is 80 bits and is divided into "fields" corresponding to the 8087's temporary real data type.

At a given point in time the TOP field in the control word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like CPU stacks in memory, the 8087 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the top of the stack. These instructions implicitly address the register pointed to by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. Explicit register addressing is "top-relative."

Status Word

The status word shown in Figure 6 reflects the overall state of the 8087; it may be stored in memory and then inspected by CPU code. The status word is a 16-bit register divided into fields as shown in Figure 6. The busy bit (bit 15) indicates whether the NEU is either executing an instruction or has an interrupt request pending (B = 1), or is idle (B = 0). Several instructions which store and manipulate the status word are executed exclusively by the CU, and these do not set the busy bit themselves.

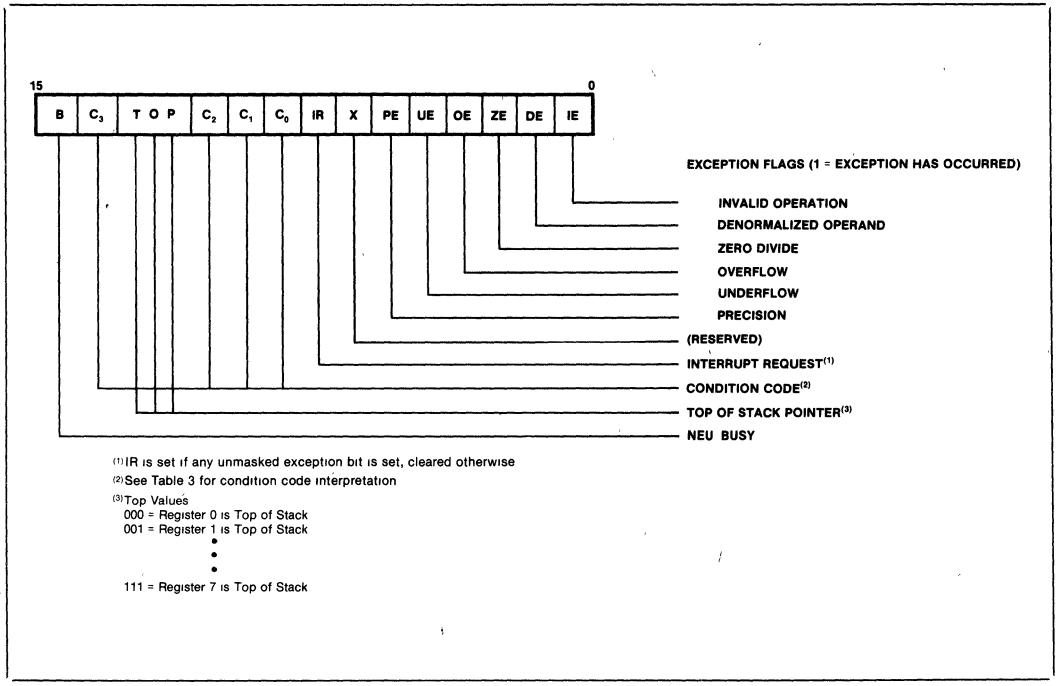


Figure 6. 8087 Status Word

The four numeric condition code bits (C_0-C_3) are similar to flags in a CPU: various instructions update these bits to reflect the outcome of 8087 operations. The effect of these instructions on the condition code bits is summarized in Table 4.

Bits 14–12 of the status word point to the 8087 register that is the current top-of-stack (TOP) as described above.

Bit 7 is the interrupt request bit. This bit is set if any unmasked exception bit is set and cleared otherwise.

Bits 5–0 are set to indicate that the NEU has detected an exception while executing an instruction.

Tag Word

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the 8087's performance. The tag word can be used, however, to interpret the contents of 8087 registers.

Instruction and Data Pointers

The instruction and data pointers (see Figure 8) are provided for user-written error handlers. Whenever the 8087 executes an NEU instruction, the CU saves the instruction address, the operand address (if present) and the instruction opcode. 8087 instructions can store this data into memory.

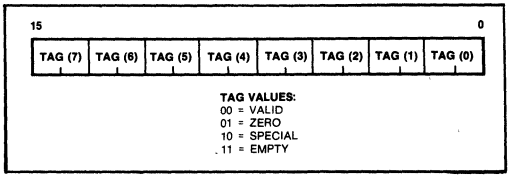


Figure 7. 8087 Tag Word

Table 4a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 4b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent =0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent =0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 4b. Condition Code Interpretation after FPREM Instruction As a Function of Dividend Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3[1]	C_1[1]	Q_0
Dividend < 4 * Modulus	C_3[1]	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

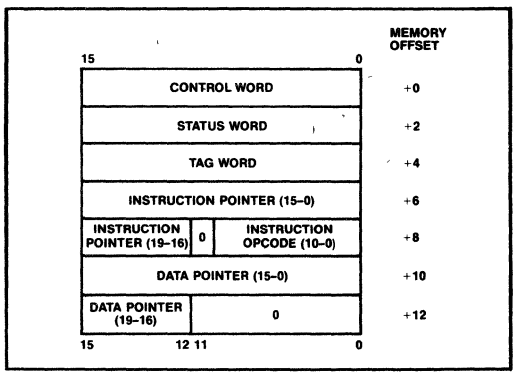


Figure 8. 8087 Instruction and Data Pointer Image in Memory

Control Word

The 8087 provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of the fields in the control word.

The low order byte of this control word configures 8087 interrupts and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 8087 recognizes and bit 7 contains a general mask bit for all 8087 interrupts. The high order byte of the control word configures the 8087 operating mode including precision, rounding, and infinity controls. The precision control bits (bits 9–8) can be used to set the 8087 internal operating precision at less than the default of temporary real precision. This can be useful in providing compatibility with earlier generation arithmetic processors of smaller precision than the 8087. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest mode specified in the proposed IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure, $\pm\infty$, or projective closure, ∞, is treated as unsigned, may be specified).

Exception Handling

The 8087 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause an interrupt if unmasked and interrupts are enabled.

If interrupts are disabled the 8087 will simply continue execution regardless of whether the host clears the exception. If a specific exception class is masked and that exception occurs, however, the 8087 will post the exception in the status register and perform an on-chip default exception handling procedure, thereby allowing processing to continue. The exceptions that the 8087 detects are the following:

1. INVALID OPERATION: Stack overflow, stack underflow, indeterminate form (0/0, $\infty - \infty$, etc.) or the use of a Non-Number (NAN) as an operand. An exponent value is reserved and any bit pattern with this value in the exponent field is termed a Non-Number and causes this exception. If this exception is masked, the 8087's default response is to generate a specific NAN called INDEFINITE, or to propagate already existing NANs as the calculation result.

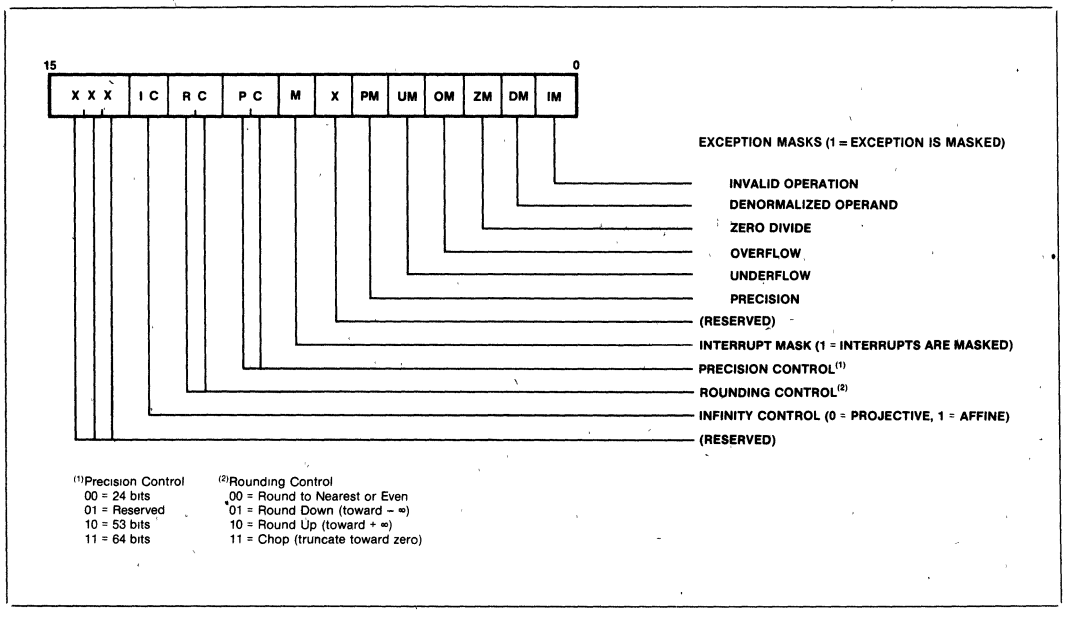


Figure 9. 8087 Control Word

2. OVERFLOW: The result is too large in magnitude to fit the specified format. The 8087 will generate an encoding for infinity if this exception is masked.

3. ZERO DIVISOR: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 8087 will generate an encoding for infinity if this exception is masked.

4. UNDERFLOW: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 8087 will denormalize (shift right) the fraction until the exponent is in range. This process is called gradual underflow.

5. DENORMALIZED OPERAND: At least one of the operands or the result is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

6. INEXACT RESULT: If the true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground .....−1.0V to +7V
Power Dissipation .....3.0 Watt

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±5%)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400 μA
I_{CC}	Power Supply Current		475	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.9	V_{CC} + 1.0	V	
C_{IN}	Capacitance of Inputs		10	pF	fc = 1 MHz
C_{IO}	Capacitance of I/O Buffer (AD0-15, A_{16}–A_{19}, BHE, S2-S0, RQ/GT) and CLK		15	pF	fc = 1 MHz
C_{OUT}	Capacitance of Outputs BUSY, INT		10	pF	fc = 1 MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±5%)

TIMING REQUIREMENTS

		8087		8087-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	125	500	ns	
TCLCH	CLK Low Time	118		68		ns	
TCHCL	CLK High Time	69		44		ns	
TCH1CH2	CLK Rise Time		10		10	ns	From 1.0V to 3.5V
TCL2CL2	CLK Fall Time		10		10	ns	From 3.5V to 1.0V
TDVCL	Data In Setup Time	30		20		ns	
TCLDX	Data In Hold Time	10		10		ns	
TRYHCH	READY Setup Time	118		68		ns	
TCHRYX	READY Hold Time	30		20		ns	
TRYLCL	READY Inactive to CLK**	−8		−8		ns	
TGVCH	RQ/GT Setup Time	30		15		ns	
TCHGX	RQ/GT Hold Time	40		30		ns	
TQVCL	QS0-1 Setup Time	30		30		ns	
TCLQX	QS0-1 Hold Time	10		10		ns	
TSACH	Status Active Setup Time	30		30		ns	
TSNCL	Status Inactive Setup Time	30		30		ns	
TILIH	Input Rise Time (Except CLK)		20		20	ns	From 0.8V to 2.0V
TIHIL	Input Fall Time (Except CLK)		12		12	ns	From 2.0V to 0.8V

** See Note 3

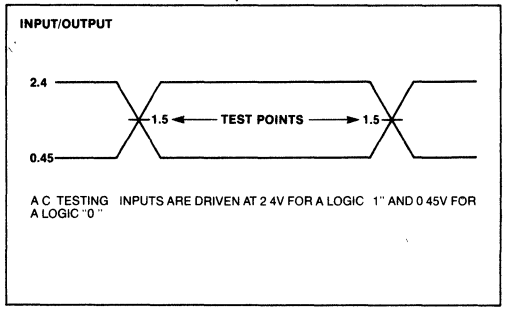
A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

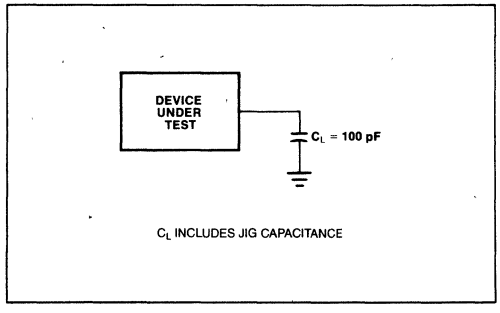
		8087		8087-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLML	Command Active Delay (See Note 1)	10	35	10	35	ns	C_L = 20 – 100pF for all 8087 Outputs (in addition to 8087 self-load)
TCLMH	Command Inactive Delay (See Note 1)	10	35	10	35	ns	
TRYHSH	Ready Active to Status Passive (See Note 2)		110		65	ns	
TCHSV	Status Active Delay	10	110	10	60	ns	
TCLSH	Status Inactive Delay	10	130	10	70	ns	
TCLAV	Address Valid Delay	10	110	10	60	ns	
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TSVLH	Status Valid to ALE High (See Note 1)		15		15	ns	
TCLLH	CLK Low to ALE Valid (See Note 1)		15		15	ns	
TCHLL	ALE Inactive Delay (See Note 1)		15		15	ns	
TCLDV	Data Valid Delay	10	110	10	60	ns	
TCHDX	Data Hold Time	10		10		ns	
TCVNV	Control Active Delay (See Note 1)	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Note 1)	10	45	10	45	ns	
TCHBV	BUSY and INT Valid Delay	10	150	10	85	ns	
TCHDTL	Direction Control Active Delay (See Note 1)		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Note 1)		30		30	ns	
TCLGL	RQ/GT Active Delay	0	85	0	50	ns	C_L = 40pF (in addition to 8087self-load)
TCLGH	RQ/GT Inactive Delay	0	85	0	50	ns	
TOLOH	Output Rise Time		20		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12	ns	From 2.0V to 0.8V

NOTES:
1. Signal at 8284A or 8288 shown for reference only.
2. Applies only to T_3 and wait states.
3. Applies only to T_2 state (8 ns into T_3).

A.C. TESTING INPUT, OUTPUT WAVEFORM

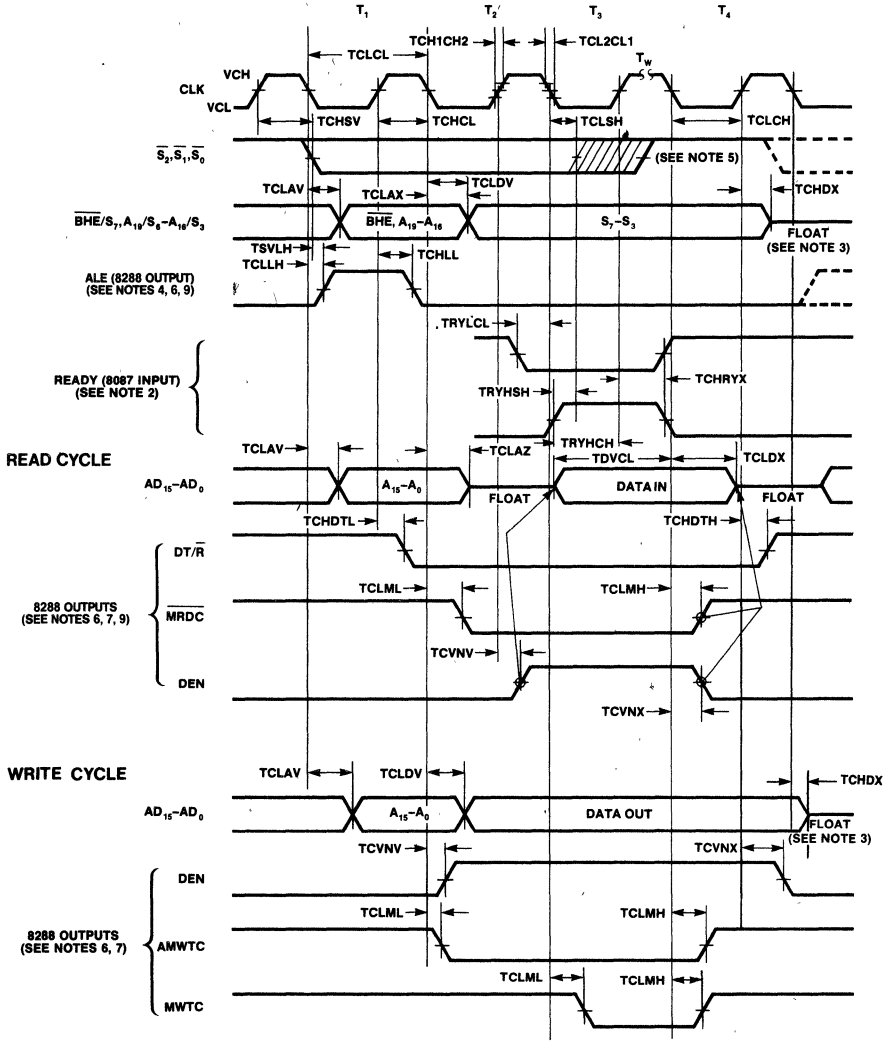


A.C. TESTING LOAD CIRCUIT



WAVEFORMS

MASTER MODE

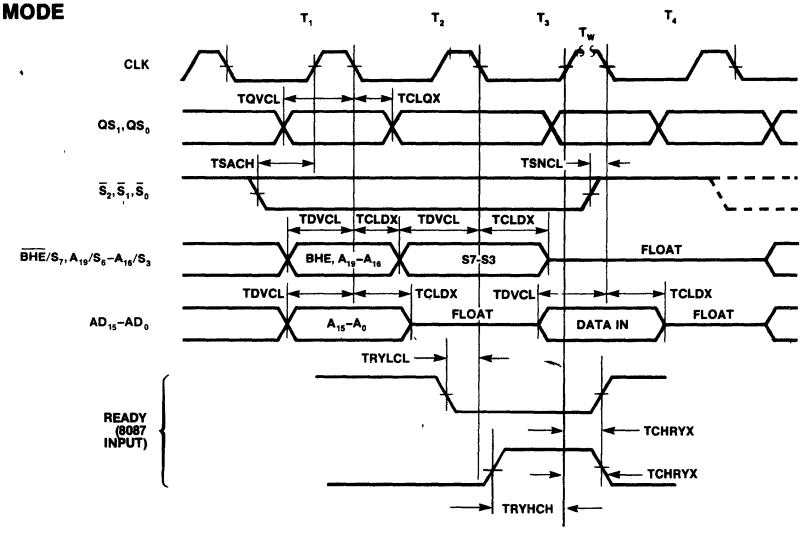


NOTES:

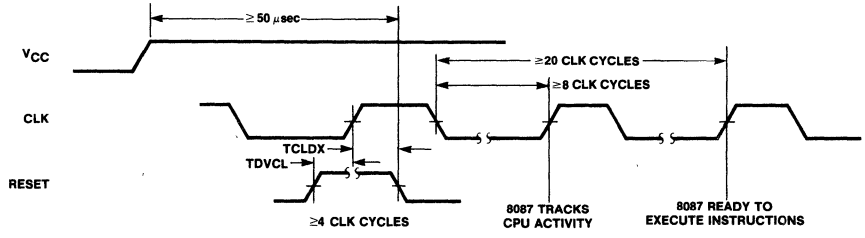
1 ALL SIGNALS SWITCH BETWEEN V_{OL} AND V_{OH} UNLESS OTHERWISE SPECIFIED
2 READY IS SAMPLED NEAR THE END OF T_2, T_3 AND T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED
3 THE LOCAL BUS FLOATS ONLY IF THE 8087 IS RETURNING CONTROL TO THE 8086/8088
4 ALE RISES AT LATER OF (TSVLH, TCLLH)
5 STATUS INACTIVE IN STATE JUST PRIOR TO T_4
6 SIGNALS AT 8284A OR 8288 ARE SHOWN FOR REFERENCE ONLY
7 THE ISSUANCE OF 8288 COMMAND AND CONTROL SIGNALS ($\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$ AND DEN) LAGS THE ACTIVE HIGH 8288 CEN
8 ALL TIMING MEASUREMENTS ARE MADE AT 1 5V UNLESS OTHERWISE NOTED
9 REFER TO THE 82188 ADVANCE BUS CONTROLLER FOR 186, 188 SYSTEMS

WAVEFORMS (Continued)

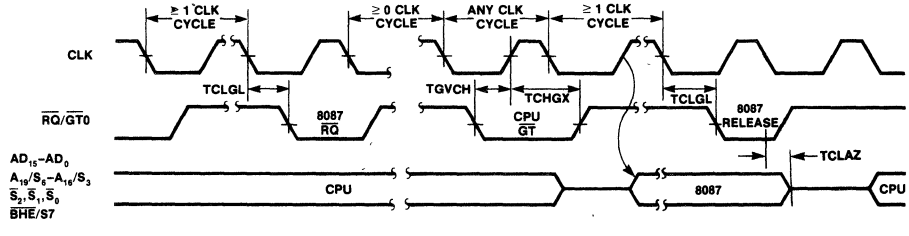
PASSIVE MODE



RESET TIMING



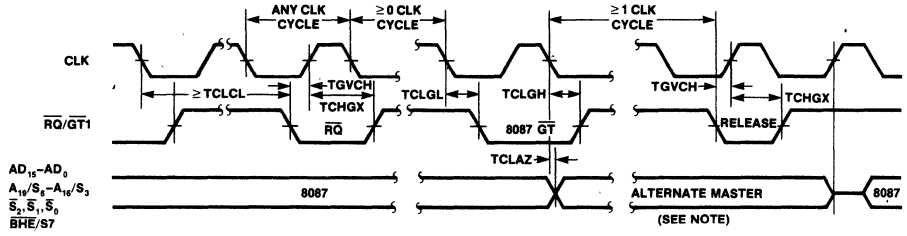
REQUEST/GRANT$_0$ TIMING



NOTE THE CPU PROVIDES ACTIVE PULLUP OF RQ/GT0, SEE TCLGH SPEC

WAVEFORMS (Continued)

REQUEST/GRANT$_1$ TIMING



NOTE ALTERNATE MASTER MAY NOT DRIVE THE BUSES OUTSIDE OF THE REGION SHOWN WITHOUT RISKING BUS CONTENTION

BUSY AND INTERRUPT TIMING

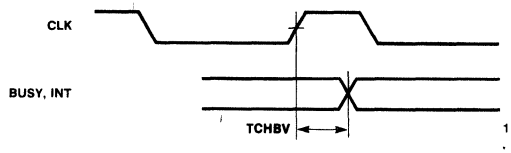


Table 5. 8087 Extensions to the 86/186 Instructions Sets

Data Transfer		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
FLD = LOAD	MF =		00	01	10	11
Integer/Real Memory to ST(0)	ESCAPE MF 1 | MOD 0 0 0 R/M	DISP	38–56 + EA	52–60 + EA	40–60 + EA	46–54 + EA
Long Integer Memory to ST(0)	ESCAPE 1 1 1 | MOD 1 0 1 R/M	DISP	60–68 + EA			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1 | MOD 1 0 1 R/M	DISP	53–65 + EA			
BCD Memory to ST(0)	ESCAPE 1 1 1 | MOD 1 0 0 R/M	DISP	290–310 + EA			
ST(i) to ST(0)	ESCAPE 0 0 1 | 1 1 0 0 0 ST(i)		17–22			
FST = STORE						
ST(0) to Integer/Real Memory	ESCAPE MF 1 | MOD 0 1 0 R/M	DISP	84–90 + EA	82–92 + EA	96–104 + EA	80–90 + EA
ST(0) to ST(i)	ESCAPE 1 0 1 | 1 1 0 1 0 ST(i)		15–22			
FSTP = STORE AND POP						
ST(0) to Integer/Real Memory	ESCAPE MF 1 | MOD 0 1 1 R/M	DISP	86–92 + EA	84–94 + EA	98–106 + EA	82–92 + EA
ST(0) to Long Integer Memory	ESCAPE 1 1 1 | MOD 1 1 1 R/M	DISP	94–105 + EA			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1 | MOD 1 1 1 R/M	DISP	52–58 + EA			
ST(0) to BCD Memory	ESCAPE 1 1 1 | MOD 1 1 0 R/M	DISP	520–540 + EA			
ST(0) to ST(i)	ESCAPE 1 0 1 | 1 1 0 1 1 ST(i)		17–24			
FXCH = Exchange ST(i) and ST(0)	ESCAPE 0 0 1 | 1 1 0 0 1 ST(i)		10–15			
Comparison						
FCOM = Compare						
Integer/Real Memory to ST(0)	ESCAPE MF 0 | MOD 0 1 0 R/M	DISP	60–70 + EA	78–91 + EA	65–75 + EA	72–86 + EA
ST(i) to ST (0)	ESCAPE 0 0 0 | 1 1 0 1 0 ST(i)		40–50			
FCOMP = Compare and Pop						
Integer/Real Memory to ST(0)	ESCAPE MF 0 | MOD 0 1 1 R/M	DISP	63–73 + EA	80–93 + EA	67–77 + EA	74–88 + EA
ST(i) to ST(0)	ESCAPE 0 0 0 | 1 1 0 1 1 ST(i)		45–52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0 | 1 1 0 1 1 0 0 1		45–55			
FTST = Test ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 1 0 0		38–48			
FXAM = Examine ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 1 0 1		12–23			

Table 5. 8087 Extensions to the 86/186 Instruction Sets (cont.)

Constants		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
	MF =		00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 1 0		11–17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 0 0		15–21			
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 1 1		16–22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 0 1		16–22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 1 0		15–21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 0 0		18–24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 0 1		17–23			
Arithmetic						
FADD = Addition						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 0 0 0 R/M	DISP	90–120 + EA	108–143 + EA	95–125 + EA	102–137 + EA
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 0 0 0 ST(i)		70–100 (Note 1)			
FSUB = Subtraction						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 1 0 R R/M	DISP	90–120 + EA	108–143 + EA	95–125 + EA	102–137 + EA
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 1 0 R R/M		70–100 (Note 1)			
FMUL = Multiplication						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 0 0 1 R/M	DISP	110–125 + EA	130–144 + EA	112–168 + EA	124–138 + EA
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 0 0 1 R/M		90–145 (Note 1)			
FDIV = Division						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 1 1 R R/M	DISP	215–225 + EA	230–243 + EA	220–230 + EA	224–238 + EA
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 1 1 R R/M		193–203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1 | 1 1 1 1 1 0 1 0		180–186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1 | 1 1 1 1 1 1 0 1		32–38			
FPREM = Partial Remainder of ST(0) – ST(1)	ESCAPE 0 0 1 | 1 1 1 1 1 0 0 0		15–190			
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1 | 1 1 1 1 1 1 0 0		16–50			

NOTE:
1. If P=1 then add 5 clocks.

Table 5. 8087 Extensions to the 86/186 Instructions Sets (cont.)

		Optional 8,16 Bit Displacement	Clock Count Range
FXTRACT = Extract Components of St(0)	ESCAPE 0 0 1 | 1 1 1 1 0 1 0 0		27–55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 0 0 1		10–17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 0 0 0		10–17
Transcendental			
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1 | 1 1 1 1 0 0 1 0		30–540
FPATAN = Partial Arctangent of ST(0) –ST(1)	ESCAPE 0 0 1 | 1 1 1 1 0 0 1 1		250–800
F2XM1 = $2^{ST(0)} - 1$	ESCAPE 0 0 1 | 1 1 1 1 0 0 0 0		310–630
FYL2X = ST(1) • Log_2 |ST(0)|	ESCAPE 0 0 1 | 1 1 1 1 0 0 0 1		900–1100
FYL2XP1 = ST(1) • Log_2 [ST(0) +1]	ESCAPE 0 0 1 | 1 1 1 1 1 0 0 1		700–1000
Processor Control			
FINIT = Initialized 8087	ESCAPE 0 1 1 | 1 1 1 0 0 0 1 1		2–8
FENI = Enable Interrupts	ESCAPE 0 1 1 | 1 1 1 0 0 0 0 0		2–8
FDISI = Disable Interrupts	ESCAPE 0 1 1 | 1 1 1 0 0 0 0 1		2–8
FLDCW = Load Control Word	ESCAPE 0 0 1 | MOD 1 0 1 R/M	DISP	7–14 + EA
FSTCW = Store Control Word	ESCAPE 0 0 1 | MOD 1 1 1 R/M	DISP	12–18 + EA
FSTSW = Store Status Word	ESCAPE 1 0 1 | MOD 1 1 1 R/M	DISP	12–18 + EA
FCLEX = Clear Exceptions	ESCAPE 0 1 1 | 1 1 1 0 0 0 1 0		2–8
FSTENV = Store Environment	ESCAPE 0 0 1 | MOD 1 1 0 R/M	DISP	40–50 + EA
FLDENV = Load Environment	ESCAPE 0 0 1 | MOD 1 0 0 R/M	DISP	35–45 + EA
FSAVE = Save State	ESCAPE 1 0 1 | MOD 1 1 0 R/M	DISP	197 – 207 + EA
FRSTOR = Restore State	ESCAPE 1 0 1 | MOD 1 0 0 R/M	DISP	197 – 207 + EA
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1 | 1 1 1 1 0 1 1 1		6–12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1 | 1 1 1 1 0 1 1 0		6–12

Table 5. 8087 Extensions to the 86/186 Instructions Sets (cont.)

		Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1 | 1 1 0 0 0 ST(i)	9–16
FNOP = No Operation	ESCAPE 0 0 1 | 1 1 0 1 0 0 0 0	10–16
FWAIT = CPU Wait for 8087	1 0 0 1 1 0 1 1	3+5n*

*n = number of times CPU examines TEST line before 8087 lowers BUSY.

NOTES:

1. if mod=00 then DISP=0*, disp-low and disp-high are absent
   if mod=01 then DISP=disp-low sign-extended to 16-bits, disp-high is absent
   if mod=10 then DISP=disp-high; disp-low
   if mod=11 then r/m is treated as an ST(i) field
2. if r/m=000 then EA=(BX) + (SI) +DISP
   if r/m=001 then EA=(BX) + (DI) +DISP
   if r/m=010 then EA=(BP) + (SI) +DISP
   if r/m=011 then EA=(BP) + (DI) +DISP
   if r/m=100 then EA=(SI) + DISP
   if r/m=101 then EA=(DI) + DISP
   if r/m=110 then EA=(BP) + DISP
   if r/m=111 then EA=(BX) + DISP

   *except if mod=000 and r/m=110 then EA =disp-high; disp-low.
3. MF= Memory Format
   00—32-bit Real
   01—32-bit Integer
   10—64-bit Real
   11—16-bit Integer
4. ST(0)= Current stack top
   ST(i) i^{th} register below stack top
5. d= Destination
   0—Destination is ST(0)
   1—Destination is ST(i)
6. P= Pop
   0—No pop
   1—Pop ST(0)
7. R= Reverse: When d=1 reverse the sense of R
   0—Destination (op) Source
   1—Source (op) Destination
8. For **FSQRT:** $-0 \leq ST(0) \leq +\infty$
   For **FSCALE:** $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
   For **F2XM1:** $0 \leq ST(0) \leq 2^{-1}$
   For **FYL2X:** $0 < ST(0) < \infty$
   $-\infty < ST(1) < +\infty$
   For **FYL2XP1:** $0 \leq |ST(0)| < (2-\sqrt{2})/2$
   $-\infty < ST(1) < \infty$
   For **FPTAN:** $0 \leq ST(0) \leq \pi/4$
   For **FPATAN:** $0 \leq ST(0) < ST(1) < +\infty$

PRELIMINARY

80130/80130-2
iAPX 86/30, 88/30, 186/30, 188/30
iRMX 86 OPERATING SYSTEM PROCESSORS

- **High-Performance 2-Chip Data Processors Containing Operating System Primitives**
- **Standard iAPX 86/10, 88/10 Instruction Set Plus Task Management, Interrupt Management, Message Passing, Synchronization and Memory Allocation Primitives**
- **Fully Extendable To and Compatible With iRMX® 86**
- **Supports Five Operating System Data Types: Jobs, Tasks, Segments, Mailboxes, Regions**
- **35 Operating System Primitives**
- **Built-In Operating System Timers and Interrupt Control Logic Expandable From 8 to 57 Interrupts**
- **8086/80150/80150-2/8088/80186/80188 Compatible At Up To 8 MHz Without Wait States**
- **MULTIBUS® System Compatible Interface**

The Intel iAPX 86/30 and iAPX 88/30 are two-chip microprocessors offering general-purpose CPU (8086) instructions combined with real-time operating system support. They provide a foundation for multiprogramming and multitasking applications. The iAPX 86/30 consists of an iAPX 86/10 (16-bit 8086 CPU) and an Operating System Firmware (OSF) component (80130). The 88/30 consists of the OSF and an iAPX 88/10 (8-bit 8088 CPU). (80186 or 80188 CPUs may be used in place of the 8086 or 8088.)

Both components of the 86/30 and 88/30 are implemented in N-channel, depletion-load, silicon-gate technology (HMOS), and are housed in 40-pin packages. The 86/30 and 88/30 provide all the functions of the iAPX 86/10, 88/10 processors plus 35 operating system primitives, hardware support for eight interrupts, a system timer, a delay timer and a baud rate generator.

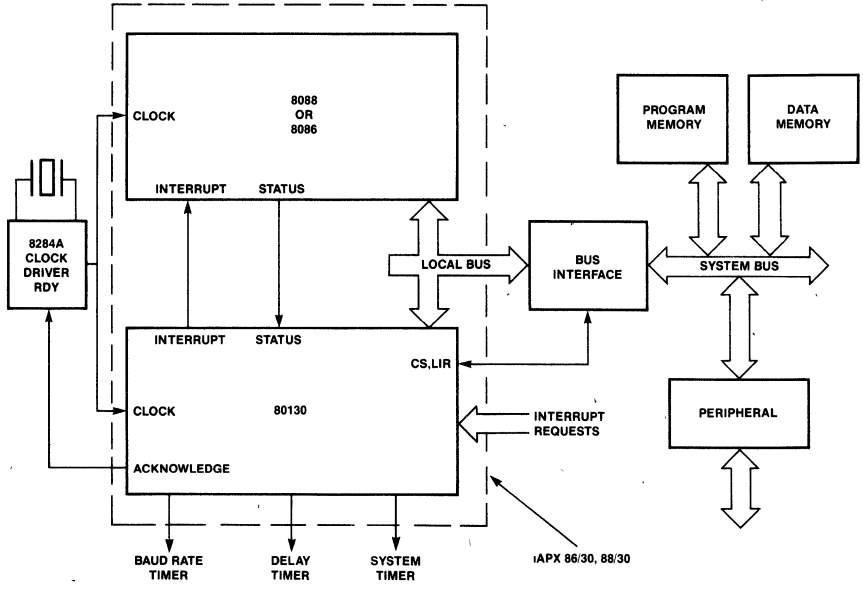


Figure 1. iAPX 86/30, 88/30 Block Diagram

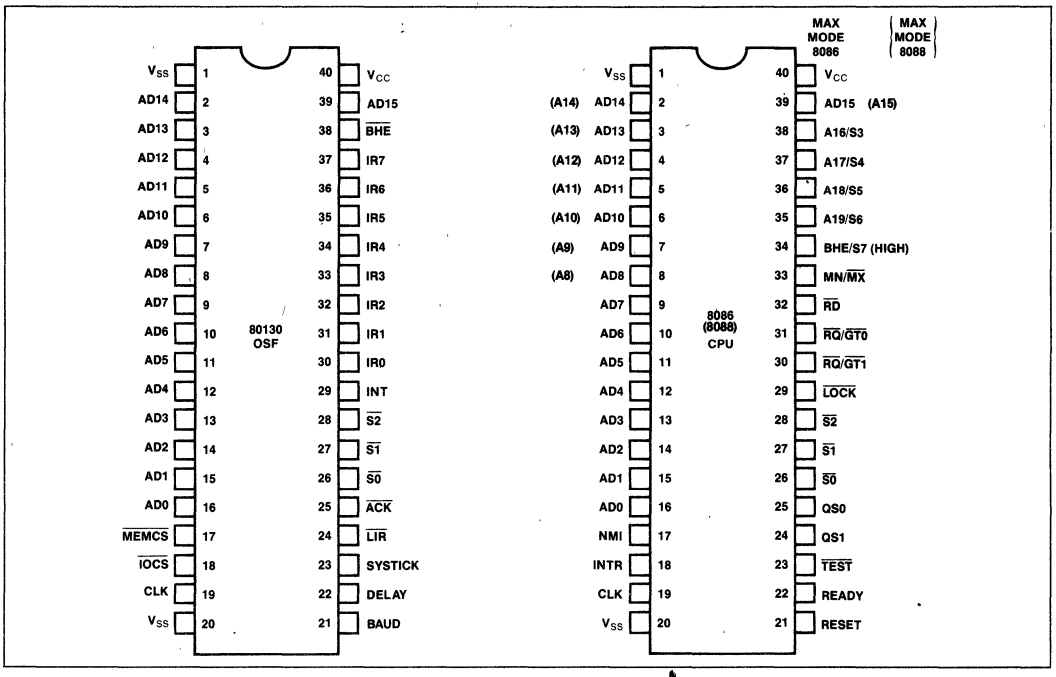


Figure 2. iAPX 86/30, 88/30 Pin Configuration

Table 1. 80130 Pin Description

Symbol	Type	Name and Function
AD_{15}–AD_0	I/O	**Address Data:** These pins constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. These lines are active HIGH. The address presented during T_1 of a bus cycle will be latched internally and interpreted as an 80130 internal address if MEMCS or IOCS is active for the invoked primitives. The 80130 pins float whenever it is not chip selected, and drive these pins only during T_2–T_4 of a read cycle and T_1 of an INTA cycle.
$\overline{BHE}/S_7$		**Bus High Enable:** The 80130 uses the $\overline{BHE}$ signal from the processor to determine whether to respond with data on the upper or lower data pins, or both. The signal is active LOW. $\overline{BHE}$ is latched by the 80130 on the trailing edge of ALE. It controls the 80130 output data as shown. $\overline{BHE}$ A_0 0 0 Word on AD_{15}–AD_0 0 1 Upper byte on AD_{15}–AD_8 1 0 Lower byte on AD_7–AD_0 1 1 Upper byte on AD_7–AD_0
$\overline{S_2}, \overline{S_1}, \overline{S_0}$	I	**Status:** For the 80130, the status pins are used as inputs only. 80130 encoding follows: $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$ 0 0 0 INTA 0 0 1 IORD 0 1 0 IOWR 0 1 1 Passive 1 0 0 Instruction fetch 1 0 1 MEMRD 1 1 X Passive

Table 1. 80130 Pin Description (Continued)

Symbol	Type	Name and Function
CLK	I	**Clock:** The system clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing. The 80130 uses the system clock as an input to the SYSTICK and BAUD timers and to synchronize operation with the host CPU.
INT	O	**Interrupt:** INT is HIGH whenever a valid interrupt request is asserted. It is normally used to interrupt the CPU by connecting it to INTR.
IR_7–IR_0	I	**Interrupt Requests:** An interrupt request can be generated by raising an IR input (LOW to HIGH) and holding it HIGH until it is acknowledged (Edge-Triggered Mode), or just by a HIGH level on an IR input (Level-Triggered Mode).
$\overline{ACK}$	O	**Acknowledge:** This line is LOW whenever an 80130 resource is being accessed. It is also LOW during the first INTA cycle and second INTA cycle if the 80130 is supplying the interrupt vector information. This signal can be used as a bus ready acknowledgement and/or bus transceiver control.
$\overline{MEMCS}$	I	**Memory Chip Select:** This input must be driven LOW when a kernel primitive is being fetched by the CPU. AD_{13}–AD_0 are used to select the instruction.
$\overline{IOCS}$	I	**Input/Output Chip Select:** When this input is low, during an IORD or IOWR cycle, the 80130's kernel primitives are accessing the appropriate peripheral function as specified by the following table: $\overline{BHE}$ A_3 A_2 A_1 A_0 0 X X X X Passive X X X X 1 Passive X 0 1 X X Passive 1 0 0 X 0 Interrupt Controller 1 1 0 0 0 Systick Timer 1 1 0 1 0 Delay Counter 1 1 1 0 0 Baud Rate Timer 1 1 1 1 0 Timer Control
$\overline{LIR}$	O	**Local Bus Interrupt Request:** This signal is LOW when the interrupt request is for a non-slave input or slave input programmed as being a local slave.
V_{CC}		**Power:** V_{CC} is the +5V supply pin.
V_{SS}		**Ground:** V_{SS} is the ground pin.
SYSTICK	O	**System Clock Tick:** Timer 0 Output. Operating System Clock Reference. SYSTICK is normally wired to IR2 to implement operating system timing interrupt.
DELAY	O	**DELAY Timer:** Output of timer 1. Reserved by Intel Corporation for future use.
BAUD	O	**Baud Rate Generator:** 8254 Mode 3 compatible output. Output of 80130 Timer 2.

FUNCTIONAL DESCRIPTION

The increased performance and memory space of iAPX 86/10 and 88/10 microprocessors have proven sufficient to handle most of today's single-task or single-device control applications with performance to spare, and have led to the increased use of these microprocessors to control *multiple* tasks or devices in real-time. This trend has created a new challenge to designers—development of real-time, multitasking application systems and software. Examples of such systems include control systems that monitor and react to external events in real-time, multifunction desktop and personal computers, PABX equipment which constantly controls the telephone traffic in a multiphone office, file servers/disk subsystems controlling and coordinating multiple disks and multiple disk users, and transaction processing systems such as electronics funds transfer.

The iAPX 86/30, 88/30 Operating System Processors

The Intel iAPX 86/30, 88/30 Operating System Processors (OSPs) were developed to help solve this

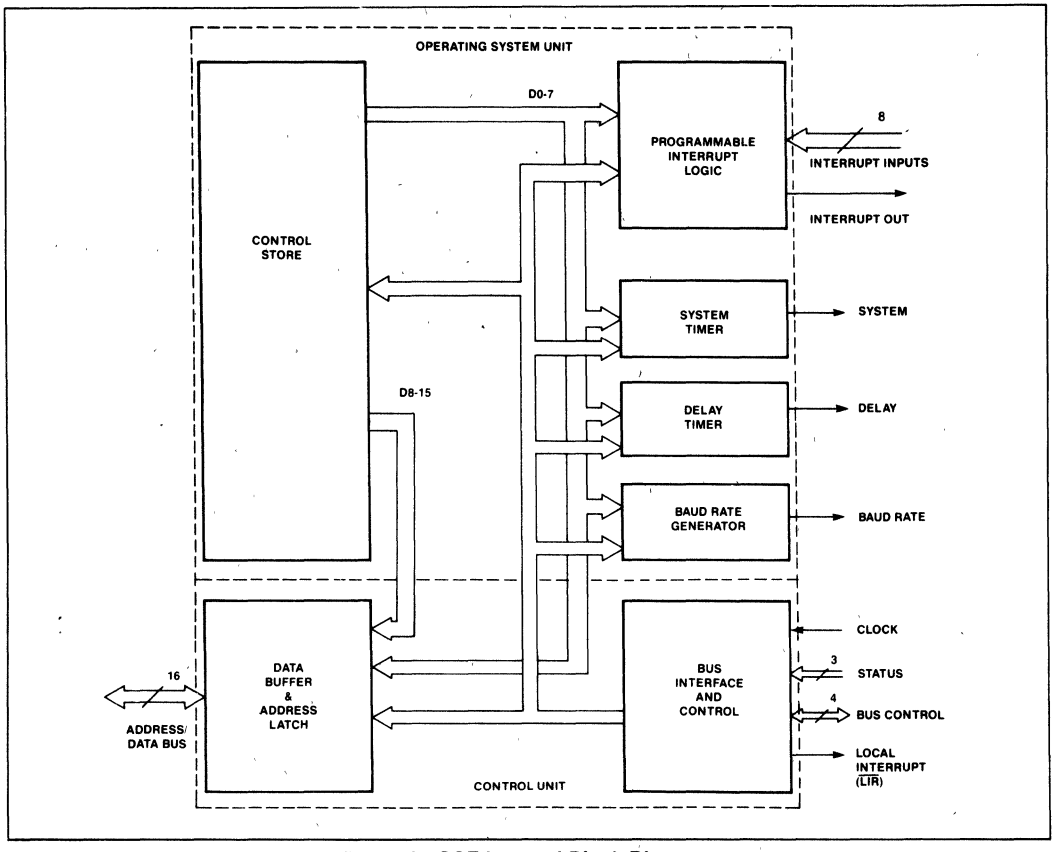


Figure 3. OSF Internal Block Diagram

problem. Their goal is to simplify the design of multitasking application systems by providing a well-defined, fully debugged set of operating system primitives implemented directly in the hardware, thereby removing the burden of designing multitasking operating system primitives from the application programmer.

Both the 86/30 and the 88/30 OSPs are two-chip sets consisting of a main processor, an 8086 or 8088 CPU, and the Intel 80130, Operating System Firmware component (OSF) (see Figure 1). The 80130 provides a set of multitasking kernel primitives, kernel control storage, and the additional support hardware, including system timers and interrupt control, required by these primitives. From the application programmer's viewpoint, the OSF extends the base iAPX 86, 88 architecture by providing 35 operating system primitive instructions, and supporting five new system data types, making the OSF a logical and easy-to-use architectural extension to iAPX 86, 88 system designs.

The OSP Approach

The OSP system data types (SDTs) and primitive instructions allocate, manage and share low-level processor resources in an efficient manner. For example, the OSP implements task context management (managing a task state image consisting of both hardware register set and software control information) for either the basic 86/10 context or the extended 86/20 (8086+8087) numerics context. The OSP manages the entire task state image both while the task is actively executing and while it is inactive. Tasks can be created, put to sleep for specified periods, suspended, executed to perform their functions, and dynamically deleted when their functions are complete.

The Operating System Processors support event-oriented systems designs. Each event may be processed by an individual responding task or along with other closely related events in a common task. External events and interrupts are processed by the OSP interrupt handler primitives using its built-in interrupt controller subsystem as they occur in real-time. The multiple tasks and the multiple events are coordinated by the OSP integral scheduler whose preemptive, priority-based scheduling algorithm and system timers organize and monitor the processing of every task to guarantee that events are processed as they occur in order of relative importance. The 86/30 also provides primitives for intertask communication (by mailboxes) and for mutual exclusion (by regions), essential functions for multitasking applications.

Programming Language Support

Programs for the OSP can be written in ASM 86/88 or PL/M 86/88, Intel's standard system languages for iAPX 86,88 systems.

The Operating System Processor Support Package (iOSP 86) provides an interface library for application programs written in any model of PL/M-86. This library also provides 80130 configuration and initialization support as well as complete user documentation.

OSF PROGRAMMING INTERFACE

The OSF provides 35 operating system kernel primitives which implement multitasking, interrupt management, free memory management, intertask communication and synchronization. Table 4 shows each primitive, and Table 5 gives the execution performance of typical primitives.

OSP primitives are executed by a combination of CPU and OSF (80130) activity. When an OSP primitive is called by an application program task, the iAPX CPU registers and stacks are used to perform the appropriate functions and relay the results to the application programs.

OSP Primitive Calling Sequences

A standard, stack-based, calling sequence is used to invoke the OSF primitives. Before a primitive is called, its operand parameters must be pushed on the task stack. The SI register is loaded with the offset of the last parameter on the stack. The entry code for the primitive is loaded into AX. The primitive invocation call is made with a CPU software interrupt (Table 4). A representative ASM86 sequence for calling a primitive is shown in Figure 4. In PL/M the OSP programmer uses a call to invoke the primitive.

```
                SAMPLE ASSEMBLY LANGUAGE PRIMITIVE CALL

PUSH P1                         ;PUSH PARAMETER 1
PUSH P2                         ;PUSH PARAMETER 2
 .                               .  .  .
 .                               .  .  .
 .                               .  .  .
PUSH PN                         ;PUSH PARAMETER N
PUSH BP                         ;STACK CALLING CONVENTION
MOV BP,SP
LEA SI,SS:NUM_BYTES_PARAM + 2[BP]
                                ;SS:SI POINTS TO FIRST
                                ;PARAMETER ON STACK
MOV AX, ENTRY CODE              ;AX SETS PRIMITIVE ENTRY CODE
INT 184                         ;OSF INTERRUPT

                    OSP PRIMITIVE INVOKED

POP BP
RET NUM_BYTES_PARAM_            ;POP PARAMETERS
                                ;CX CONTAINS EXCEPTION CODES
                                ;DL CONTAINS PARAMETER NUMBER
                                ;   THAT CAUSED EXCEPTION (IF
                                ;   CX IS NON ZERO)
                                ;AX CONTAINS WORD RETURN VALUE
                                ;ES:BX CONTAINS POINTER
                                ;   RETURN VALUE
```

Figure 4. ASM/86 OSP Calling Convention

OSP Functional Description

Each major function of the OSP is described below. These are:

- Job and Task Management
- Interrupt Management
- Free Memory Management
- Intertask Communication
- Intertask Synchronization
- Environmental Control

The system data types (or SDTs) supported by the OSP are capitalized in the description. A short description of each SDT appears in Table 2.

JOB and TASK Management

Each OSP JOB is a controlled environment in which the applications program executes and the OSF system data types reside. Each individual application program is normally a separate OSP JOB, whether it has one initial task (the minimum) or multiple tasks. JOBs partition the system memory into pools. Each memory pool provides the storage areas in which the OSP will allocate TASK state images and other system data types created by the executing TASKs, and free memory for TASK working space. The OSP supports multiple executing TASKs within a JOB by managing the resources used by each, including the CPU registers, NPX registers, stacks, the system data types, and the available free memory space pool.

When a TASK is created, the OSP allocates memory (from the free memory of its JOB environment) for the TASK's stack and data area and initializes the additional TASK attributes such as the TASK priority level and its error handler location. (As an option, the caller of CREATE TASK may assign previously defined stack and data areas to the TASK.) Task priorities are integers between 0 and 255 (the lower the priority number the higher the scheduling priority of the TASK). Generally, priorities up to 128 will be assigned to TASKs which are to process interrupts. Priorities above 128 do not cause interrupts to be disabled, these priorities (129 to 255) are appropriate for non-interrupt TASKs. If an 8087 Numerics Processor Extension is used, the error recovery interrupt level assigned to it will have a higher priority than a TASK executing on it, so that error handling is performed correctly.

EXECUTION STATUS

A TASK has an execution status or execution state. The OSP provides five execution states: RUNNING, READY, ASLEEP, SUSPENDED, and ASLEEP-SUSPENDED.

— A TASK is RUNNING if it has control of the processor.

— A TASK is READY if it is not asleep, suspended, or asleep-suspended. For a TASK to become the running (executing) TASK, it must be the highest priority TASK in the ready state.

— A TASK is ASLEEP if it is waiting for a request to be granted or a timer event to occur. A TASK may put itself into the ASLEEP state.

— A TASK is SUSPENDED if it is placed there by another TASK or if it suspends itself. A TASK may have multiple suspensions, the count of suspensions is managed by the OSP as the TASK suspension depth.

— A TASK is ASLEEP-SUSPENDED if it is both waiting and suspended.

TASK attributes, the CPU register values, and the 8087 register values (if the 8087 is configured into the application) are maintained by the OSP in the TASK state image. Each TASK will have a unique TASK state image.

SCHEDULING

The OSP schedules the processor time among the various TASKs on the basis of priority. A TASK has an execution priority relative to all other TASKs in the system, which the OSP maintains for each TASK in its TASK state image. When a TASK of higher priority than the executing TASK becomes ready to execute, the OSP switches the control of the processor to the higher priority TASK. First, the OSP saves the outgoing (lower priority) TASK's state including CPU register values in its TASK state image. Then, it restores the CPU registers from the TASK state image of the incoming (higher priority) TASK. Finally, it causes the CPU to start or resume executing the higher priority TASK.

TASK scheduling is performed by the OSP. The OSP's priority-oriented preemptive scheduler determines which TASK executes by comparing their relative priorities. The scheduler insures that the highest priority TASK with a status of READY will execute. A TASK will continue to execute until an interrupt with a higher priority occurs, or until it requests unavailable resources, for which it is willing to wait, or until it makes specific resources available to a higher priority TASK waiting for those resources.

TASKs can become READY by receiving a message, receiving control, receiving an interrupt, or by timing out. The OSP always monitors the status of all the TASKs (and interrupts) in the system. Preemptive scheduling allows the system to be responsive to the external environment while only devoting CPU resources to TASKs with work to be performed.

TIMED WAIT

The OSP timer hardware facilities support timed waits and timeouts. Thus, in many primitives, a TASK can specify the length of time it is prepared to wait for an event to occur, for the desired resources to become available or for a message to be received at a MAILBOX. The timing interval (or System Tick) can be adjusted, with a lower limit of 1 millisecond.

APPLICATION CONTROL OF TASK EXECUTION

Programs may alter TASK execution status and priority dynamically. One TASK may suspend its own execution or the execution of another TASK for a period of time, then resume its execution later. Multiple suspensions are provided. A suspended TASK may be suspended again.

The eight OSP Job and TASK management primitives are:

CREATE JOB	Partitions system resources and creates a TASK execution environment.
CREATE TASK	Creates a TASK state image. Specifies the location of the TASK code instruction stream, its execution priority, and the other TASK attributes.

DELETE TASK	Deletes the TASK state image, removes the instruction stream from execution and deallocates stack resources. Does not delete INTERRUPT TASKS.
SUSPEND TASK	Suspends the specified TASK or, if already suspended, increments its suspension depth by one. Execute state is SUSPEND.
RESUME TASK	Decrements the TASK suspension depth by one. If the suspension depth is then zero, the primitive changes the task execution status to READY, or ASLEEP (if ASLEEP/SUSPENDED).
SLEEP	Places the requesting TASK in the ASLEEP state for a specified number of System Ticks. (The TICK interval can be configured down to 1 millisecond.)
SET PRIORITY	Alters the priority of a TASK.

Interrupt Management

The OSP supports up to 256 interrupt levels organized in an interrupt vector, and up to 57 external interrupt sources of which one is the NMI (Non-Maskable Interrupt). The OSP manages each interrupt level independently. The OSF INTERRUPT SUBSYSTEM provides two mechanisms for interrupt management: INTERRUPT HANDLERs and INTERRUPT TASKs. INTERRUPT HANDLERs disable all maskable interrupts and should be used only for servicing interrupts that require little processing time. Within an INTERRUPT HANDLER only certain OSF Interrupt Management primitives (DISABLE, ENTER INTERRUPT, EXIT INTERRUPT, GET LEVEL, SIGNAL INTERRUPT) and basic CPU instructions can be used, other OSP primitives cannot be. The INTERRUPT TASK approach permits all OSP primitives to be issued and masks only lower priority interrupts.

Work flow between an INTERRUPT HANDLER and an INTERRUPT TASK assigned to the same level is regulated with the SIGNAL INTERRUPT and WAIT INTERRUPT primitives. The flow is asynchronous. When an INTERRUPT HANDLER signals an INTERRUPT TASK, the INTERRUPT HANDLER becomes immediately available to process another interrupt. The number of interrupts (specified for the level) the INTERRUPT HANDLER can queue for the INTERRUPT TASK can be limited to the value specified in the SET INTERRUPT primitive. When the INTERRUPT TASK is finished processing, it issues a WAIT INTERRUPT primitive, and is immediately ready to process the queue of interrupts that the INTERRUPT HANDLER has built with repeated SIGNAL INTERRUPT primitives while the INTERRUPT TASK was processing. If there were no interrupts at the level, the queue is empty and the INTERRUPT TASK is SUSPENDED. See the Example (Figure 5) and Figures 6 and 7.

OSP external INTERRUPT LEVELs are directly related to internal TASK scheduling priorities. The OSP maintains a single list of priorities including both tasks and INTERRUPT LEVELs. The priority of the executing TASK automatically determines which interrupts are masked. Interrupts are managed by INTERRUPT LEVEL number. The OSP supports eight levels directly and may be extended by means of slave 8259As to a total of 57.

The nine Interrupt Management OSP primitives are:

DISABLE	Disables an external INTERRUPT LEVEL.
ENABLE	Enables an external INTERRUPT LEVEL.
ENTER INTERRUPT	Gives an Interrupt Handler its own data segment, separate from the data segment of the interrupted task.
EXIT INTERRUPT	Performs an "END of INTERRUPT" operation. Used by an INTERRUPT HANDLER which does not invoke an INTERRUPT TASK. Reenables interrupts, when the INTERRUPT HANDLER gives up control.
GET LEVEL	Returns the interrupt level number of the executing INTERRUPT HANDLER.
RESET INTERRUPT	Cancels the previous assignment made to an interrupt level by SET INTERRUPT primitive request. If an INTERRUPT TASK has been assigned, it is also deleted. The interrupt level is disabled.
SET INTERRUPT	Assigns an INTERRUPT HANDLER to an interrupt level and, optionally, an INTERRUPT TASK.

```
/*  CODE EXAMPLE A   INTERRUPT TASK TO KEEP TRACK OF TIME-OF-DAY
DECLARE SECOND$COUNT BYTE,
    MINUTE$COUNT BYTE,
    HOURS$COUNT BYTE;
TIME$TASK:  PROCEDURE;
    DECLARE TIME$EXCEPT$CODE WORD;
    AC$CYCLE$COUNT=0;
    CALL RQ$SET$INTERRUPT(AC$INTERRUPT$LEVEL, 01H),
        @AC$HANDLER,0,@TIME$EXCEPT$CODE);
    CALL RQ$RESUME$TASK(INIT$TASK$TOKEN,@TIME$EXCEPT$CODE);
    DO HOUR$COUNT=0 TO 23;
        DO MINUTE$COUNT=0 TO 59;
            DO SECOND$COUNT=0 TO 59;
                CALL RQ$WAIT$INTERRUPT(AC$INTERRUPT$LEVEL,
                    @TIME$EXCEPT$CODE);
                IF SECOND$COUNT MOD 5=0
                    THEN CALL PROTECTED$CRT$OUT(BEL);
                END;    /*  SECOND LOOP  */
            END;    /*  MINUTE LOOP  */
        END;    /*  HOUR LOOP  */
    CALL RQ$RESET$INTERRUPT(AC$INTERRUPT$LEVEL, @TIME$EXCEPT$CODE);
    END TIME$TASK;
/*  CODE EXAMPLE B     INTERRUPT HANDLER TO SUBDIVIDE A.C. SIGNAL BY 60.  */
DECLARE AC$CYCLE$COUNT BYTE;
AC$HANDLER:  PROCEDURE INTERRUPT 59;
    DECLARE AC$EXCEPT$CODE WORD;
    AC$CYCLE$COUNT=AC$CYCLE$COUNT +1;
    IF AC$CYCLE$COUNT>=60 THEN DO;
        AC$CYCLE$COUNT=0;
        CALL RQ$SIGNAL$INTERRUPT(AC$INTERRUPT$LEVEL,@AC$EXCEPT$CODE);
        END;
    END  AC$HANDLER;
```

Figure 5. OSP Examples

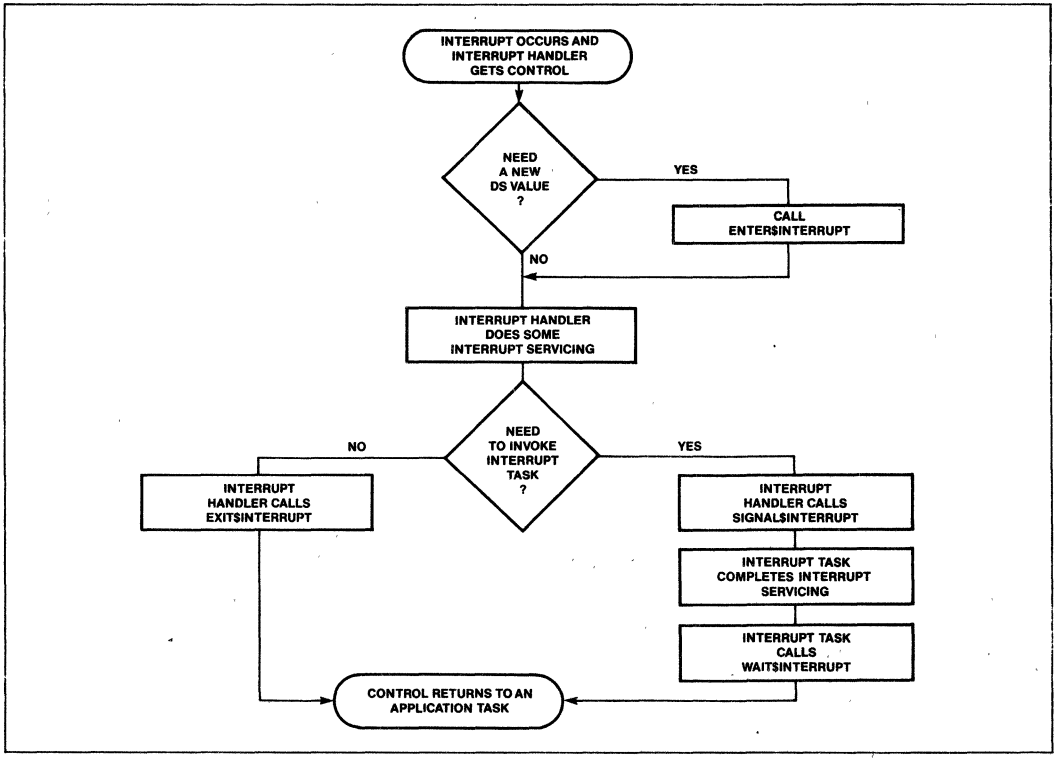


Figure 6. Interrupt Handling Flowchart

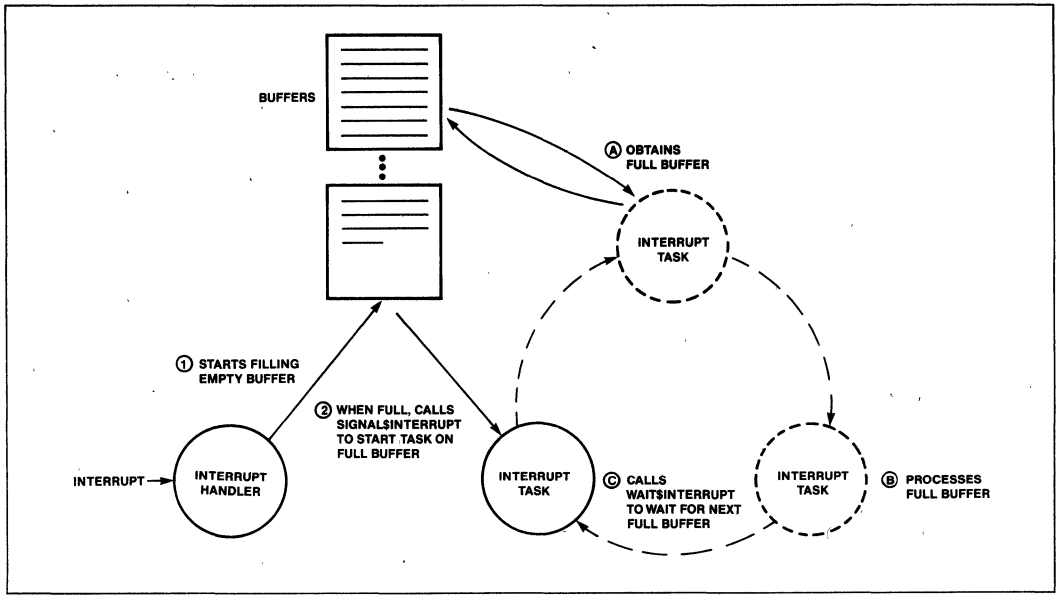


Figure 7. Multiple Buffer Example

SIGNAL INTERRUPT	Used by an INTERRUPT HANDLER to activate an Interrupt Task.
WAIT INTERRUPT	Suspends the calling Interrupt Task until the INTERRUPT HANDLER performs a SIGNAL INTERRUPT to invoke it. If a SIGNAL INTERRUPT for the task has occurred, it is processed.

FREE MEMORY MANAGEMENT

The OSP Free Memory Manager manages the memory pool which is allocated to each JOB for its execution needs. (The CREATE JOB primitive allocates the new JOB's memory pool from the memory pool of the parent JOB.) The memory pool is part of the JOB resources but is not yet allocated between the tasks of the JOB. When a TASK, MAILBOX, or REGION system data type structure is created within that JOB, the OSP implicitly allocates memory for it from the JOB's memory pool, so that a separate call to allocate memory is not required. OSP primitives that use free memory management implicitly include CREATE JOB, CREATE TASK, DELETE TASK, CREATE MAILBOX, DELETE MAILBOX, CREATE REGION, and DELETE REGION. The CREATE SEGMENT primitive explicitly allocates a memory area when one is needed by the TASK. For example, a TASK may explicitly allocate a SEGMENT for use as a memory buffer. The SEGMENT length can be any multiple of 16 bytes between 16 bytes and 64K bytes in length. The programmer may specify any number of bytes from 1 byte to 64 KB, the OSP will transparently round the value up to the appropriate segment size.

The two explicit memory allocation/deallocation primitives are:

CREATE SEGMENT	Allocates a SEGMENT of specified length (in 16-byte-long paragraphs) from the JOB Memory Pool.
DELETE SEGMENT	Deallocates the SEGMENT's memory area, and returns it to the JOB memory pool.

Intertask Communication

The OSP has built-in intertask synchronization and communication, permitting TASKs to pass and share information with each other. OSP MAILBOXes contain controlled handshaking facilities which guarantee that a *complete* message will always be sent from a sending TASK to a receiving TASK. Each MAILBOX consists of two interlocked queues, one of TASKs

and the other of Messages. Four OSP primitives for intertask synchronization and communication are provided:

CREATE MAILBOX	Creates intertask message exchange.
DELETE MAILBOX	Deletes an intertask message exchange.
RECEIVE MESSAGE	Calling TASK receives a message from the MAILBOX.
SEND MESSAGE	Calling TASK sends a message to the MAILBOX.

The CREATE MAILBOX primitive allocates a MAILBOX for use as an information exchange between TASKs. The OSP will post information at the MAILBOX in a FIFO (First-In First-Out) manner when a SEND MESSAGE instruction is issued. Similarily, a message is retrieved by the OSP if a TASK issues a RECEIVE MESSAGE primitive. The TASK which creates the MAILBOX may make it available to other TASKs to use.

If no message is available, the TASK attempting to receive a message may choose to wait for one or continue executing.

The queue management method for the task queue (FIFO or PRIORITY) determines which TASK in the MAILBOX TASK queue will receive a message from the MAILBOX. The method is specified in the CREATE MAILBOX primitive.

Intertask Synchronization and Mutual Exclusion

Mutual exclusion is essential to multiprogramming and multiprocessing systems. The REGION system data type implements mutual exclusion. A REGION is represented by a queue of TASKS waiting to use a resource which must be accessed by only one TASK at a time. The OSP provides primitives to use REGIONs to manage mutually exclusive data and resources. Both critical code sections and shared data structures can be protected by these primitives from simultaneous use by more than one task. REGIONs support both FIFO (First-In First-Out) or Priority queueing disciplines for the TASKS seeking to enter the REGION. The REGION SDT can also be used to implement software locks.

Multiple REGIONs are allowed, and are automatically exited in the reverse order of entry. While in a REGION, a TASK cannot be suspended by itself or any other TASK, and thereby avoids deadlock.

There are five OSP primitives for mutual exclusion:

CREATE REGION	Create a REGION (lock).
SEND CONTROL	Give up the REGION.
ACCEPT CONTROL	Request the REGION, but do not wait if it is not available.
RECEIVE CONTROL	Request a REGION, wait if not immediately available.
DELETE REGION	Delete a REGION.

The OSP also provides dynamic priority adjustment for TASKs within priority REGIONs: If a higher-priority TASK issues a RECEIVE CONTROL primitive, while a (lower-priority) TASK has the use of the same REGION, the lower-priority TASK will be transparently, and temporarily, elevated to the waiting TASK's priority until it relinquishes the REGION via SEND CONTROL. At that point, since it is no longer using the critical resource, the TASK will have its normal priority restored.

OSP Control Facilities

The OSP also includes system primitives that provide both control and customization capabilities to a multitasking system. These primitives are used to control the deletion of SDTs and the recovery of free memory in a system, to allow interrogation of operating system status, and to provide uniform means of adding user SDTs and type managers.

DELETION CONTROL

Deletion of each OSP system data type is explicitly controlled by the applications programmer by setting a deletion attribute for that structure. For example, if a SEGMENT is to be kept in memory until DMA activity is completed, its deletion attribute should be disabled. Each TASK, MAILBOX, REGION, and SEGMENT SDT is created with its deletion attribute enabled (i.e., they may be deleted). Two OSP primitives control the deletion attribute: ENABLE DELETION and DISABLE DELETION.

ENVIRONMENTAL CONTROL

The OSP provides inquiry and control operations which help the user interrogate the application environment and implement flexible exception handling. These features aid in run-time decision making and in application error processing and recovery. There are five OSP environmental control primitives.

OS EXTENSIONS

The OSP architecture is defined to allow new user-defined System Data Types and the primitives to manipulate them to be added to OSP capabilities

provided by the built-in System Data Types. The type managers created for the user-defined SDTs are called user OS extensions and are installed in the system by the SET OS EXTENSION primitive. Once installed, the functions of the type manager may be invoked with user primitives conforming to the OSP interface. For well-structured extended architectures, each OS extension should support a separate user-defined system data type, and every OS extension should provide the same calling sequence and program interface for the user as is provided for a built-in SDT. The type manager for the extension would be written to suit the needs of the application. OSP interrupt vector entries (224–255) are reserved for user OS extensions and are not used by the OSP. After assigning an interrupt number to the extension, the extension user may then call it with the standard OSP call sequence (Figure 4), and the unique software interrupt number assigned to the extension.

ENABLE DELETION	Allows a specific SEGMENT, TASK, MAILBOX, or REGION SDT to be deleted.
DISABLE DELETION	Prevents a specific SEGMENT, TASK, MAILBOX, or REGION SDT from being deleted.
GET TYPE	Given a token for an instance of a system data type, returns the type code.
GET TASK TOKENS	Returns to the caller information about the current task environment.
GET EXCEPTION HANDLER	Returns information about the calling TASK's current information handler: its address, and when it is used.
SET EXCEPTION HANDLER	Provides the address and usage of an exception handler for a TASK.
SET OS EXTENSION	Modifies one of the interrupt vector entries reserved for OS extensions (224–255) to point to a user OS extension procedure.
SIGNAL EXCEPTION	For use in OS extension error processing.

EXCEPTION HANDLING

The OSP supports exception handlers. These are similar to CPU exception handlers such as OVERFLOW and ILLEGAL OPERATION. Their purpose is to allow the OSP primitives to report parameter errors in primitive calls, and errors in primitive usage. Exception handling procedures are flexible and can be individually programmed by the application. In general, an exception handler if called will perform one or more of the following functions:

—Log the Error.
—Delete/Suspend the Task that caused the exception.
—Ignore the error, presumably because it is not serious.

An EXCEPTION HANDLER is written as a procedure. If PLM/86 is used, the "compact," "medium" or "large" model of computation should be specified for the compilation of the program. The mode in which the EXCEPTION HANDLER operates may be specified in the SET EXCEPTION HANDLER primitive. The return information from a primitive call is shown in Figure 4. CX is used to return standard system error conditions. Table 7 shows a list of these conditions, using the *default* EXCEPTION HANDLER of the OSP.

HARDWARE DESCRIPTION

The 80130 operates in a closely coupled mode with the iAPX 86/10 or 88/10 CPU. The 80130 resides on the CPU local multiplexed bus (Figure 8). The main processor is always configured for maximum mode operation. The 80130 automatically selects between its 88/30 and 86/30 operating modes.

The 80130 used in the 86/30 configuration, as shown in Figure 8 (or a similar 88/30 configuration), operates at both 5 and 8 MHz without requiring processor wait states. Wait state memories are fully supported, however. The 80130 may be configured with both an 8087 NPX and an 8089 IOP, and provides full context control over the 8087.

The 80130 (shown in Figure 3) is internally divided into a control unit (CU) and operating system unit (OSU). The OSU contains facilities for OSP kernel support including the system timers for scheduling and timing waits, and the interrupt controller for interrupt management support.

iAPX 86/30, iAPX 88/30 System Configuration

The 80130 is both I/O and memory mapped to the local CPU bus. The CPU's status S0/-S2/ is decoded along with IOCS/ (with BHE and AD_3–AD_0) or MEMCS/ (with AD_{13}–AD_0). The pins are internally latched. See Table 1 for the decoding of these lines.

Memory Mapping

Address lines A_{19}–A_{14} can be used to form MEMCS/ since the 80130's memory-mapped portion is aligned along a 16K-byte boundry. The 80130 can reside on any 16K-byte boundry excluding the highest (FC000H-FFFFFH) and lowest (00000H-003FFH). The 80130 control store code is position-independent except as limited above, in order to make it compatible with many decoding logic designs. AD_{13}–AD_0 are decoded by the 80130's kernel control store.

I/O Mapping

The I/O-mapped portion of the 80130 must be aligned along a 16-byte boundry. Address lines A_{15}–A_4 should be used to form IOCS/.

System Performance

The approximate performance of representitive OSP primitives is given in Table 5. These times are shown for a typical iAPX 86/30 implementation with an 8 MHz clock. These execution times are very comparable to the execution times of similar functions in minicomputers (where available) and are an order of magnitude faster than previous generation microprocessors.

Initialization

Both application system initialization and OSP-specific initialization/configuration are required to use the OSP. Configuration is based on a "database" provided by the user to the iOSP 86 support package. The OSP-specific initialization and configuration information area is assigned to a user memory address adjacent to the 80130's memory-mapped location. (See Application Note 130 for further details.) The configuration data defines whether 8087 support is configured in the system, specifies if slave 8259A interrupt controllers are used in addition to the 80130, and sets the operating system time base (Tick Interval). Also located in the configuration area are the exception handler control parameters, the address location of the (separate) application system configuration area and the OSP extensions in use. The OSP application system configuration area may be located anywhere in the user memory and must include the starting address of the application instruction code to be executed, plus the locations of the RAM memory blocks to be managed by the OSP free memory manager. Complete application system support and the required 80130 configuration support are provided by the iAPX 86/30 and iAPX 88/30 OPERATING SYSTEM PROCESSOR SUPPORT PACKAGE (iOSP 86).

RAM Requirements

The OSP manages its own interrupt vector, which is assigned to low RAM memory. Working RAM storage is required as stack space and data area. The memory space must be allocated in user RAM.

OSP interrupt vector memory locations 0H–3FFH must be RAM based. The OSP requires 2 bytes of allocated RAM. The processor working storage is dynamically allocated from free memory. Approximately 300 bytes of stack should be allocated for each OSP task.

TYPICAL SYSTEM CONFIGURATION

Figure 8 shows the processing cluster of a "typical" iAPX 86/30 or iAPX 88/30 OSP system. Not shown are subsystems likely to vary with the application. The configuration includes an 8086 (or 8088) operating in maximum mode, an 8284A clock generator and an 8288 system controller. Note that the 80130 is located on the CPU side of any latches or transceivers. See Intel Application Note 130 for further details on configuration.

OSP Timers

The OSP Timers are connected to the lower half of the data bus and are addressed at even addresses. The timers are read as two successive bytes, always LSB followed by MSB. The MSB is always latched on a read operation and remains latched until read. Timers are not gatable.

Baud Rate Generator

The baud rate generator is 8254 compatible (square wave mode 3). Its output, BAUD, is initially high and remains high until the Count Register is loaded. The first falling edge of the clock after the Count Register is loaded causes the transfer of the internal counter to the Count Register. The output stays high for N/2 [(N+1)/2 if N is odd] and then goes low for N/2 [(N−1)/2 if N is odd]. On the falling edge of the clock which signifies the final count for the output in low state, the output returns to high state and the Count Register is transferred to the internal counter. The whole process is then repeated. Baud Rates are shown in Table 6.

The baud rate generator is located at 0CH (12), relative to the 16-byte boundary in the I/O space in which the 80130 component is located ("OSF" in the following example), the timer control word is located at

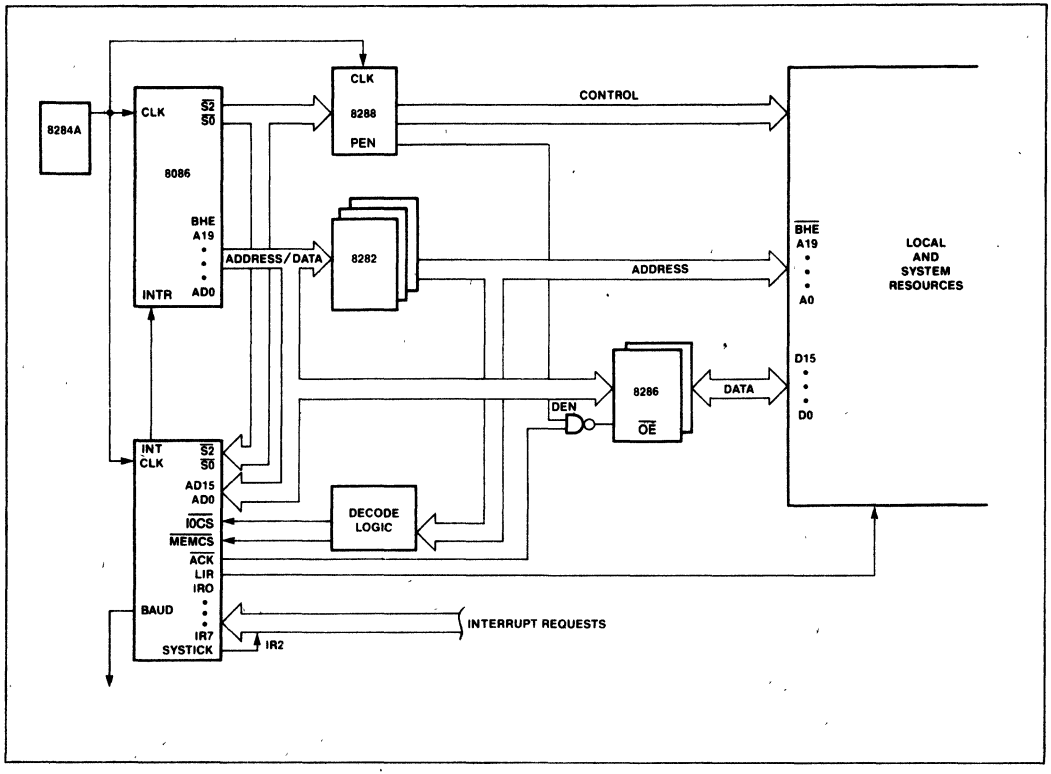


Figure 8. Typical OSP Configuration

relative address, 0EH(14). Timers are addressed with IOCS=0. Timers 0 and 1 are assigned to the use by the OSP, and should not be altered by the user.

For most baud-rate generator applications, the command byte

0B6H Read/Write Baud-Rate Delay Value

will be used. A typical sequence to set a baud rate of 9600 using a count value of 52 follows (see Table 6):

```
MOV  AX,0B6H     ;Prepare to Write Delay to
                  Timer 3.
OUT  OSF+14,AX   ;Control Word.
MOV  AX, 52
OUT  OSF+12,AL   ;LSB written first
XCHG AL,AH
OUT  OSF+12,AL   ;MSB written after.
```

The 80130 timers are subset compatible with 8254 timers.

Interrupt Controller

The Programmable Interrupt Controller (PIC), is also an integral unit of the 80130. Its eight input pins handle eight vectored priority interrupts. One of these pins must be used for the SYSTICK time function in timing waits, using an external connection as shown. During the 80130 initialization and configuration sequence, each 80130 interrupt pin is individually programmed as either level or edge sensitive. External slave 8259A interrupt controllers can be used to expand the total number of OSP external interrupts to 57.

In addition to standard PIC funtions, 80130 PIC unit has an $\overline{LIR}$ output signal, which when low indicates an interrupt acknowledge cycle. $\overline{LIR}$=0 is provided to control the 8289 Bus Arbiter SYSB/$\overline{RESB}$ pin. This will avoid the need of requesting the system bus to acknowledge local bus non-slave interrupts. The user defines the interrupt system as part of the configuration.

INTERRUPT SEQUENCE

The OSP interrupt sequence is as follows:

1. One or more of the interrupts is set by a low-to-high transition on edge-sensitive IR inputs or by a high input on level-sensitive IR inputs.
2. The 80130 evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an interrupt acknowledge cycle which is encoded in $\overline{S_2}$-$\overline{S_0}$.
4. Upon receiving the first interrupt acknowledge from the CPU, the highest-priority interrupt is set by the 80130 and the corresponding edge detect latch is reset. The 80130 does not drive the address/data bus during this bus cycle but does acknowledge the cycle by making $\overline{ACK}$=0 and sending the $\overline{LIR}$ value for the IR input being acknowledged.
5. The CPU will then initiate a second interrupt acknowledge cycle. During this cycle, the 80130 will supply the cascade address of the interrupting input at T_1 on the bus and also release an 8-bit pointer onto the bus if appropriate, where it is read by the CPU. If the 80130 does supply the pointer, then $\overline{ACK}$ will be low for the cycle. This cycle also has the value $\overline{LIR}$ for the IR input being acknowledged.
6. This completes the interrupt cycle. The ISR bit remains set until an appropriate EXIT INTERRUPT primitive (EOI command) is called at the end of the Interrupt Handler.

OSP APPLICATION EXAMPLE

Figure 5 shows an application of the OSP primitives to keep track of time of day in a simplified example. The system design uses a 60 Hz A.C. signal as a time base. The power supply provides a TTL-compatible signal which drives one of 80130 edge-triggered interrupt request pins once each A.C. cycle. The Interrupt Handler responds to the interrupts, keeping track of one second's A.C. cycles. The Interrupt Task counts the seconds and after a day deletes itself. In typical systems it might perform a data logging operation once each day. The Interrupt Handler and Interrupt Task are written as separate modular programs.

The Interrupt Handler will actually service interrupt 59 when it occurs. It simply counts each interrupt, and at a count of 60 performs a SIGNAL INTERRUPT to notify the Interrupt Task that a second has elapsed. The Interrupt Handler (ACS HANDLER) was assigned to this level by the SET INTERRUPT primitive. After doing this, the Interrupt Task performed the Primitive RESUME TASK to resume the application task (INITS TASKS TOKEN).

The main body of the task is the counting loop. The Interrupt Task is signaled by the SIGNAL INTERRUPT primitive in the Interrupt Handler (at interrupt level ACS INTERRUPTS LEVEL). When the task is signalled by the Interrupt Handler it will execute the loop exactly one time, increasing the time count variables. Then it will execute the WAIT INTERRUPT primitive, and wait until awakened by the Interrupt Handler. Normally, the task will now wait some period of time for the next signal. However, since the interface between the Handler and the Task is asynchronous, the handler may have already queued the interrupt for servicing, the writer of the task does not have to worry about this possibility.

At the end of the day, the task will exit the loop and execute RESET INTERRUPT, which disables the interrupt level, and deletes the interrupt task. The OSP now reclaims the memory used by the Task and schedules another task. If an exception occurs, the coded value for the exception is available in TIMES EXCEPTS CODE after the execution of the primitive.

A typical PL/M-86 calling sequence is illustrated by the call to RESET INTERRUPT shown in Figure 5.

Table 2. OSP System Data Type Summary

Job	Jobs are the means of organizing the program environment and resources. An application consists of one or more jobs. Each iAPX 86/30 system data type is contained in some job. Jobs are independent of each other, but they may share access to resources. Each job has one or more tasks, one of which is an initial task. Jobs are given pools of memory, and they may create subordinate offspring jobs, which may borrow memory from their parents.
Task	Tasks are the means by which computations are accomplished. A task is an instruction stream with its own execution stack and private data. Each task is part of a job and is restricted to the resources provided by its job. Tasks may perform general interrupt handling as well as other computational functions. Each task has a set of attributes, which is maintained for it by the iAPX 86/30, which characterize its status. These attributes are: its containing job its register context its priority (0–255) its execution state (asleep, suspended, ready, running, asleep/suspended). its suspension depth its user-selected exception handler its optional 8087 extended task state
Segment	Segments are the units of memory allocation. A segment is a physically contiguous sequence of 16-byte, 8086 paragraph-length, units. Segments are created dynamically from the free memory space of a Job as one of its Tasks requests memory for its use. A segment is deleted when it is no longer needed. The iAPX 86/30 maintains and manages free memory in an orderly fashion, it obtains memory space from the pool assigned to the containing job of the requesting task and returns the space to the job memory pool (or the parent job pool) when it is no longer needed. It does not allocate memory to create a segment if sufficient free memory is not available to it, in that case it returns an error exception code.
Mailbox	Mailboxes are the means of intertask communication. Mailboxes are used by tasks to send and receive message segments. The iAPX 86/30 creates and manages two queues for each mailbox. One of these queues contains message segments sent to the mailbox but not yet received by any task. The other mailbox queue consists of tasks that are waiting to receive messages. The iAPX 86/30 operation assures that waiting tasks receive messages as soon as messages are available. Thus at any moment one or possibly both of two mailbox queues will be empty.
Region	Regions are the means of serialization and mutual exclusion. Regions are familiar as "critical code regions." The iAPX 86/30 region data type consists of a queue of tasks. Each task waits to execute in mutually exclusive code or to access a shared data region, for example to update a file record.
Tokens	The OSP interface makes use of a 16-bit TOKEN data type to identify individual OSF data structures. Each of these (each instance) has its own unique TOKEN. When a primitive is called, it is passed the TOKENs of the data structures on which it will operate.

Table 3. System Data Type Codes and Attributes

S.D.T.	Code	Attributes
Jobs	1	Tasks Memory Pool S.D.T. Directory
Tasks	2	Priority Stack Code State Exception Handler
Mailboxes	3	Queue of S.D.T.s (generally segments) Queue of Tasks waiting for S.D.T.s
Region	5	Queue of Tasks waiting for mutually exclusive code or data
Segments	6	Buffer Length

Table 4. OSP Primitives

Class	OSP Primitive	Interrupt Number	Entry Code in AX	Parameters On Caller's Stack
JOB	CREATE JOB	184	0100H	*See 80130 User Manual
TASK	CREATE TASK	184	0200H	Priority, IP Ptr, Data Segment, Stack Seg, Stack Size Task Information, ExcptPtr
	DELETE TASK	184	0201H	TASK, ExcptPtr
	SUSPEND TASK	184	0202H	TASK, ExcptPtr
	RESUME TASK	184	0203H	TASK, ExcptPtr
	SET PRIORITY	184	0209H	TASK, Priority, ExcptPtr
	SLEEP	184	0204H	Time Limit,ExcptPtr
INTERRUPT	DISABLE	190	0705H	Level, ExcptPtr
	ENABLE	184	0704H	Level #, ExcptPtr
	ENTER INTERRUPT	184	0703H	Level #, ExcptPtr
	EXIT INTERRUPT	186	NONE	Level #,ExcptPtr
	GET LEVEL	188	0702H	Level #, ExcptPtr
	RESET INTERRUPT	184	0706H	Level #, ExcptPtr
	SET INTERRUPT	184	0701H	Level, Interrupt Task Flag Interrupt Handler Ptr, Interrupt Handler DataSeg ExcptPtr
	SIGNAL INTERRUPT	185	NONE	Level, ExcptPtr
	WAIT INTERRUPT	187	NONE	Level, ExcptPtr
SEGMENT	CREATE SEGMENT	184	0600H	Size, ExcptPtr
	DELETE SEGMENT	184	0603H	SEGMENT, ExceptPtr

Table 4. OSP Primitives (Continued)

Class	OSP Primitive	Interrupt Number	Entry Code in AX	Parameters On Caller's Stack
MAILBOX	CREATE MAILBOX	184	0300H	Mailbox flags, ExcptPtr
	DELETE MAILBOX	184	0301H	MAILBOX, ExcptPtr
	RECEIVE MESSAGE	184	0303H	MAILBOX, Time Limit ResponsePtr, ExcptPtr
	SEND MESSAGE	184	0302H	MAILBOX,Message Response, ExcptPtr
REGION	ACCEPT CONTROL	184	0504H	REGION, ExcptPtr
	CREATE REGION	184	0500H	Region Flags, ExcptPtr
	DELETE REGION	184	0501H	REGION, ExcptPtr
	RECEIVE CONTROL	184	0503H	REGION, ExcptPtr
	SEND CONTROL	184	0502H	ExcptPtr
ENVIRONMENTAL	DISABLE DELETION	184	0001H	TOKEN,ExcptPtr
	ENABLE DELETION	184	0002H	TOKEN,ExcptPtr
	GET EXCEPTION HANDLER	184	0800H	Ptr,ExcptPtr
	GET TYPE	184	0000H	TOKEN,ExcptPtr
	GET TASK TOKENS	184	0206H	Request, ExcptPtr
	SET EXCEPTION HANDLER	184	0801H	Ptr, ExcptPtr
	SET OS EXTENSION	184	0700H	Code,InstPtr, ExcptPtr
	SIGNAL EXCEPTION	184	0802H	Exception Code, Parameter Number, StackPtr,0,0,ExcptPtr

NOTES:
All parameters are pushed onto the OSP stack. Each parameter is one word. See Figure 3 for Call Sequence.

Explanation of the Symbols

JOB	OSP JOB SDT Token
TASK	OSP TASK SDT Token
REGION	OSP REGION SDT Token
MAILBOX	OSP MAILBOX SDT Token
SEGMENT	OSP SEGMENT SDT Token
TOKEN	Any SDT Token
Level	Interrupt Level Number
ExcptPtr	Pointer to Exception Code
Message	Message Token
Ptr	Pointer to Code,Stack etc. Address
Seg	Value Loaded into appropriate Segment Register
----	Value Parameter.

Table 5. OSP Primitive Performance Examples

Datatype Class	Primitive Execution Speed* (microseconds)	
JOB	CREATE JOB	2950
TASK	CREATE TASK (no preemption)	1360
SEGMENT	CREATE SEGMENT	700
MAILBOX	SEND MESSAGE (with task switch)	475
	SEND MESSAGE (no task switch)	265
	RECEIVE MESSAGE (task waiting)	540
	RECEIVE MESSAGE (message waiting)	260
REGION	SEND CONTROL	170
	RECEIVE CONTROL	205

*8 MHz iAPX 86/30 OSP Configuation.

Table 6. Baud Rate Count Values (16X)

Baud Rate	8 MHz Count Value	5 MHz Count Value
300	1667	1042
600	833	521
1200	417	260
2400	208	130
4800	104	65
9600	52	33

Table 7a. Mnemonic Codes for Unavoidable Exceptions

E$OK	Exception Code Value = 0 the operation was successful
E$TIME	Exception Code Value = 1 the specified time limit expired before completion of the operations was possible
E$MEM	Exception Code Value = 2 insufficient nucleus memory is available to satisfy the request
E$BUSY	Exception Code Value = 3 specified region is currently busy
E$LIMIT	Exception Code Value = 4 attempted violation of a job, semaphore, or system limit
E$CONTEXT	Exception Code Value = 5 the primitive was called in an illegal context (e.g., call to enable for an already enabled interrupt)
E$EXIST	Exception Code Value = 6 a token argument does not currently refer to any object; note that the object could have been deleted at any time by its owner
E$STATE	Exception Code Value = 7 attempted illegal state transition by a task
ENOTCONFIGURED	Exception Code Value = 8 the primitive called is not configured in this system
E$INTERRUPT$SATURATION	Exception Code Value = 9 The interrupt task on the requested level has reached its user specified saturation point for interrupt service requests. No further interrupts will be allowed on the level until the interrupt task executes a WAIT$INTERRUPT. (This error is only returned, in line, to interrupt handlers.)
E$INTERRUPT$OVERFLOW	Exception Code Value = 10 The interrupt task on the requested level previously reached its saturation point and caused an E$INTERRUPT$SATURATION condition. It subsequently executed an ENABLE allowing further interrupts to come in and has received another SIGNAL$INTERRUPTcall, bringing it over its specified saturation point for interrupt service requests. (This error is only returned, in line, to interrupt handlers).

Table 7b. Mnemonic Codes for Avoidable Exceptions

E$ZERO$DIVIDE	Exception Code Value = 8000H divide by zero interrupt occurred
E$OVERFLOW	Exception Code Value = 8001H overflow interrupt occurred
E$TYPE	Exception Code Value = 8002H a token argument referred to an object tha was not of required type
E$BOUNDS	Exception Code Value = 8003H an offset argument is out of segment bounds
E$PARAM	Exception Code Value = 8004H a (non-token,non-offset) argument has an illegal value
EBADCALL	Exception Code Value = 8005H an entry code for which there is no corresponding primitive was passed
E$ARRAY$BOUNDS = 8006H	Hardware or Language has detected an array overflow
ENDPERROR	Exception Code Value = 8007H an 8087 (Numeric data Processor) error has been detected; (the 8087 status information is contained in a parameter to the exception handler)

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bins ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Voltage on Any Pin With
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 1.0 Watts

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5 to 5.5V)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + .5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
I_{CC}	Power Supply Current		200	mA	T_A = 25 C
I_{LI}	Input Leakage Current		10	μA	$0 < V_{IN} < V_{CC}$
I_{LR}	IR Input Load Current		10 −300	μA μA	$V_{IN} = V_{CC}$ $V_{IN} = 0$
I_{LO}	Output Leakage Current		10	μA	$.45 = V_{IN} = V_{CC}$
V_{CLI}	Clock Input Low		0.6	V	
V_{CHI}	Clock Input High	3.9		V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		15	pF	
I_{CLI}	Clock Input Leakage Current		10 150 10	μA μA μA	$V_{IN} = V_{CC}$ V_{IN} = 2.5V V_{IN} = 0V

A.C. CHARACTERISTICS (T_A = 0-70°C, V_{CC} = 4.5-5.5 Volt, V_{SS} = Ground)

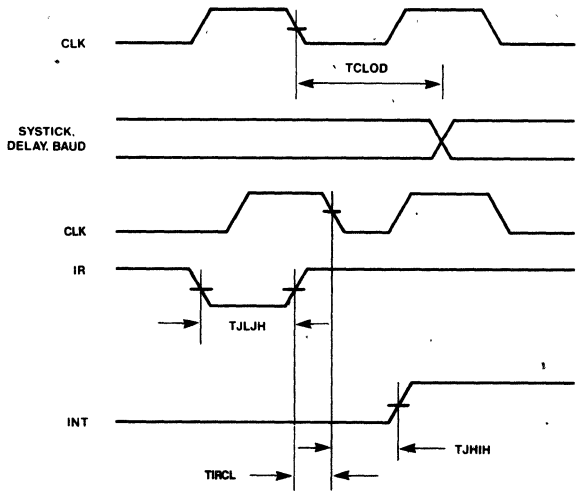
		80130		80130-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
T_{CLCL}	CLK Cycle Period	200	–	125	–	ns	
T_{CLCH}	CLK Low Time	90	–	55	–	ns	
T_{CHCL}	CLK High Time	69	2000	44	2000	ns	
T_{SVCH}	Status Active Setup Time	80	–	65	–	ns	
T_{CHSV}	Status Inactive Hold Time	10	–	10	–	ns	
T_{SHCL}	Status Inactive Setup Time	55	–	55	–	ns	
T_{CLSH}	Status Active Hold Time	10	–	10	–	ns	
T_{ASCH}	Address Valid Setup Time	8	–	8	–	ns	
T_{CLAH}	Address Hold Time	10	–	10	–	ns	
T_{CSCL}	Chip Select Setup Time	20	–	20	–	ns	
T_{CHCS}	Chip Select Hold Time	0	–	0	–	ns	
T_{DSCL}	Write Data Setup Time	80	–	60	–	ns	
T_{CHDH}	Write Data Hold Time	10	–	10	–	ns	
T_{JLJH}	IR Low Time	100	–	100	–	ns	
T_{CLDV}	Read Data Valid Delay	–	140	–	105	ns	C_L = 200 pE
T_{CLDH}	Read Data Hold Time	10	–	10	–	ns	
T_{CLDX}	Read Data to Floating	10	100	10	100	ns	
T_{CLCA}	Cascade Address Delay Time	–	85	–	65	ns	

A.C. CHARACTERISTICS (Continued)

		80130		80130-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Notes**
T_{CLCF}	Cascade Addresse Hold Time	10	–	10	–	ns	
T_{IAVE}	INTA Status t Acknowledge	–	80	–	80	ns	
T_{CHEH}	Acknowledge Hold Time	0	–	0	–	ns	
T_{CSAK}	Chip Select to ACK	–	110	–	110	ns	
T_{SACK}	Status to ACK	–	140	–	140	ns	
T_{AACK}	Address to ACK	–	90	–	90	ns	
T_{CLOD}	Timer Output Delay Time	–	200	–	200	ns	C_L = 100 pF
T_{CLOD1}	Timer1 Output Delay Time	–	200	–	200	ns	C_L = 100 pF
T_{JHIH}	INT Output Delay	–	200	–	200	ns	
T_{IRCL}	IR Input Set Up	20				ns	

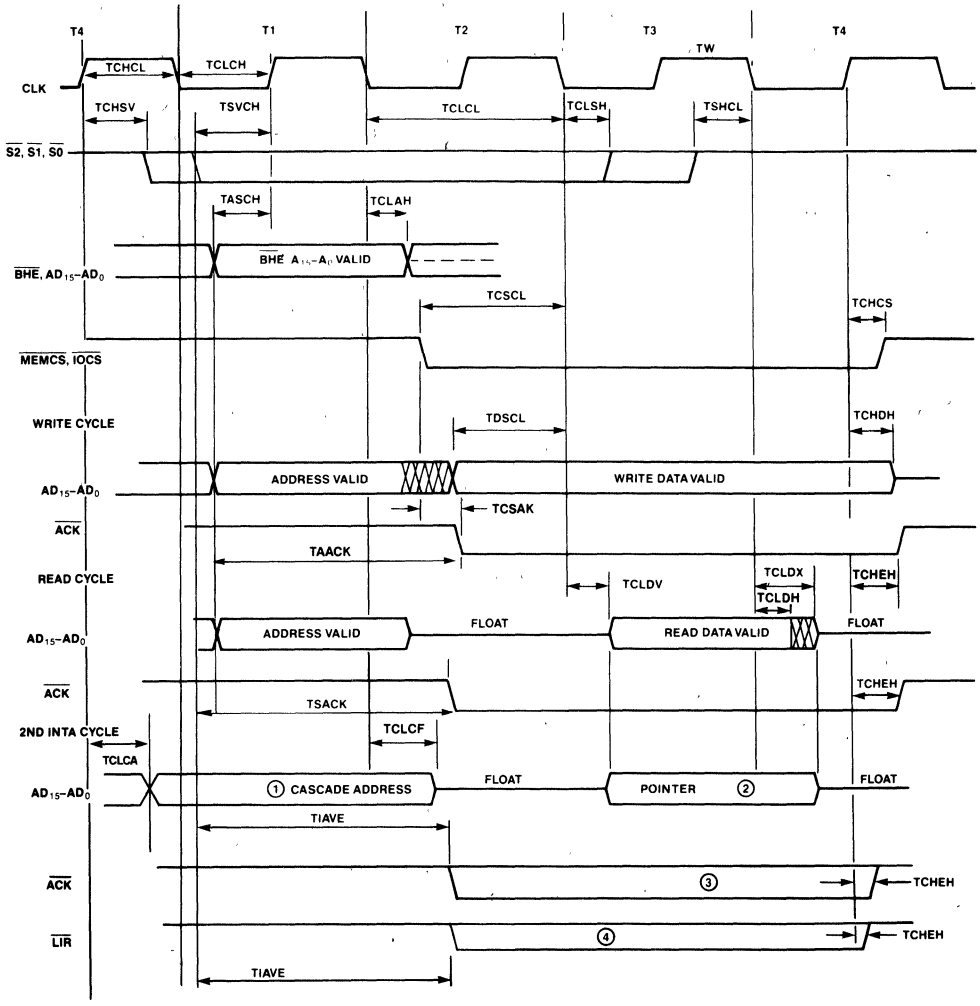
WAVEFORMS

A.C.



WAVEFORMS

A.C.



NOTES
1 CASCADE ADDRESS PRESENTED ON AD8, AD9 AND AD10 CORRESPONDING TO CAS0, CAS1 AND CAS2 RESPECTIVELY AD11-AD15 LINES ARE ACTIVE AND HAVE UNKNOWN VALUES AD0-AD7 ARE TRISTATE
2 POINTER VALUE IS ACTIVE ONLY IF POINTER IS GENERATED FROM THE 80150 AND NOT FROM EXTERNAL SLAVE UNIT
3 ACTIVE LOW ONLY WHEN POINTER DATA IS BEING SUPPLIED BY THE 80150
4 LOW ONLY FOR LOCAL INTERRUPT

80150/80150-2 ADVANCE INFORMATION

iAPX 86/50, 88/50, 186/50, 188/50 CP/M-86* OPERATING SYSTEM PROCESSORS

- **High-Performance Two-Chip Data Processors Containing the Complete CP/M-86 Operating System**
- **Standard On-Chip BIOS (Basic Input/Output System) Contains Drivers for 8272A, 8274, 8255A, 8251A**
- **BIOS Extensible with User-Supplied Peripheral Drivers**
- **User Intervention Points Allow Addition of New System Commands**
- **Memory Disk Makes Possible Diskless CP/M-86 Systems**
- **No License or Serialization Required**
- **Built-in Operating System Timers and Interrupt Controller**
- **8086/80150/80150-2/8088/80186/80188 Compatible At Up To 8 MHz Without Wait States**

The Intel iAPX 86/50, 88/50, 186/50, and 188/50 are two-chip microprocessors offering general-purpose CPU instructions combined with the CP/M-86 operating system. Respectively, they consist of the 8- and 16-bit software compatible 8086, 8088, 80186, and 80188 CPU plus the 80150 CP/M-86 operating system extension.

CP/M-86 is a single-user operating system designed for computers based on the Intel iAPX 86, 88, 186, and 188 microprocessors. The system allows full utilization of the one megabyte of memory available for application programs. The 80150 stores CP/M-86 in its 16K bytes of on-chip memory. The 80150 will run third-party applications software written to run under standard Digital Research CP/M-86.

The 80150 is implemented in N-Channel, depletion-load, silicon-gate technology (HMOS), and is housed in a 40-pin package. Included on the 80150 are the CP/M-86 operating system, Version 1.1, plus hardware support for eight interrupts, a system timer, a delay timer, and a baud rate generator.

*CP/M-86 is a trademark of Digital Research, Inc

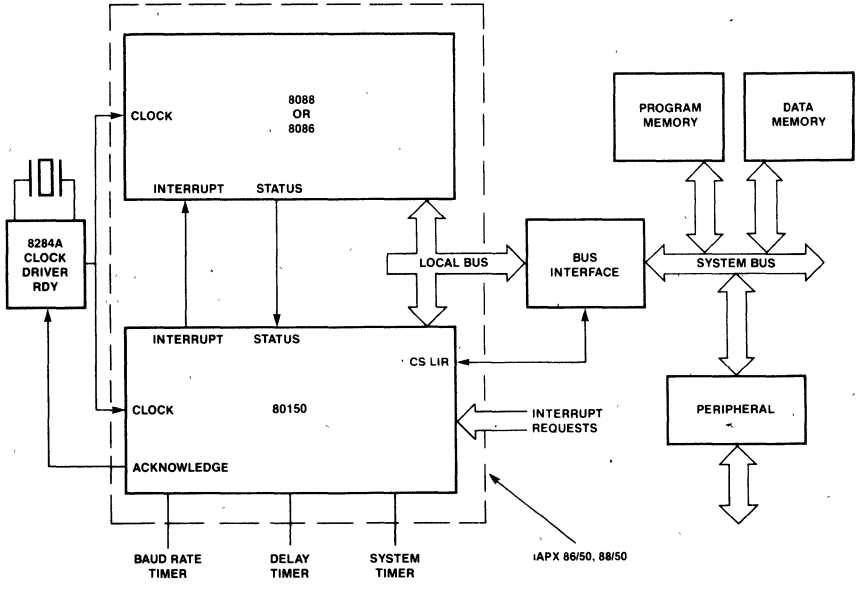


Figure 1. iAPX 86/50, 88/50 Block Diagram

SEPTEMBER 1982
ORDER NUMBER: 210705-002

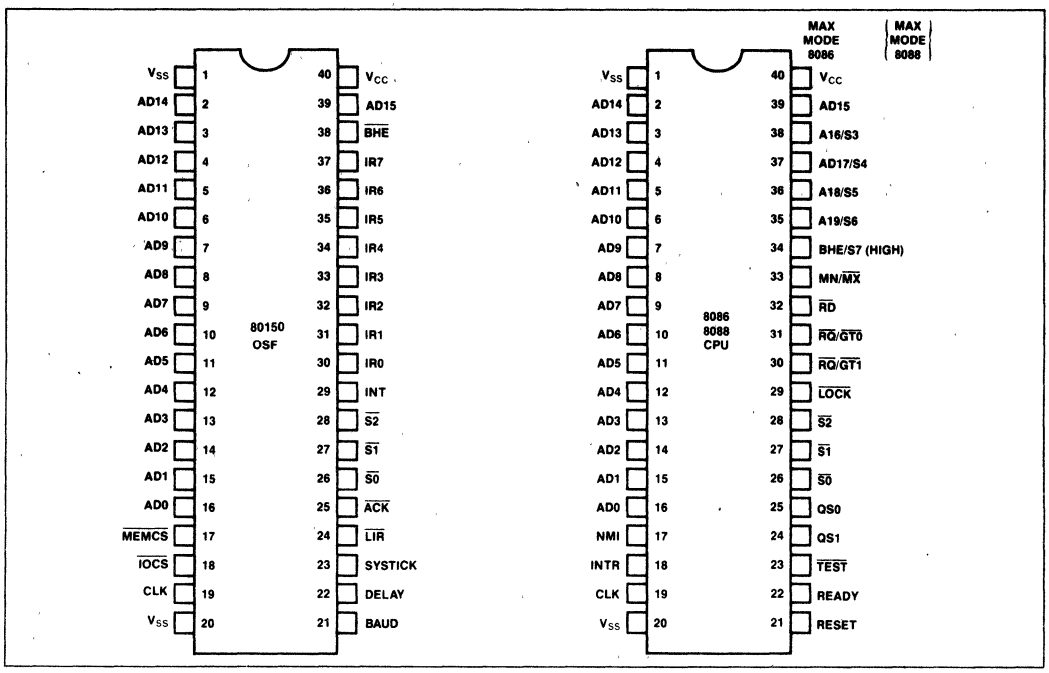


Figure 2. iAPX 86/50, 88/50 Pin Configuration

Table 1. 80150 Pin Description

Symbol	Type	Name and Function
AD_{15}–AD_0	I/O	**Address Data:** These pins constitute the time multiplexed memory address (T_1) and data (T_2, T_3, T_W, T_4) bus. These lines are active HIGH. The address presented during T_1 of a bus cycle will be latched internally and interpreted as an 80150 internal address if MEMCS or IOCS is active for the invoked primitives. The 80150 pins float whenever it is not chip selected, and drive these pins only during $T_2 - T_4$ of a read cycle and T_1 of an INTA cycle.
$\overline{BHE}/S_7$	I	**Bus High Enable:** The 80150 uses the BHE signal from the processor to determine whether to respond with data on the upper or lower data pins, or both. The signal is active LOW. BHE is latched by the 80150 on the trailing edge of ALE. It controls the 80150 output data as shown. $\overline{BHE}$ A_0 0 0 Word on AD_{15}–AD_0 0 1 Upper byte on $AD_{15} - AD_8$ 1 0 Lower byte on AD_7–AD_0 1 1 Upper byte on AD_7–AD_0
$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$	I	**Status:** For the 80150, the status pins are used as inputs only. 80150 encoding follows: $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$ 0 0 0 INTA 0 0 1 IORD 0 1 0 IOWR 0 1 1 Passive 1 0 0 Instruction fetch 1 0 1 MEMRD 1 1 X Passive

AFN-01467A

Table 1. 80150 Pin Description (Continued)

Symbol	Type	Name and Function
CLK	I	**Clock:** The system clock provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing. The 80130 uses the system clock as an input to the SYSTICK and BAUD timers and to synchronize operation with the host CPU
INT	O	**Interrupt:** INT is HIGH whenever a valid interrupt request is asserted. It is normally used to interrupt the CPU by connecting it to INTR.
IR_7–IR_0	I	**Interrupt Requests:** An interrupt request can be generated by raising an IR input (LOW to HIGH) and holding it HIGH until it is acknowledged (Edge-Triggered Mode), or just by a HIGH level on an IR input (Level-Triggered Mode).
$\overline{ACK}$	O	**Acknowledge:** This line is LOW whenever an 80150 resource is being accessed. It is also LOW during the first INTA cycle and second INTA cycle if the 80150 is supplying the interrupt vector information. This signal can be used as a bus ready acknowledgement and/or bus transceiver control.
$\overline{MEMCS}$	I	**Memory Chip Select:** This input must be driven LOW when a kernel primitive is being fetched by the CPU AD_{13}–AD_0 are used to select the instruction.
$\overline{IOCS}$	I	**Input/Output Chip Select:** When this input is low, during an IORD or IOWR cycle, the 80150's kernel primitives are accessing the appropriate peripheral function as specified by the following table: $\overline{BHE}$ A_3 A_2 A_1 A_0 0 X X X X Passive X X X X 1 Passive X 0 1 X X Passive 1 0 0 X 0 Interrupt Controller 1 1 0 0 0 Systick Timer 1 1 0 1 0 Delay Counter 1 1 1 0 0 Baud Rate Timer 1 1 1 1 0 Timer Control
$\overline{LIR}$	O	**Local Bus Interrupt Request:** This signal is LOW when the interrupt request is for a non-slave input or slave input programmed as being a local slave.
V_{CC}		**Power:** V_{CC} is the +5V supply pin.
V_{SS}		**Ground:** V_{SS} is the ground pin
SYSTICK	O	**System Clock Tick:** Timer 0 Output.
DELAY	O	**DELAY Timer:** Output of timer 1
BAUD	O	**Baud Rate Generator:** 8254 Mode 3 compatible output. Output of 80150 Timer 2.

The 80150 breaks new ground in operating system software-on-silicon components. It is unique because it is the first time that an industry-standard personal/small business computer operating system is being put in silicon. The 80150 contains Digital Research's CP/M-86 operating system, which is designed for Intel's line of software- and interface-compatible iAPX 86, 88, 186, and 188 microprocessors. Since the entire CP/M-86 operating system is contained on the chip, it is now possible to design a diskless computer that runs proven and commonly available applications software. The 80150 is a true operating system extension to the host microprocessor, since it also integrates key operating system-related peripheral functions onto the chip.

MODULAR DESIGN

Based on a proven, modular design, the system includes the:

- CCP: Console Command Processor

  The CCP is the human interface to the operating system and performs decoding and

execution of user commands.

- BDOS: Basic Disk Operating System

  The BDOS is the logical, invariant portion of the operating system; it supports a named file system with a maximum of 16 logical drives, containing up to 8 megabytes each for a potential of 128 megabytes of on-line storage.

- BIOS: Basic Input/Output System

  The physical, variant portion of the operating system, the BIOS contains the system-dependent input/output device handlers.

CP/M* COMPATIBILITY

CP/M-86 files are completely compatible with CP/M for 8080- and 8085-based microcomputer systems. This simplifies the conversion of software developed under CP/M to take full advantage of iAPX 86, 88, 186, 188-based systems.

The user will notice no significant difference between CP/M and CP/M-86. Commands such as DIR, TYPE, REN, and ERA respond the same way in both systems.

CP/M-86 uses the iAPX 86, 88, 186, 188 registers corresponding to 8080 registers for system call and return parameters to further simplify software transport. The 80150 allows application code and data segments to overlap, making the mixture of code and data that often appears in CP/M applications acceptable to the iAPX 86, 88, 186, 188.

Unique Capabilities of CP/M-86 in Silicon

1. CP/M-86 on-a-chip reduces software development required by the system designer. It can change the implementation of the operating system into the simple inclusion of the 80150 on the CPU board.

   As described later, the designer can either simply incorporate the Intel chip without the need for writing even a single line of additional code, or he can add additional device drivers by writing only the small amount of additional code required.

2. The 80150 is the most cost-effective way to implement CP/M-86 in a microcomputer. The integration of CP/M-86 with the 16K bytes of system memory it requires, the two boot ROMS required in a diskette-based CP/M-86, and the on-chip peripherals (interrupt controller and timers) lead to savings in software, parts cost, board space, and interconnect wiring.

3. The reliability of the microcomputer is increased significantly. Since CP/M-86 is now always in the system as a standard hardware operating system, a properly functioning system diskette is not required. CP/M-86 in hardware can no longer be overwritten accidentally by a runaway program. System reliability is enhanced by the decreased dependence on floppy disks and fewer chips and interconnections required by the highly integrated 80150.

4. The microcomputer system boots up CP/M-86 on power-on, rather than requiring the user to go through a complicated boot sequence, thus lowering the user expertise required.

5. Diskless CP/M-based systems are now easy to design. Since CP/M is already in the microcomputer hardware, there is no need for a disk drive in the system if it is not desired. Without a disk drive, a system is more portable, simpler to use, less costly, and more reliable.

6. The administrative costs associated with distributing CP/M-86 are eliminated. Since CP/M-86 is now resident on the 80150 in the microcomputer system, there is no end-user licensing required nor is there any serialization requirement for the 80150 (because no CP/M diskette is used).

7. End-users will value having their CP/M operating system resident in their computer rather than on a diskette. They will no longer have to back up the operating system or have a diskette working properly to bring the system up in CP/M, increasing their confidence in the integrity, reliability, and usability of the system.

80150 FUNCTIONAL DESCRIPTION

The 80150 is a processor extension that is fully compatible with the 8086, 8088, 80186, and 80188 microprocessors. When the 80150 is combined with the microprocessor, the two-chip set is called an Operating System Processor and is denoted as the iAPX 86/50, 88/50, 186/50, or 188/50. The basic system configuration is shown in Figure 1. The 80150 connects directly to the multiplexed address/data bus and runs up to 8 MHz without wait states.

A. Hardware. Figure 3 is a functional diagram of the 80150 itself. CP/M-86 is stored in the 16K-bytes of control store. The timers are compatible with the standard 8254 timer. The interrupt controller, with its eight programmable interrupt inputs and one interrupt output, is compatible with the 8259A Programmable Interrupt Controller. External slave 8259A inter-

*CP/M is a registered trademark of Digital Research, Inc

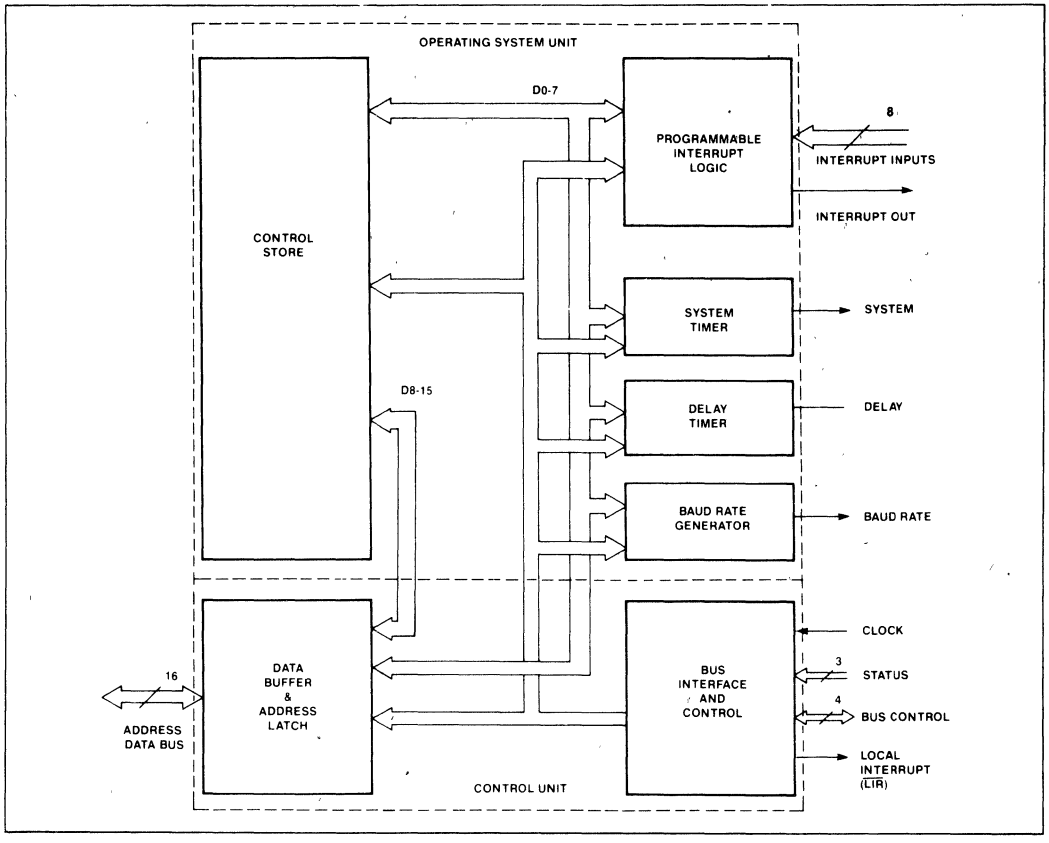


Figure 3. 80150 Internal Block Diagram

rupt controllers can be cascaded with the 80150 to expand the total number of interrupts to 57.

B. Software. Digital Research's version 1.1 of CP/M-86 forms the basis of the 80150. CP/M consists of three major parts: the Console Command Processor (CCP), the Basic Disk Operating System (BDOS), and the Basic Input/Output System (BIOS). Details on CP/M-86 are provided in Digital Research's *CP/M-86 Operating System User's Guide* and *CP/M-86 Operating System System Guide.*

CCP - Console Command Processor

The CCP provides all of the capabilities provided by Digital Research's CCP. Built-in commands have been expanded to include capabilities normally included as transient utilities on the Digital Research CP/M-86 diskette. Commands are provided to format diskettes, transfer files between devices (based on Digital Research's Peripheral Interchange Program PIP), and alter and display I/O device and file status (based on Digital Research's STAT).

Through User Intervention Points, the standard CP/M-86 CCP is enhanced to allow the user to add new built-in commands to further customize a CP/M-86 system.

BDOS - Basic Disk Operating System

Once the CCP has parsed a command, it sends it to the BDOS, which performs system services such as managing disk directories and files. Some of the standard BDOS functions provide:

Console Status
Console Input and Output
List Output
Select Drive
Set Track and Sector

Read/Write Sector
Load Program

The BDOS in the 80150 provides the same functions as the standard Digital Research CP/M-86 BDOS.

BIOS - Basic Input/Output System

The BIOS contains the system-dependent I/O drivers. The 80150 BIOS offers two fundamental configuration options:

1. **A predefined configuration** which supports minimum cost CP/M-86 microcomputer systems and which requires no operating system development by the system designer.
2. **An OEM-configurable mode,** where the designer can choose among several drivers offered on the 80150 or substitute or add any additional device drivers of his choice.

These two options negate the potential software-on-silicon pitfall of inflexibility in system design. The OEM can customize the end system as desired.

The **predefined configuration** offers a choice among several peripheral chip drivers included on the 80150. Drivers for the following chips are included in the 80150 BIOS:

8251A	Universal Synchronous/ Asynchronous Receiver/Transmitter (USART)
8274	Multi-Protocol Serial Controller (MPSC)
8255A	Programmable Parallel Interface (PPI)
8272A	Floppy Disk Controller

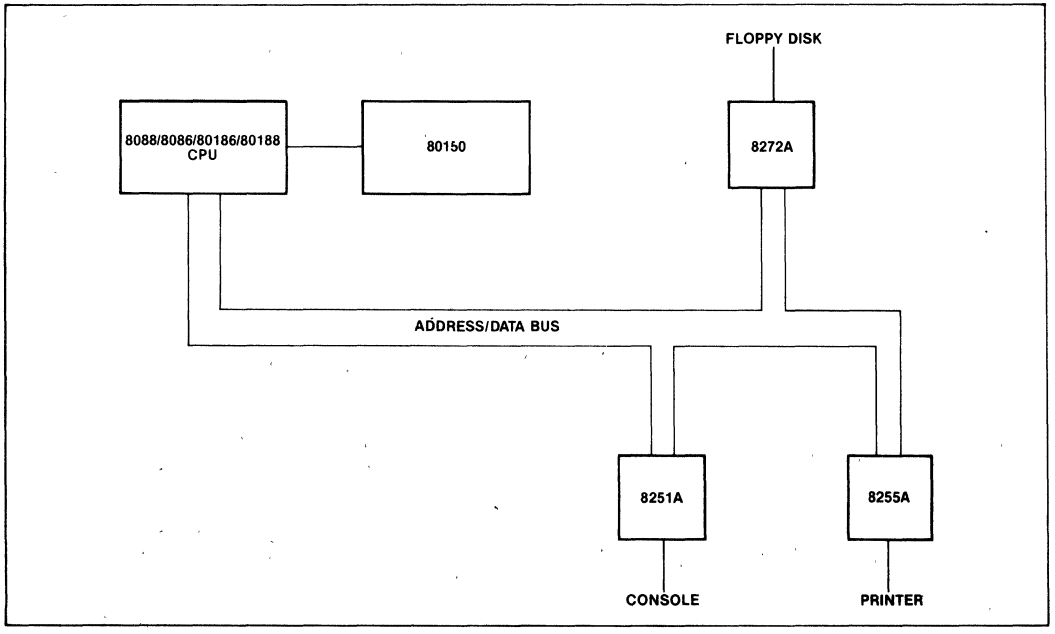


Figure 4. Predefined Configuration

Even in the predefined configuration, the system designer (or end user, if the system designer desires) may select parameters such as the baud rates for the console and printer, and the floppy disk size (standard 8″ or 5¼″ mini-floppy) and format (FM single density or MFM double density, single-sided or double-sided).

Drivers for the 80150 on-chip timers and interrupt controller are also included in the BIOS.

The 80150 takes advantage of the 80186 and 80188 on-chip peripherals in an iAPX 186/50 or 188/50 system. For example, the integrated DMA controller is used. Also fully utilized are the integrated memory chip selects and I/O chip selects.

Since all microcomputer configurations cannot be anticipated, the **OEM-configurable mode** allows the system designer to use any set of peripheral chips desired. This configuration is shown in Figure 5.

By simply changing the jump addresses in a configuration table, the designer can also gain the flexibility of adding custom BIOS drivers for other peripheral chips, such as bubble memories or more complex CRT controllers. These drivers would be stored in memory external to the 80150 itself. By providing the configurability option, the 80150 is applicable to a far broader range of designs that it would be with an inflexible BIOS.

MEMORY ORGANIZATION

When using the **predefined configuration** of the 80150 BIOS, the 80150 must be placed in the top 16K of the address space of the microprocessor (starting at location FC000H) so that the 80150 gains control when the microprocessor is reset. Upon receipt of control, the 80150 writes a **configuration block** into the bottom of the microprocessor's address space, which must be in RAM. The 80150 uses the area after the interrupt vectors for system configuration information and scratch-pad storage.

When using the **OEM-configurable** mode of the 80150 BIOS, the 80150 is placed on any 16K boun-

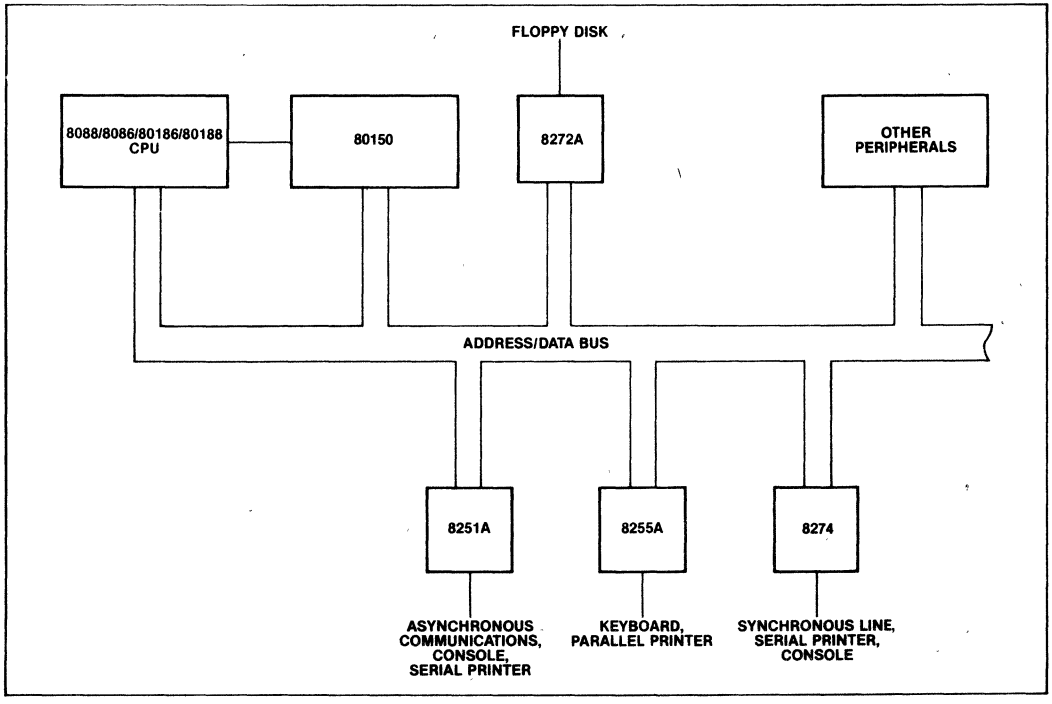


Figure 5. OEM Configurable System

dary of memory **except** the highest (FC000H) or lowest (00000H). The user writes interface code (in the form of a simple boot ROM) to incorporate and link additional features and changes into the standard 80150 environment. The configuration block may be located as desired in the address space, and its size may vary widely depending on the application.

Memory Disk

A unique capability offered by the 80150 is the Memory Disk. The Memory Disk consists of a block of RAM whose size can be selected by the designer. The Memory Disk is treated by the BDOS as any standard floppy disk, and is one of the 16 disks that CP/M can address. Thus files can be opened and closed, programs stored, and statistics gathered on the amount of Memory Disk space left.

The Memory Disk opens the possibility of a portable low-cost diskless microcomputer or network station. Applications software can be provided in a number of ways:

a. telephone lines via a modem.
b. ROM-based software.
c. a network.
d. bubble memory based software.
e. low-cost cassettes.

TYPICAL SYSTEM CONFIGURATION

Figure 6 shows the processing cluster of a "typical" iAPX 86/50 or iAPX 88/50 OSP system. Not shown are subsystems likely to vary with the application. The configuration includes an 8086 (or 8088) operating in maximum mode, an 8284A clock generator and an 8288 system controller. Note that the 80150 is located on the CPU side of any latches or transceivers.

Timers

The Timers are connected to the lower half of the data bus and are addressed at even addresses. The timers are read as two successive bytes,

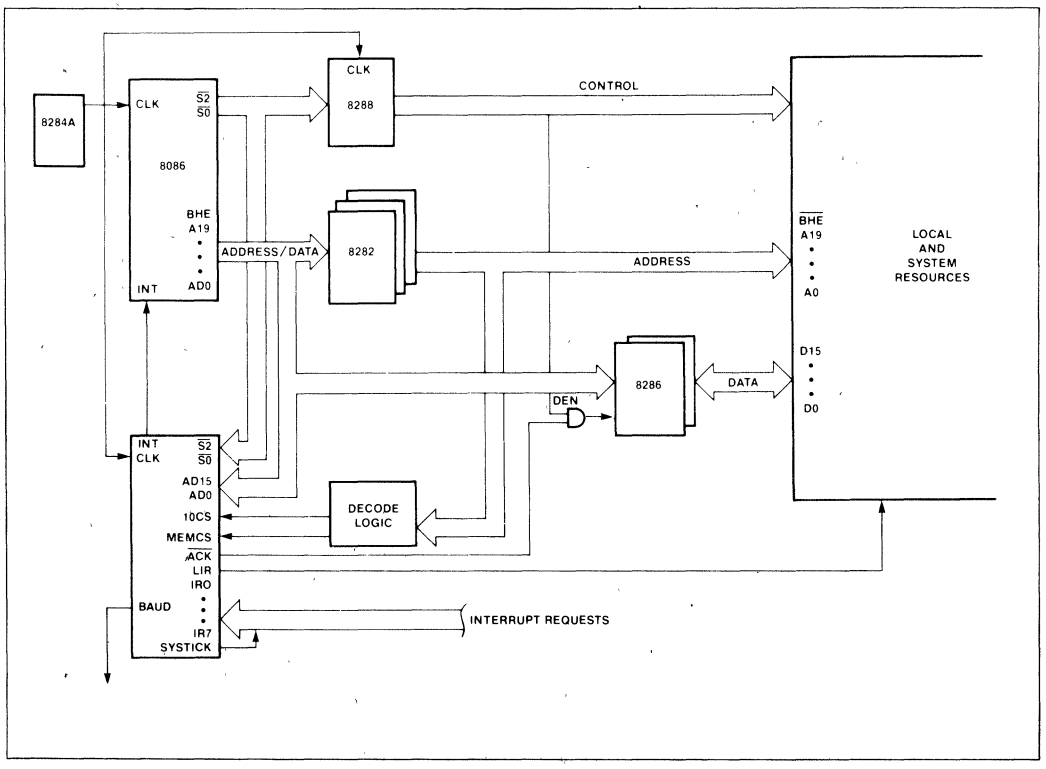


Figure 6. Typical OSP Configuration

always LSB followed by MSB. The MSB is always latched on a read operation and remains latched until read. Timers are not gatable. An external 8254 Programmable Interval Timer may be added to the system.

Baud Rate Generator

The baud rate generator operates like an 8254 (square wave mode 3). Its output, BAUD, is initially high and remains high until the Count Register is loaded. The first falling edge of the clock after the Count Register is loaded causes the transfer of the internal counter to the Count Register. The output stays high for N/2 [(N + 1)/2 if N is odd] and then goes low for N/2 [(N – 1)/2 if N is odd]. On the falling edge of the clock which signifies the final count for the output in low state, the output returns to high state and the Count Register is transferred to the internal counter. The baud rates can vary from 300 to 9600 baud.

The baud rate generator is located at 0CH (12), relative to the 16-byte boundary in the I/O space in which the 80150 component is located. The timer control word is located at relative address, 0EH(14). Timers are addressed with IOCS = 0. Timers 0 and 1 are assigned to use by the OSP, and should not be altered by the user.

The 80150 timers are subset compatible with 8254 timers.

Interrupt Controller

The Programmable Interrupt Controller (PIC), is also an integral unit of the 80150. Its eight input pins handle eight vectored priority interrupts. One of these pins must be used for the SYSTICK time function in timing waits, using an external connection as shown. During the 80150 initialization and configuration sequence, each 80150 interrupt pin is individually programmed as either level or edge sensitive. External slave 8259A interrupt controllers can be used to expand the total number of interrupts to 57.

In addition to standard PIC functions, the 80150 PIC unit has an $\overline{LIR}$ output signal, which when low indicates an interrupt acknowledge cycle. $\overline{LIR} = 0$ is provided to control the 8289 Bus Arbiter SYSB/$\overline{RESB}$ pin. This will avoid the need of requesting the system bus to acknowledge local bus non-slave interrupts. The user defines the interrupt system as part of the configuration.

INTERRUPT SEQUENCE

The interrupt sequence is as follows:

1. One or more of the interrupts is set by a low-to-high transition on edge-sensitive IR inputs or by a high input on level-sensitive IR inputs.
2. The 80150 evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an interrupt acknowledge cycle which is encoded in $\overline{S_2} - \overline{S_0}$.
4. Upon receiving the first interrupt acknowledge from the CPU, the highest-priority interrupt is set by the 80150 and the corresponding edge detect latch is reset. The 80150 does not drive the address/data bus during this bus cycle but does acknowledge the cycle by making $\overline{ACK} = 0$ and sending the $\overline{LIR}$ value for the IR input being acknowledged.
5. The CPU will then initiate a second interrupt acknowledge cycle. During this cycle, the 80150 will supply the cascade address of the interrupting input at T_1 on the bus and also release an 8-bit pointer onto the bus if appropriate, where it is read by the CPU. If the 80150 does supply the pointer, then $\overline{ACK}$ will be low for the cycle. This cycle also has the value $\overline{LIR}$ for the IR input being acknowledged.
6. This completes the interrupt cycle. The ISR bit remains set until an appropriate EXIT INTERRUPT primitive (EOI command) is called at the end of the Interrupt Handler.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to 150°C
Voltage on Any Pin With
Respect to Ground ..... − 1.0V to + 7V
Power Dissipation ..... . 1 0 Watts

NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended period may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5 to 5.5V)

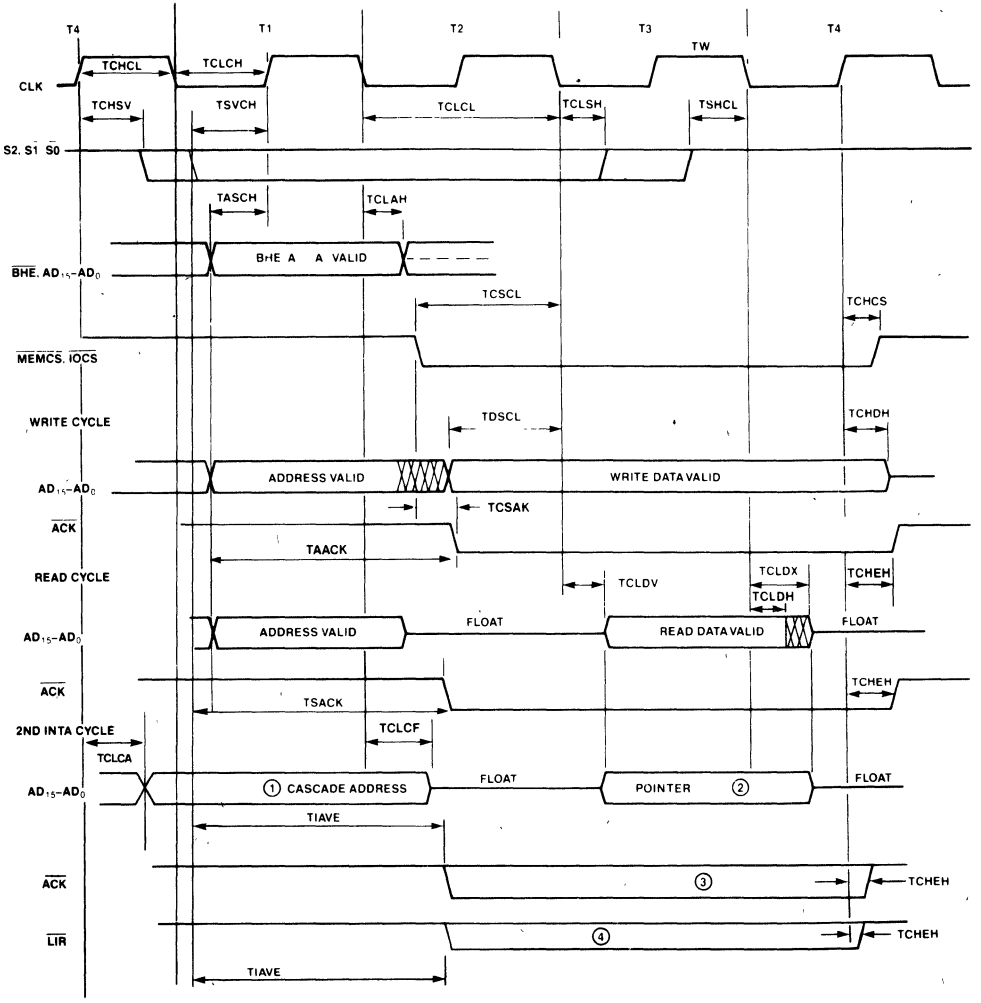
Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	− 0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + .5	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −400μA
I_{CC}	Power Supply Current		200	mA	T_A = 25 C
I_{LI}	Input Leakage Current		10	μA	$0 < V_{IN} < V_{CC}$
I_{LR}	IR Input Load Current		10 −300	μA μA	$V_{IN} = V_{CC}$ $V_{IN} = 0$
I_{LO}	Output Leakage Current		10	μA	$.45 \le V_{IN} \le V_{CC}$
V_{CLI}	Clock Input Low		0.6	V	
V_{CHI}	Clock Input High	3.9		V	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		15	pF	
I_{CLI}	Clock Input Leakage Current		10 150 10	μA μA μA	$V_{IN} = V_{CC}$ V_{IN} = 2.5V V_{IN} = 0V

A.C. CHARACTERISTICS (T_A = 0-70°C, V_{CC} = 4.5-5.5 Volt, V_{SS} = Ground)

		80150		80150-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
T_{CLCL}	CLK Cycle Period	200	–	125	–	ns	
T_{CLCH}	CLK Low Time	90	–	55	–	ns	
T_{CHCL}	CLK High Time	69	2000	44	2000	ns	
T_{SVCH}	Status Active Setup Time	80	–	65	–	ns	
T_{CHSV}	Status Inactive Hold Time	10	–	10	–	ns	
T_{SHCL}	Status Inactive Setup Time	55	–	55	–	ns	
T_{CLSH}	Status Active Hold Time	10	–	10	–	ns	
T_{ASCH}	Address Valid Setup Time	8	–	8	–	ns	
T_{CLAH}	Address Hold Time	10	–	10	–	ns	
T_{CSCL}	Chip Select Setup Time	20	–	20	–	ns	
T_{CHCS}	Chip Select Hold Time	0	–	0	–	ns	
T_{DSCL}	Write Data Setup Time	80	–	60	–	ns	
T_{CHDH}	Write Data Hold Time	10	–	10	–	ns	
T_{JLJH}	IR Low Time	100	–	100	–	ns	
T_{CLDV}	Read Data Valid Delay	–	140	–	105	ns	C_L = 200 pF
T_{CLDH}	Read Data Hold Time	10	–	10	–	ns	
T_{CLDX}	Read Data to Floating	10	100	10	100	ns	
T_{CLCA}	Cascade Address Delay Time	–	85	–	65	ns	

WAVEFORMS

A.C.



NOTES
1. CASCADE ADDRESS PRESENTED ON AD8, AD9 AND AD10 CORRESPONDING TO CAS0, CAS1 AND CAS2 RESPECTIVELY AD11-AD15 LINES ARE ACTIVE AND HAVE UNKNOWN VALUES AD0-AD7 ARE TRISTATE
2. POINTER VALUE IS ACTIVE ONLY IF POINTER IS GENERATED FROM THE 80150 AND NOT FROM EXTERNAL SLAVE UNIT
3. ACTIVE LOW ONLY WHEN POINTER DATA IS BEING SUPPLIED BY THE 80150
4. LOW ONLY FOR LOCAL INTERRUPT

8282/8283
OCTAL LATCH

- **Address Latch for iAPX 86, 88, 186, 188, MCS-80®, MCS-85®, MCS-48® Famlies**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Data Register and Buffer**
- **Transparent during Active Strobe**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 8282 and 8283 are 8-bit bipolar latches with 3-state output buffers. They can be used to implement latches, buffers, or multiplexers. The 8283 inverts the input data at its outputs while the 8282 does not. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with these devices.

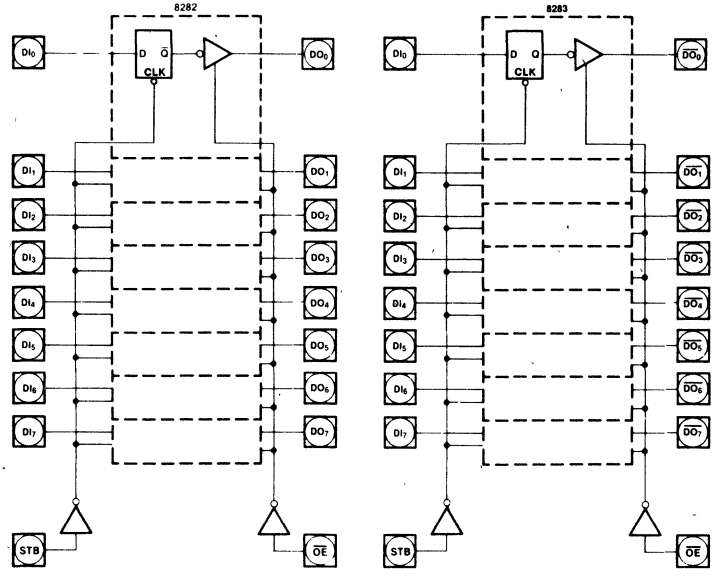


Figure 1. Logic Diagrams

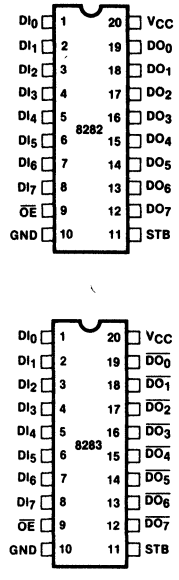


Figure 2. Pin Configurations

Table 1. Pin Description

Pin	Description
STB	STROBE (Input). STB is an input control pulse used to strobe data at the data input pins (A_0-A_7) into the data latches. This signal is active HIGH to admit input data. The data is latched at the HIGH to LOW transition of STB.
$\overline{OE}$	OUTPUT ENABLE (Input). $\overline{OE}$ is an input control signal which when active LOW enables the contents of the data latches onto the data output pin (B_0-B_7). OE being inactive HIGH forces the output buffers to their high impedance state.
DI_0-DI_7	DATA INPUT PINS (Input). Data presented at these pins satisfying setup time requirements when STB is strobed and latched into the data input latches.
DO_0-DO_7 (8282) $\overline{DO_0}$-$\overline{DO_7}$ (8283)	DATA OUTPUT PINS (Output). When $\overline{OE}$ is true, the data in the data latches is presented as inverted (8283) or non-inverted (8282) data onto the data output pins.

FUNCTIONAL DESCRIPTION

The 8282 and 8283 octal latches are 8-bit latches with 3-state output buffers. Data having satisfied the setup time requirements is latched into the data latches by strobing the STB line HIGH to LOW. Holding the STB line in its active HIGH state makes the latches appear transparent. Data is presented to the data output pins by activating the $\overline{OE}$ input line. When $\overline{OE}$ is inactive HIGH the output buffers are in their high impedance state. Enabling or disabling the output buffers will not cause negative-going transients to appear on the data output bus.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias.....0°C to 70°C
Storage Temperature.....−65°C to +150°C
All Output and Supply Voltages.....−0.5V to +7V
All Input Voltages.....−1.0V to +5.5V
Power Dissipation.....1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_C	Input Clamp Voltage		−1	V	I_C = −5 mA
I_{CC}	Power Supply Current		160	mA	
I_F	Forward Input Current		−0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage		.45	V	I_{OL} = 32 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −5 mA
I_{OFF}	Output Off Current		±50	μA	V_{OFF} = 0.45 to 5.25V
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

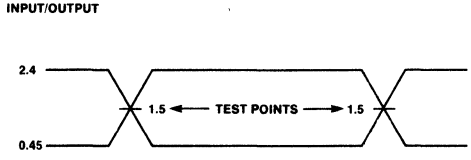
NOTE:
1. Output Loading I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF.*

A.C. CHARACTERISTICS (V_{CC} = 5V ±10%, T_A = 0°C to 70°C (See Note 2)
Loading: Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
TIVOV	Input to Output Delay				(See Note 1)
	—Inverting	5	22	ns	
	—Non-Inverting	5	30	ns	
TSHOV	STB to Output Delay				
	—Inverting	10	40	ns	
	—Non-Inverting	10	45	ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TIVSL	Input to STB Setup Time	0		ns	
TSLIX	Input to STB Hold Time	25		ns	
TSHSL	STB High Time	15		ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 0.8V

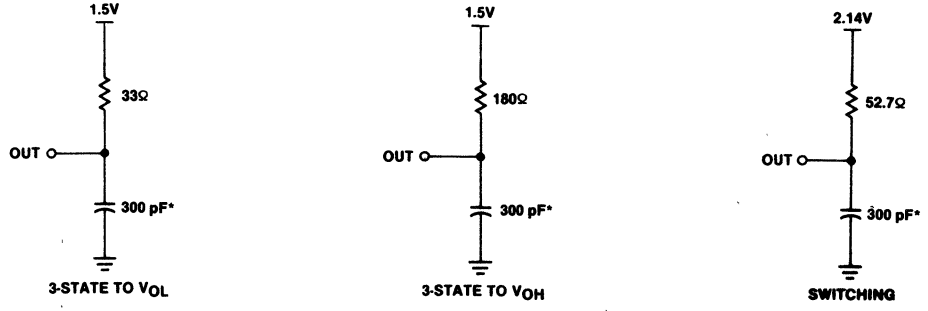
NOTE:
*C_L = 200 pF for plastic 8282/8283.
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TSHOV = 45, 55; TEHOZ = 25; TELOV = 50.

A.C. TESTING INPUT, OUTPUT WAVEFORM



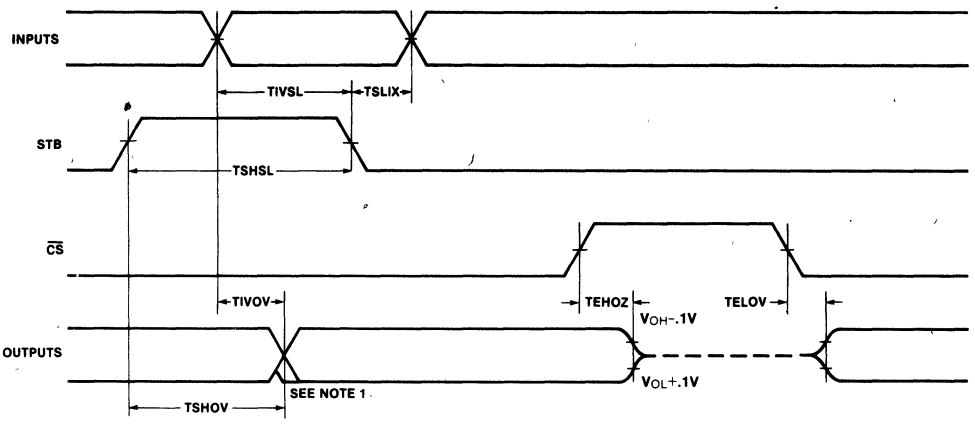
A.C. TESTING. INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE MEASURED FROM 0.8V TO 2.0V AND ARE DRIVEN AT 5ns ± 2ns

OUTPUT TEST LOAD CIRCUITS

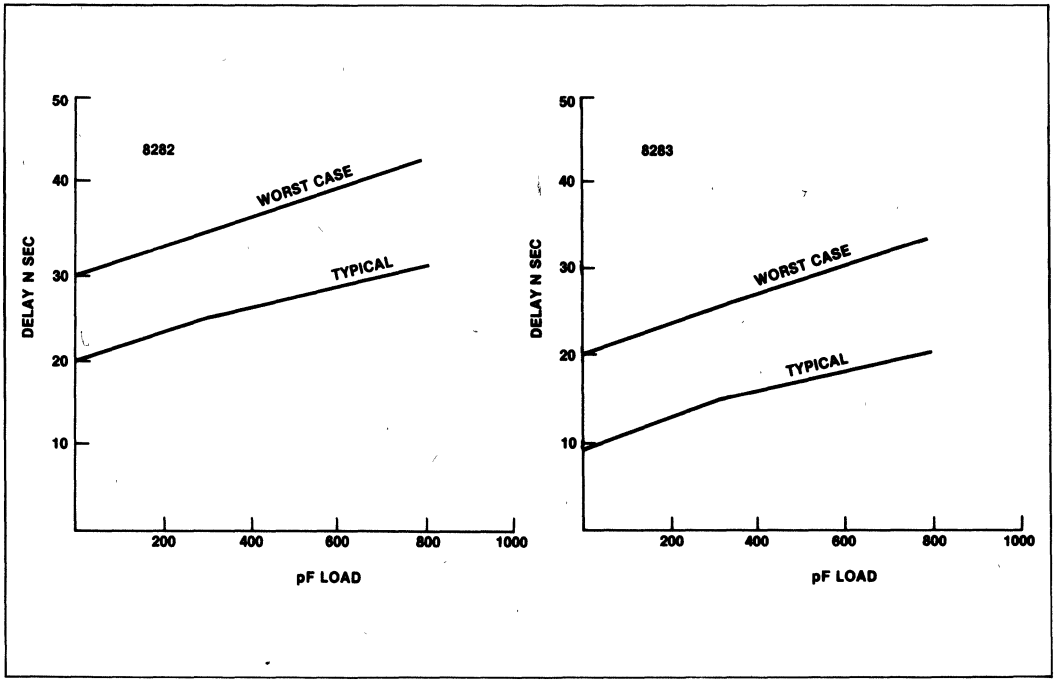


*200 pF for plastic 8282/8283.

WAVEFORMS



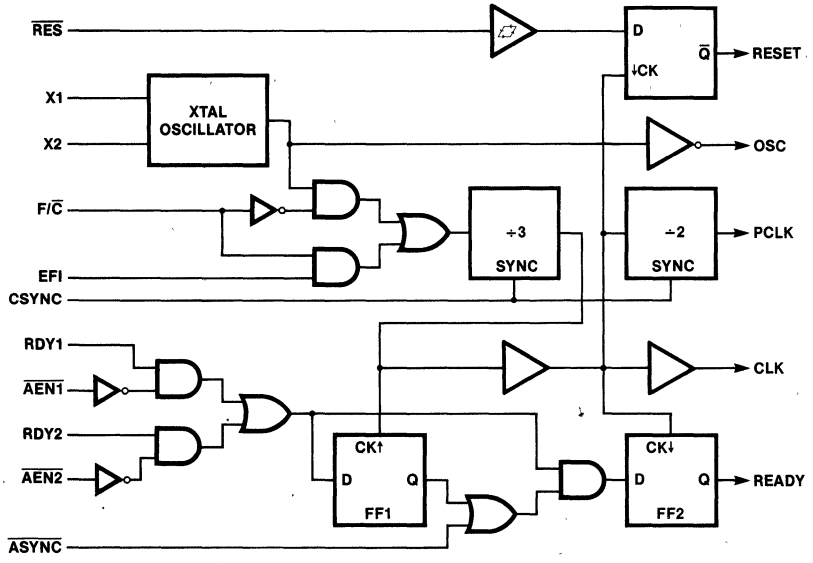
NOTE: 1. OUTPUT MAY BE MOMENTARILY INVALID FOLLOWING THE HIGH GOING STB TRANSITION.
2. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.



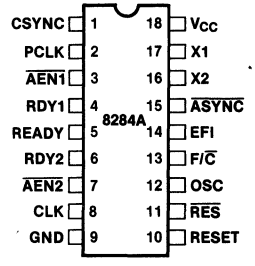
Output Delay vs. Capacitance

8284A/8284A-1
CLOCK GENERATOR AND DRIVER FOR iAPX 86, 88 PROCESSORS

- Generates the System Clock for the iAPX 86, 88 Processors:
  5 MHz, 8 MHz with 8284A
  10 MHz with 8284A-1
- Uses a Crystal or a TTL Signal for Frequency Source
- Provides Local READY and Multibus™ READY Synchronization
- 18-Pin Package
- Single +5V Power Supply
- Generates System Reset Output from Schmitt Trigger Input
- Capable of Clock Synchronization with Other 8284As
- Available in EXPRESS
  - Standard Temperature Range
  - Extended Temperature Range



8284A/8284A-1 Block Diagram



8284A/8284A-1 Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
$\overline{AEN1}$, $\overline{AEN2}$	I	**Address Enable:** $\overline{AEN}$ is an active LOW signal. AEN serves to qualify its respective Bus Ready Signal (RDY1 or RDY2). $\overline{AEN1}$ validates RDY1 while $\overline{AEN2}$ validates RDY2. Two AEN signal inputs are useful in system configurations which permit the processor to access two Multi-Master System Busses. In non Multi-Master configurations the $\overline{AEN}$ signal inputs are tied true (LOW).
RDY1, RDY2	I	**Bus Ready:** (Transfer Complete). RDY is an active HIGH signal which is an indication from a device located on the system data bus that data has been received, or is available. RDY1 is qualified by $\overline{AEN1}$ while RDY2 is qualified by $\overline{AEN2}$.
$\overline{ASYNC}$	I	**Ready Synchronization Select:** $\overline{ASYNC}$ is an input which defines the synchronization mode of the READY logic. When $\overline{ASYNC}$ is low, two stages of READY synchronization are provided. When $\overline{ASYNC}$ is left open (internal pull-up resistor is provided) or HIGH a single stage of READY synchronization is provided.
READY	O	**Ready:** READY is an active HIGH signal which is the synchronized RDY signal input. READY is cleared after the guaranteed hold time to the processor has been met.
X1, X2	I	**Crystal In:** X1 and X2 are the pins to which a crystal is attached. The crystal frequency is 3 times the desired processor clock frequency.
F/$\overline{C}$	I	**Frequency/Crystal Select:** F/$\overline{C}$ is a strapping option. When strapped LOW, F/$\overline{C}$ permits the processor's clock to be generated by the crystal. When F/$\overline{C}$ is strapped HIGH, CLK is generated from the EFI input.
EFI	I	**External Frequency:** When F/$\overline{C}$ is strapped HIGH, CLK is generated from the input frequency appearing on this pin. The input signal is a square wave 3 times the frequency of the desired CLK output.
CLK	O	**Processor Clock:** CLK is the clock output used by the processor and all devices which directly connect to the processor's local bus (i.e., the bipolar support chips and other MOS devices). CLK has an output frequency which is ⅓ of the crystal or EFI input frequency and a ⅓ duty cycle. An output HIGH of 4.5 volts (V_{CC}= 5V) is provided on this pin to drive MOS devices.
PCLK	O	**Peripheral Clock:** PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK and has a 50% duty cycle.
OSC	O	**Oscillator Output:** OSC is the TTL level output of the internal oscillator circuitry. Its frequency is equal to that of the crystal.
$\overline{RES}$	I	**Reset In:** $\overline{RES}$ is an active LOW signal which is used to generate RESET. The 8284A provides a Schmitt trigger input so that an RC connection can be used to establish the power-up reset of proper duration.
RESET	O	**Reset:** RESET is an active HIGH signal which is used to reset the 8086 family processors. Its timing characteristics are determined by $\overline{RES}$.
CSYNC	I	**Clock Synchronization:** CSYNC is an active HIGH signal which allows multiple 8284As to be synchronized to provide clocks that are in phase. When CSYNC is HIGH the internal counters are reset. When CSYNC goes LOW the internal counters are allowed to resume counting. CSYNC needs to be externally synchronized to EFI. When using the internal oscillator CSYNC should be hardwired to ground.
GND		**Ground.**
V_{CC}		**Power:** +5V supply.

FUNCTIONAL DESCRIPTION

General

The 8284A is a single chip clock generator/driver for the iAPX 86, 88 processors. The chip contains a crystal-controlled oscillator, a divide-by-three counter, complete MULTIBUS™ "Ready" synchronization and reset logic. Refer to Figure 1 for Block Diagram and Figure 2 for Pin Configuration.

Oscillator

The oscillator circuit of the 8284A is designed primarily for use with an external series resonant, fundamental mode, crystal from which the basic operating frequency is derived.

The crystal frequency should be selected at three times the required CPU clock. X1 and X2 are the two crystal input crystal connections. For the most stable operation of the oscillator (OSC) output circuit, two series resistors (R_1 = R_2 = 510 Ω) as shown in the waveform figures are recommended. The output of the oscillator is buffered and brought out on OSC so that other system timing signals can be derived from this stable, crystal-controlled source.

For systems which have a V_{CC} ramp time ≥ 1V/ms and/or have inherent board capacitance between X1 or X2, exceeding 10 pF (not including 8284A pin capacitance), the two 510Ω resistors should be used. This circuit provides optimum stability for the oscillator in such extreme conditions. It is advisable to limit stray capacitances to less than 10 pF on X1 and X2 to minimize deviation from operating at the fundamental frequency.

Clock Generator

The clock generator consists of a synchronous divide-by-three counter with a special clear input that inhibits the counting. This clear input (CSYNC) allows the output clock to be synchronized with an external event (such as another 8284A clock). It is necessary to synchronize the CSYNC input to the EFI clock external to the 8284A. This is accomplished with two Schottky flip-flops. The counter output is a 33% duty cycle clock at one-third the input frequency.

The F/$\overline{C}$ input is a strapping pin that selects either the crystal oscillator or the EFI input as the clock for the ÷3 counter. If the EFI input is selected as the clock source, the oscillator section can be used independently for another clock source. Output is taken from OSC.

Clock Outputs

The CLK output is a 33% duty cycle MOS clock driver designed to drive the iAPX 86, 88 processors directly. PCLK is a TTL level peripheral clock signal whose output frequency is ½ that of CLK. PCLK has a 50% duty cycle.

Reset Logic

The reset logic provides a Schmitt trigger input ($\overline{RES}$) and a synchronizing flip-flop to generate the reset timing. The reset signal is synchronized to the falling edge of CLK. A simple RC network can be used to provide power-on reset by utilizing this function of the 8284A.

READY Synchronization

Two READY inputs (RDY1, RDY2) are provided to accommodate two Multi-Master system busses. Each input has a qualifier ($\overline{AEN1}$ and $\overline{AEN2}$, respectively). The $\overline{AEN}$ signals validate their respective RDY signals. If a Multi-Master system is not being used the $\overline{AEN}$ pin should be tied LOW.

Synchronization is required for all asynchronous active-going edges of either RDY input to guarantee that the RDY setup and hold times are met. Inactive-going edges of RDY in normally ready systems do not require synchronization but must satisfy RDY setup and hold as a matter of proper system design.

The $\overline{ASYNC}$ input defines two modes of READY synchronization operation.

When $\overline{ASYNC}$ is LOW, two stages of synchronization are provided for active READY input signals. Positive-going asynchronous READY inputs will first be synchronized to flip-flop one at the rising edge of CLK and then synchronized to flip-flop two at the next falling edge of CLK, after which time the READY output will go active (HIGH). Negative-going asynchronous READY inputs will be synchronized directly to flip-flop two at the falling edge of CLK, after which time the READY output will go inactive. This mode of operation is intended for use by asynchronous (normally not ready) devices in the system which cannot be guaranteed by design to meet the required RDY setup timing, T_{R1VCL}, on each bus cycle.

When $\overline{ASYNC}$ is high or left open, the first READY flip-flop is bypassed in the READY synchronization logic. READY inputs are synchronized by flip-flop two on the falling edge of CLK before they are presented to the processor. This mode is available for synchronous devices that can be guaranteed to meet the required RDY setup time.

$\overline{ASYNC}$ can be changed on every bus cycle to select the appropriate mode of synchronization for each device in the system.

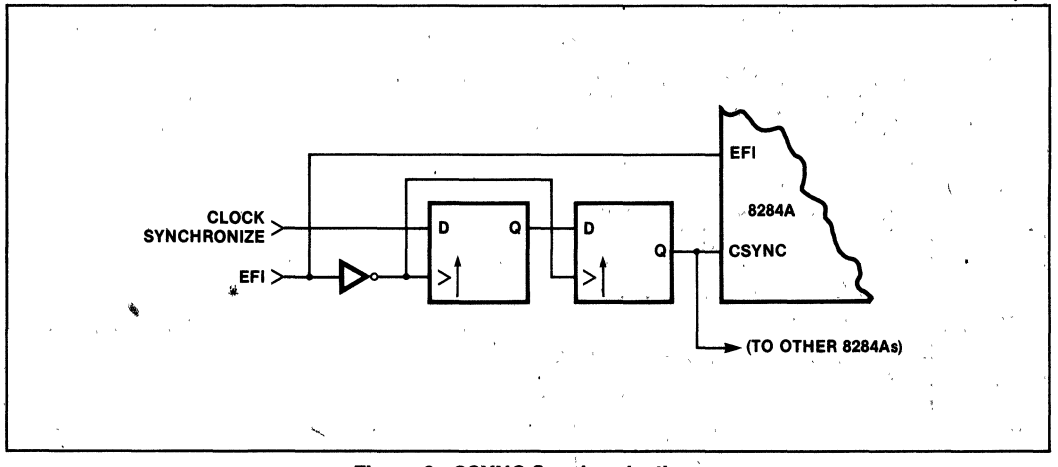


Figure 3. CSYNC Synchronization

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
All Output and Supply Voltages ..... −0.5V to +7V
All Input Voltages ..... −1.0V to +5.5V
Power Dissipation ..... 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
I_F	Forward Input Current ($\overline{ASYNC}$) Other Inputs		−1.3 −0.5	mA mA	$V_F = 0.45V$ $V_F = 0.45V$
I_R	Reverse Input Current ($\overline{ASYNC}$) Other Inputs		50 50	μA μA	$V_R = V_{CC}$ $V_R = 5.25V$
V_C	Input Forward Clamp Voltage		−1.0	V	$I_C = -5mA$
I_{CC}	Power Supply Current		162	mA	
V_{IL}	Input LOW Voltage		0.8	V	
V_{IH}	Input HIGH Voltage	2.0		V	
V_{IHR}	Reset Input HIGH Voltage	2.6		V	
V_{OL}	Output LOW Voltage		0.45	V	5mA
V_{OH}	Output HIGH Voltage CLK Other Outputs	4 2.4		V V	−1mA −1mA
$V_{IHR} - V_{ILR}$	$\overline{RES}$ Input Hysteresis	0.25		V	

A.C. CHARACTERISTICS ($T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Units	Test Conditions
t_{EHEL}	External Frequency HIGH Time	13		ns	90%-90% V_{IN}
t_{ELEH}	External Frequency LOW Time	13		ns	10%-10% V_{IN}
t_{ELEL}	EFI Period	33		ns	(Note 1)
	XTAL Frequency	12	25	MHz	
t_{R1VCL}	RDY1, RDY2 Active Setup to CLK	35		ns	$\overline{ASYNC}$ = HIGH
t_{R1VCH}	RDY1, RDY2 Active Setup to CLK	35		ns	$\overline{ASYNC}$ = LOW
t_{R1VCL}	RDY1, RDY2 Inactive Setup to CLK	35		ns	
t_{CLR1X}	RDY1, RDY2 Hold to CLK	0		ns	
t_{AYVCL}	$\overline{ASYNC}$ Setup to CLK	50		ns	
t_{CLAYX}	$\overline{ASYNC}$ Hold to CLK	0		ns	
t_{A1VR1V}	$\overline{AEN1}$, $\overline{AEN2}$ Setup to RDY1, RDY2	15		ns	
t_{CLA1X}	$\overline{AEN1}$, $\overline{AEN2}$ Hold to CLK	0		ns	
t_{YHEH}	CSYNC Setup to EFI	20		ns	
t_{EHYL}	CSYNC Hold to EFI	10		ns	
t_{YHYL}	CSYNC Width	$2 \cdot t_{ELEL}$		ns	
t_{I1HCL}	$\overline{RES}$ Setup to CLK	65		ns	(Note 1)
t_{CLI1H}	$\overline{RES}$ Hold to CLK	20		ns	(Note 1)

A.C. CHARACTERISTICS (Continued)

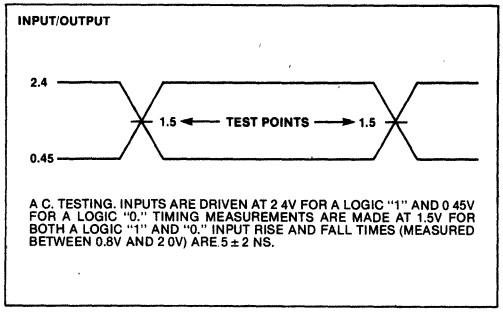
TIMING RESPONSES

Symbol	Parameter	Min. 8284A	Min. 8284A-1	Max.	Units	Test Conditions
t_{CLCL}	CLK Cycle Period	125	100		ns	
t_{CHCL}	CLK HIGH Time	$(1/3\ t_{CLCL})+2$	39		ns	
t_{CLCH}	CLK LOW Time	$(2/3\ t_{CLCL})-15$	53		ns	
t_{CH1CH2} t_{CL2CL1}	CLK Rise or Fall Time			10	ns	1.0V to 3.5V
t_{PHPL}	PCLK HIGH Time	$t_{CLCL}-20$	$t_{CLCL}-20$		ns	
t_{PLPH}	PCLK LOW Time	$t_{CLCL}-20$	$t_{CLCL}-20$		ns	
t_{RYLCL}	Ready Inactive to CLK (See Note 3)	−8	−8		ns	
t_{RYHCH}	Ready Active to CLK (See Note 2)	$(2/3\ t_{CLCL})-15$	53		ns	
t_{CLIL}	CLK to Reset Delay			40	ns	
t_{CLPH}	CLK to PCLK HIGH DELAY			22	ns	
t_{CLPL}	CLK to PCLK LOW Delay			22	ns	
t_{OLCH}	OSC to CLK HIGH Delay	−5	−5	22	ns	
t_{OLCL}	OSC to CLK LOW Delay	2	2	35	ns	
t_{OLOH}	Output Rise Time (except CLK)			20	ns	From 0.8V to 2.0V
t_{OHOL}	Output Fall Time (except CLK)			12	ns	From 2.0V to 0.8V

NOTES:

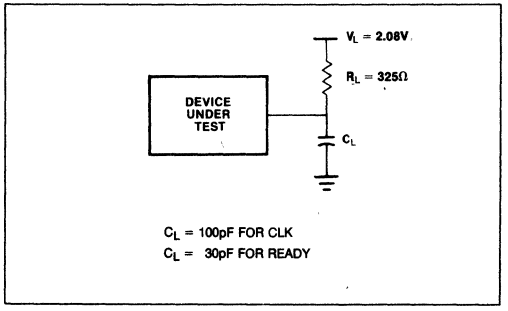
1. Setup and hold necessary only to guarantee recognition at next clock.
2. Applies only to T3 and TW states.
3. Applies only to T2 states.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING. INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0." TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES (MEASURED BETWEEN 0.8V AND 2.0V) ARE 5 ± 2 NS.

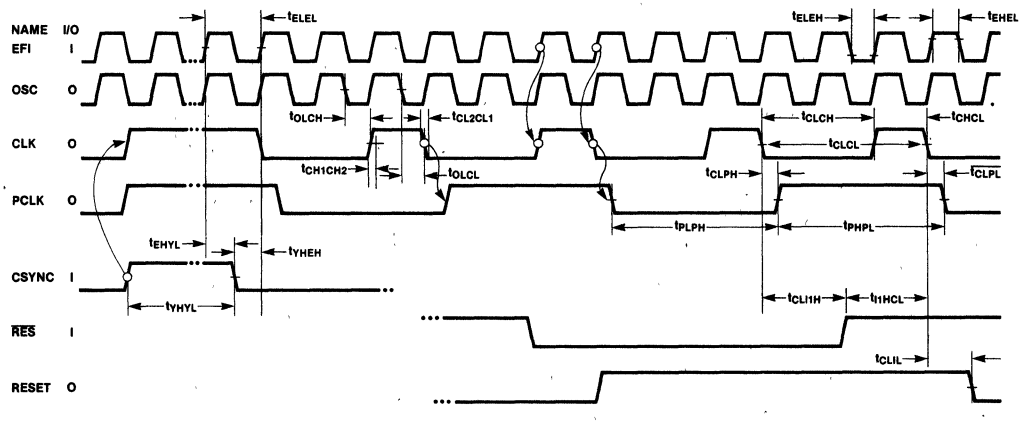
A.C. TESTING LOAD CIRCUIT



C_L = 100pF FOR CLK
C_L = 30pF FOR READY

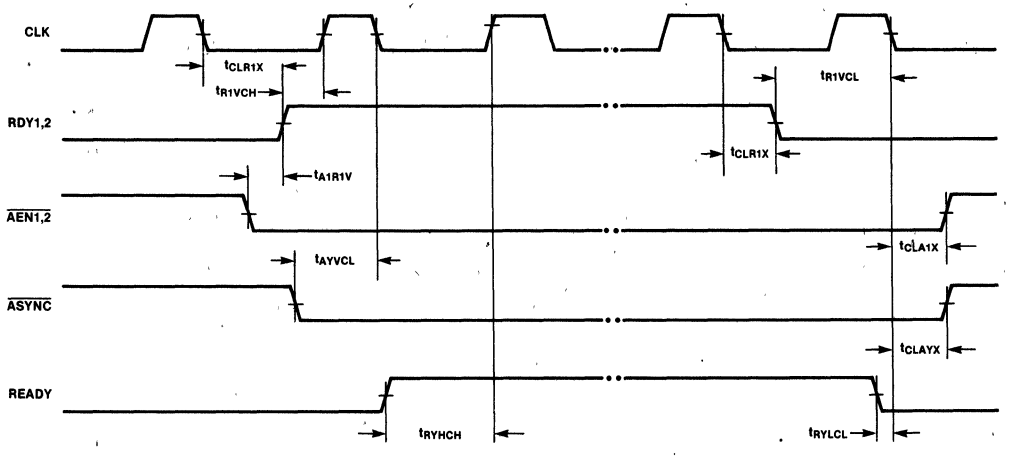
WAVEFORMS

CLOCKS AND RESET SIGNALS



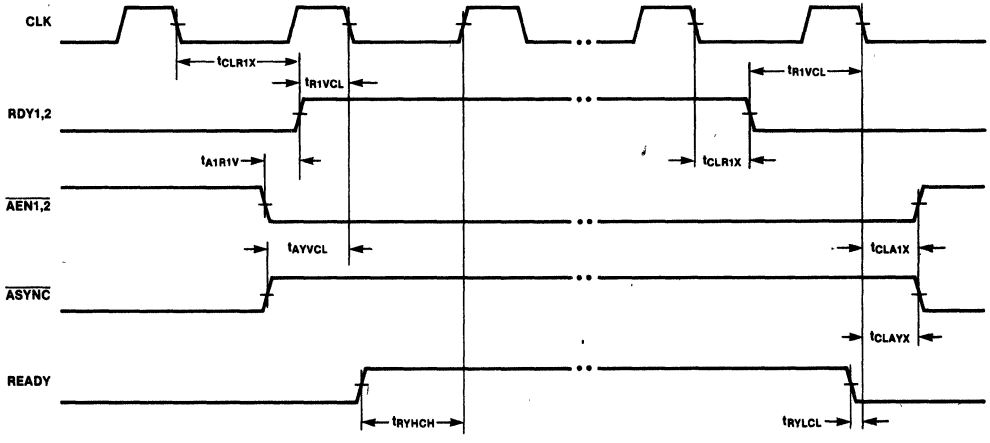
NOTE: ALL TIMING MEASUREMENTS ARE MADE AT 1.5 VOLTS, UNLESS OTHERWISE NOTED.

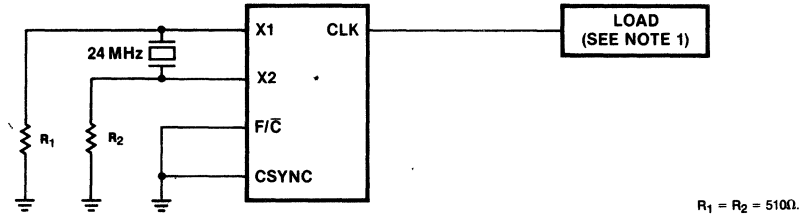
READY SIGNALS (FOR ASYNCHRONOUS DEVICES)



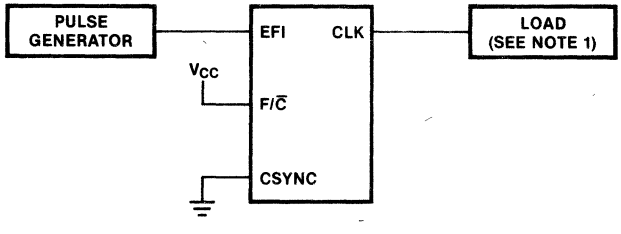
WAVEFORMS (Continued)

READY SIGNALS (FOR SYNCHRONOUS DEVICES)

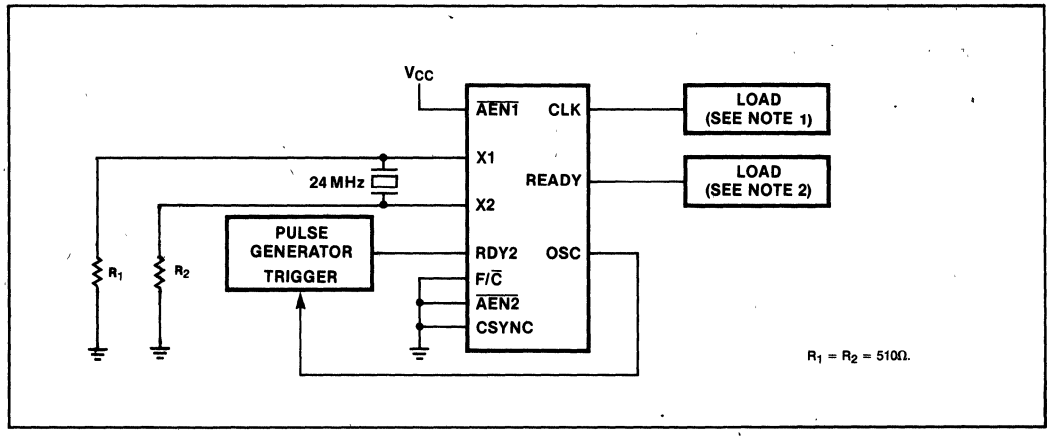




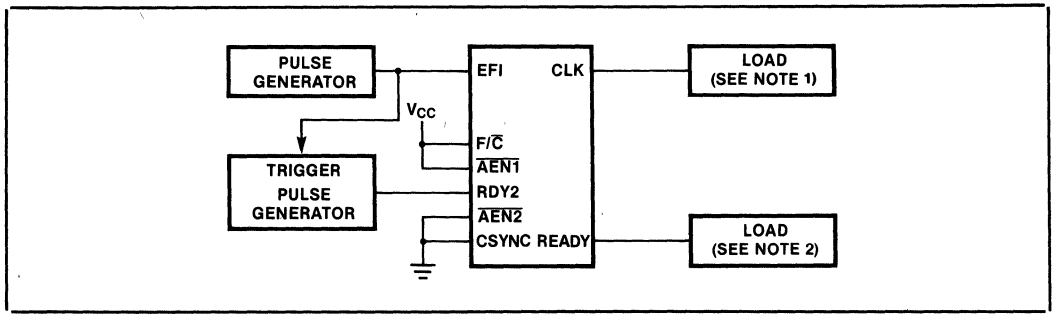
Clock High and Low Time (Using X1, X2)



Clock High and Low Time (Using EFI)



Ready to Clock (Using X1, X2)



Ready to Clock (Using EFI)

NOTES:
1 C_L = 100 pF
2 C_L = 30 pF

8286/8287
OCTAL BUS TRANSCEIVER

- **Data Bus Buffer Driver for iAPX 86,88,186,188, MCS-80™, MCS-85™, and MCS-48™ Families**
- **High Output Drive Capability for Driving System Data Bus**
- **Fully Parallel 8-Bit Transceivers**
- **3-State Outputs**
- **20-Pin Package with 0.3" Center**
- **No Output Low Noise when Entering or Leaving High Impedance State**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its outputs while the 8286 does not. Thus, a wide variety of applications for buffering in microcomputer systems can be met.

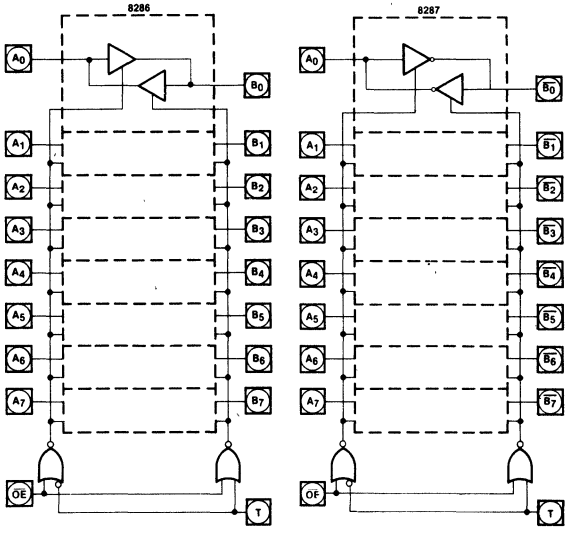


Figure 1. Logic Diagrams

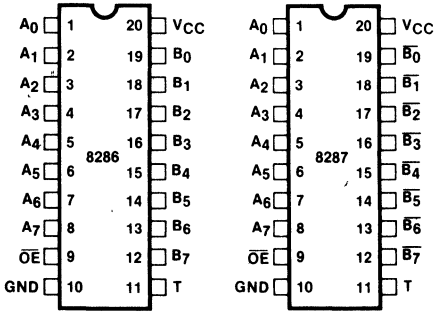


Figure 2. Pin Configurations

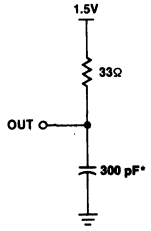
Table 1. Pin Description

Symbol	Type	Name and Function
T	I	**Transmit:** T is an input control signal used to control the direction of the transceivers. When HIGH, it configures the transceiver's B_0–B_7 as outputs with A_0–A_7 as inputs. T LOW configures A_0–A_7 as the outputs with B_0–B_7 serving as the inputs.
$\overline{OE}$	I	**Output Enable:** $\overline{OE}$ is an input control signal used to enable the appropriate output driver (as selected by T) onto its respective bus. This signal is active LOW.
A_0–A_7	I/O	**Local Bus Data Pins:** These pins serve to either present data to or accept data from the processor's local bus depending upon the state of the T pin.
B_0–B_7(8286) $\overline{B_0}$–$\overline{B_7}$(8287)	I/O	**System Bus Data Pins:** These pins serve to either present data to or accept data from the system bus depending upon the state of the T pin.

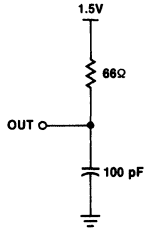
FUNCTIONAL DESCRIPTION

The 8286 and 8287 transceivers are 8-bit transceivers with high impedance outputs. With T active HIGH and $\overline{OE}$ active LOW, data at the A_0–A_7 pins is driven onto the B_0–B_7 pins. With T inactive LOW and $\overline{OE}$ active LOW, data at the B_0–B_7 pins is driven onto the A_0–A_7 pins. No output low glitching will occur whenever the transceivers are entering or leaving the high impedance state.

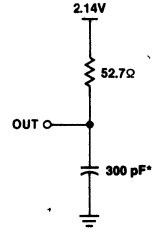
TEST LOAD CIRCUITS



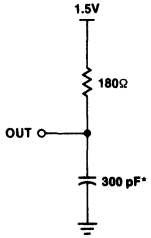
B OUTPUT



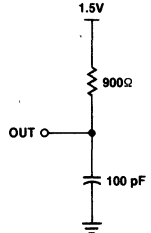
A OUTPUT



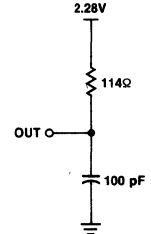
B OUTPUT



B OUTPUT



A OUTPUT



A OUTPUT

*200 pF for plastic 8286/8287

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
All Output and Supply Voltages . . . . . −0.5V to +7V
All Input Voltages . . . . . −1.0V to +5.5V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		-1	V	I_C = -5 mA
I_{CC}	Power Supply Current—8287 —8286		130 160	mA mA	
I_F	Forward Input Current		-0.2	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = 5.25V
V_{OL}	Output Low Voltage —B Outputs —A Outputs		.45 .45	V V	I_{OL} = 32 mA I_{OL} = 16 mA
V_{OH}	Output High Voltage —B Outputs —A Outputs	2.4 2.4		V V	I_{OH} = -5 mA I_{OH} = -1 mA
I_{OFF} I_{OFF}	Output Off Current Output Off Current		I_F I_R		V_{OFF} = 0.45V V_{OFF} = 5.25V
V_{IL}	Input Low Voltage —A Side —B Side		0.8 0.9	V V	V_{CC} = 5.0V, See Note 1 V_{CC} = 5.0V, See Note 1
V_{IH}	Input High Voltage	2.0		V	V_{CC} = 5.0V, See Note 1
C_{IN}	Input Capacitance		12	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTE:
1. B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*: A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF.

A.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C) (See Note 2)

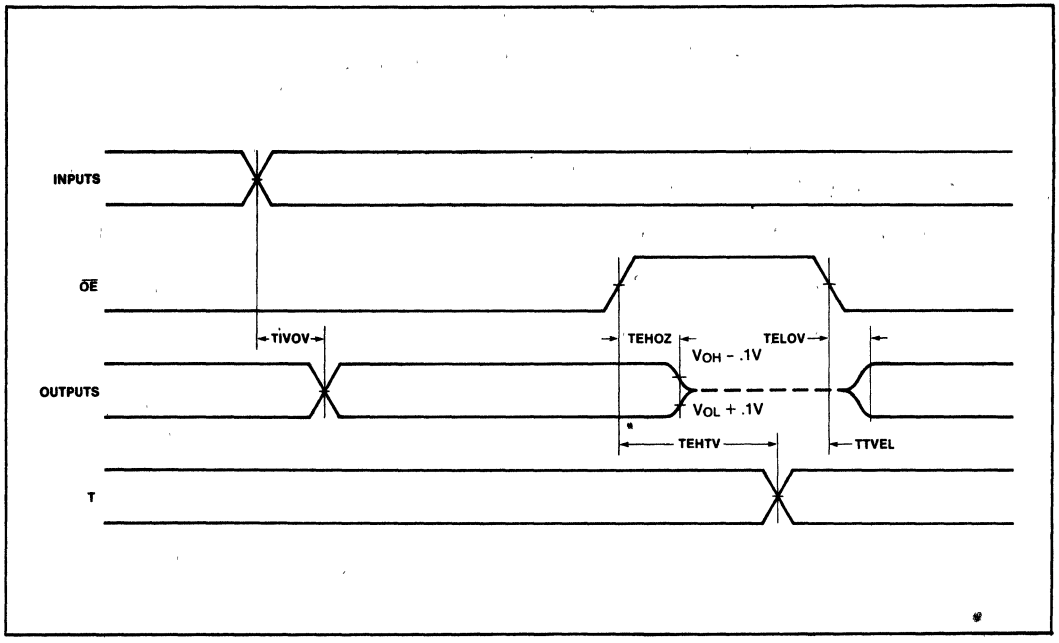
Loading: B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*
A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF

Symbol	Parameter	Min	Max	Units	Test Conditions
TIVOV	Input to Output Delay				
	Inverting	5	22	ns	(See Note 1)
	Non-Inverting	5	30	ns	
TEHTV	Transmit/Receive Hold Time	5		ns	
TTVEL	Transmit/Receive Setup	10		ns	
TEHOZ	Output Disable Time	5	18	ns	
TELOV	Output Enable Time	10	30	ns	
TOLOH	Input, Output Rise Time		20	ns	From 0.8 V to 2.0V
TOHOL	Input, Output Fall Time		12	ns	From 2.0V to 8.0V

*C_L = 200 pF for plastic 8286/8287

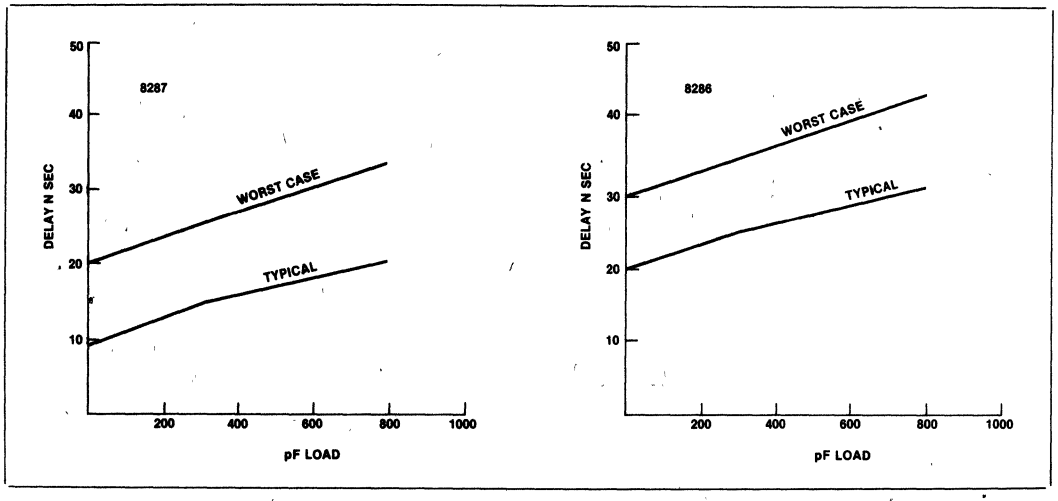
NOTE:
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TEHOZ = 25; TELOV = 50.

WAVEFORMS



NOTE:

1. All timing measurements are made at 1.5V unless otherwise noted.



Output Delay versus Capacitance

8288
BUS CONTROLLER
FOR iAPX 86, 88 PROCESSORS

- **Bipolar Drive Capability**
- **Provides Advanced Commands**
- **Provides Wide Flexibility in System Configurations**
- **3-State Command Output Drivers**
- **Configurable for Use with an I/O Bus**
- **Facilitates Interface to One or Two Multi-Master Busses**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel® 8288 Bus Controller is a 20-pin bipolar component for use with medium-to-large iAPX 86, 88 processing systems. The bus controller provides command and control timing generation as well as bipolar bus drive capability while optimizing system performance.

A strapping option on the bus controller configures it for use with a multi-master system bus and separate I/O bus.

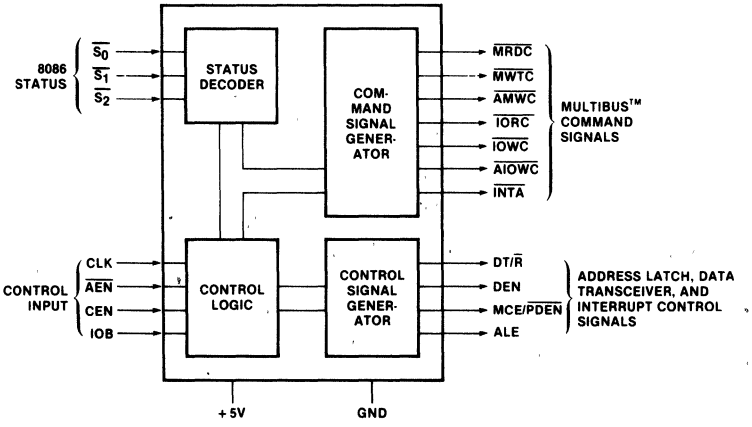


Figure 1. Block Diagram

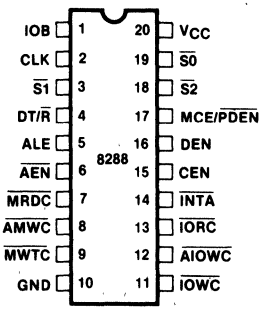


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5V supply.
GND		**Ground.**
$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$	I	**Status Input Pins:** These pins are the status input pins from the 8086, 8088 or 8089 processors. The 8288 decodes these inputs to generate command and control signals at the appropriate time. When these pins are not in use (passive) they are all HIGH. (See chart under Command and Control Logic.)
CLK	I	**Clock:** This is a clock signal from the 8284 clock generator and serves to establish when command and control signals are generated.
ALE	O	**Address Latch Enable:** This signal serves to strobe an address into the address latches. This signal is active HIGH and latching occurs on the falling (HIGH to LOW) transition. ALE is intended for use with transparent D type latches.
DEN	O	**Data Enable:** This signal serves to enable data transceivers onto either the local or system data bus. This signal is active HIGH.
DT/$\overline{R}$	O	**Data Transmit/Receive:** This signal establishes the direction of data flow through the transceivers. A HIGH on this line indicates Transmit (write to I/O or memory) and a LOW indicates Receive (Read).
$\overline{AEN}$	I	**Address Enable:** $\overline{AEN}$ enables command outputs of the 8288 Bus Controller at least 115 ns after it becomes active (LOW). $\overline{AEN}$ going inactive immediately 3-states the command output drivers. $\overline{AEN}$ does not affect the I/O command lines if the 8288 is in the I/O Bus mode (IOB tied HIGH).
CEN	I	**Command Enable:** When this signal is LOW all 8288 command outputs and the DEN and $\overline{PDEN}$ control outputs are forced to their inactive state. When this signal is HIGH, these same outputs are enabled.
IOB	I	**Input/Output Bus Mode:** When the IOB is strapped HIGH the 8288 functions in the I/O Bus mode. When it is strapped LOW, the 8288 functions in the System Bus mode. (See sections on I/O Bus and System Bus modes).

Symbol	Type	Name and Function
$\overline{AIOWC}$	O	**Advanced I/O Write Command:** The $\overline{AIOWC}$ issues an I/O Write Command earlier in the machine cycle to give I/O devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{AIOWC}$ is active LOW.
$\overline{IOWC}$	O	**I/O Write Command:** This command line instructs an I/O device to read the data on the data bus. This signal is active LOW.
$\overline{IORC}$	O	**I/O Read Command:** This command line instructs an I/O device to drive its data onto the data bus. This signal is active LOW.
$\overline{AMWC}$	O	**Advanced Memory Write Command:** The $\overline{AMWC}$ issues a memory write command earlier in the machine cycle to give memory devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{AMWC}$ is active LOW.
$\overline{MWTC}$	O	**Memory Write Command:** This command line instructs the memory to record the data present on the data bus. This signal is active LOW.
$\overline{MRDC}$	O	**Memory Read Command:** This command line instructs the memory to drive its data onto the data bus. This signal is active LOW.
$\overline{INTA}$	O	**Interrupt Acknowledge:** This command line tells an interrupting device that its interrupt has been acknowledged and that it should drive vectoring information onto the data bus. This signal is active LOW.
MCE/$\overline{PDEN}$	O	This is a dual function pin. **MCE (IOB is tied LOW):** Master Cascade Enable occurs during an interrupt sequence and serves to read a Cascade Address from a master PIC (Priority Interrupt Controller) onto the data bus. The MCE signal is active HIGH. **$\overline{PDEN}$ (IOB is tied HIGH):** Peripheral Data Enable enables the data bus transceiver for the I/O bus that DEN performs for the system bus. $\overline{PDEN}$ is active LOW.

FUNCTIONAL DESCRIPTION

Command and Control Logic

The command logic decodes the three 8086, 8088 or 8089 CPU status lines ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) to determine what command is to be issued.

This chart shows the meaning of each status "word".

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Processor State	8288 Command
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Code Access	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

The command is issued in one of two ways dependent on the mode of the 8288 Bus Controller.

I/O Bus Mode — The 8288 is in the I/O Bus mode if the IOB pin is strapped HIGH. In the I/O Bus mode all I/O command lines ($\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$) are always enabled (i.e., not dependent on $\overline{AEN}$). When an I/O command is initiated by the processor, the 8288 immediately activates the command lines using $\overline{PDEN}$ and DT/$\overline{R}$ to control the I/O bus transceiver. The I/O command lines should not be used to control the system bus in this configuration because no arbitration is present. This mode allows one 8288 Bus Controller to handle two external busses. No waiting is involved when the CPU wants to gain access to the I/O bus. Normal memory access requires a "Bus Ready" signal ($\overline{AEN}$ LOW) before it will proceed. It is advantageous to use the IOB mode if I/O or peripherals dedicated to one processor exist in a multi-processor system.

System Bus Mode — The 8288 is in the System Bus mode if the IOB pin is strapped LOW. In this mode no command is issued until 115 ns after the $\overline{AEN}$ Line is activated (LOW). This mode assumes bus arbitration logic will inform the bus controller (on the $\overline{AEN}$ line) when the bus is free for use. Both memory and I/O commands wait for bus arbitration. This mode is used when only one bus exists. Here, both I/O and memory are shared by more than one processor.

COMMAND OUTPUTS

The advanced write commands are made available to initiate write procedures early in the machine cycle. This signal can be used to prevent the processor from entering an unnecessary wait state.

The command outputs are:

- $\overline{MRDC}$ — Memory Read Command
- $\overline{MWTC}$ — Memory Write Command
- $\overline{IORC}$ — I/O Read Command
- $\overline{IOWC}$ — I/O Write Command
- $\overline{AMWC}$ — Advanced Memory Write Command
- $\overline{AIOWC}$ — Advanced I/O Write Command
- $\overline{INTA}$ — Interrupt Acknowledge

$\overline{INTA}$ (Interrupt Acknowledge) acts as an I/O read during an interrupt cycle. Its purpose is to inform an interrupting device that its interrupt is being acknowledged and that it should place vectoring information onto the data bus.

CONTROL OUTPUTS

The control outputs of the 8288 are Data Enable (DEN), Data Transmit/Receive (DT/$\overline{R}$) and Master Cascade Enable/Peripheral Data Enable (MCE/$\overline{PDEN}$). The DEN signal determines when the external bus should be enabled onto the local bus and the DT/$\overline{R}$ determines the direction of data transfer. These two signals usually go to the chip select and direction pins of a transceiver.

The MCE/$\overline{PDEN}$ pin changes function with the two modes of the 8288. When the 8288 is in the IOB mode (IOB HIGH) the $\overline{PDEN}$ signal serves as a dedicated data enable signal for the I/O or Peripheral System bus.

INTERRUPT ACKNOWLEDGE AND MCE

The MCE signal is used during an interrupt acknowledge cycle if the 8288 is in the System Bus mode (IOB LOW). During any interrupt sequence there are two interrupt acknowledge cycles that occur back to back. During the first interrupt cycle no data or address transfers take place. Logic should be provided to mask off MCE during this cycle. Just before the second cycle begins the MCE signal gates a master Priority Interrupt Controller's (PIC) cascade address onto the processor's local bus where ALE (Address Latch Enable) strobes it into the address latches. On the leading edge of the second interrupt cycle the addressed slave PIC gates an interrupt vector onto the system data bus where it is read by the processor.

If the system contains only one PIC, the MCE signal is not used. In this case the second Interrupt Acknowledge signal gates the interrupt vector onto the processor bus.

ADDRESS LATCH ENABLE AND HALT

Address Latch Enable (ALE) occurs during each machine cycle and serves to strobe the current address into the address latches. ALE also serves to strobe the status ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) into a latch for halt state decoding.

COMMAND ENABLE

The Command Enable (CEN) input acts as a command qualifier for the 8288. If the CEN pin is high the 8288 functions normally. If the CEN pin is pulled LOW, all command lines are held in their inactive state (not 3-state). This feature can be used to implement memory partitioning and to eliminate address conflicts between system bus devices and resident bus devices.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
All Output and Supply Voltages ..... −0.5V to +7V
All Input Voltages ..... −1.0V to +5.5V
Power Dissipation ..... 1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_C	Input Clamp Voltage		−1	V	I_C = −5 mA
I_{CC}	Power Supply Current		230	mA	
I_F	Forward Input Current		−0.7	mA	V_F = 0.45V
I_R	Reverse Input Current		50	μA	V_R = V_{CC}
V_{OL}	Output Low Voltage				
	Command Outputs		0.5	V	I_{OL} = 32 mA
	Control Outputs		0.5	V	I_{OL} = 16 mA
V_{OH}	Output High Voltage				
	Command Outputs	2.4		V	I_{OH} = −5 mA
	Control Outputs	2.4		V	I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	
V_{IH}	Input High Voltage	2.0		V	
I_{OFF}	Output Off Current		100	μA	V_{OFF} = 0.4 to 5.25V

A.C. CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)*

TIMING REQUIREMENTS

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
TCLCL	CLK Cycle Period	100		ns	
TCLCH	CLK Low Time	50		ns	
TCHCL	CLK High Time	30		ns	
TSVCH	Status Active Setup Time	35		ns	
TCHSV	Status Inactive Hold Time	10		ns	
TSHCL	Status Inactive Setup Time	35		ns	
TCLSH	Status Active Hold Time	10		ns	

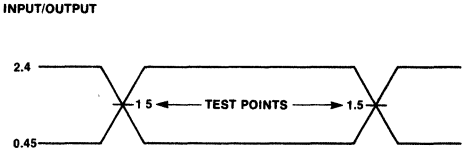
* Note: For Extended Temperature EXPRESS the Preliminary Values are TCLCL = 125; TCLCH = 50; TCHCL = 30; TCVNX = 50; TCLLH, TCLMCH = 25; TSVLH, TSVMCH = 25.

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

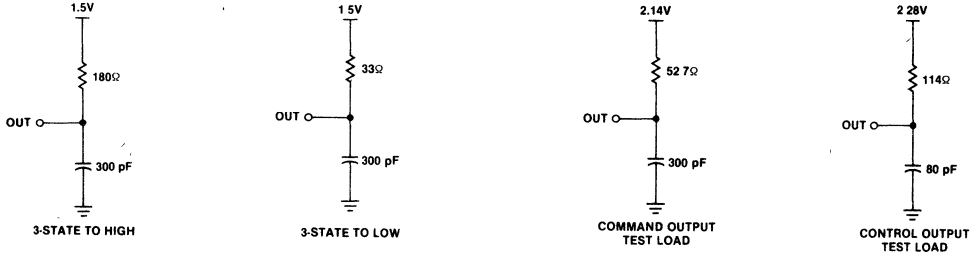
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
TCVNV	Control Active Delay	5	45	ns	
TCVNX	Control Inactive Delay	10	45	ns	
TCLLH, TCLMCH	ALE MCE Active Delay (from CLK)		20	ns	
TSVLH, TSVMCH	ALE MCE Active Delay (from Status)		20	ns	
TCHLL	ALE Inactive Delay	4	15	ns	$\overline{MRDC}$, $\overline{IORC}$, $\overline{MWTC}$, $\overline{IOWC}$, $\overline{INTA}$, $\overline{AMWC}$, $\overline{AIOWC}$: I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF
TCLML	Command Active Delay	10	35	ns	
TCLMH	Command Inactive Delay	10	35	ns	
TCHDTL	Direction Control Active Delay		50	ns	
TCHDTH	Direction Control Inactive Delay		30	ns	
TAELCH	Command Enable Time		40	ns	
TAEHCZ	Command Disable Time		40	ns	
TAELCV	Enable Delay Time	115	200	ns	Other: I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 80 pF
TAEVNV	AEN to DEN		20	ns	
TCEVNV	CEN to DEN, PDEN		25	ns	
TCELRH	CEN to Command		TCLML	ns	
TOLOH	Output, Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Output, Fall Time		12	ns	From 2.0V to 0.8V

A.C. TESTING INPUT, OUTPUT WAVEFORM

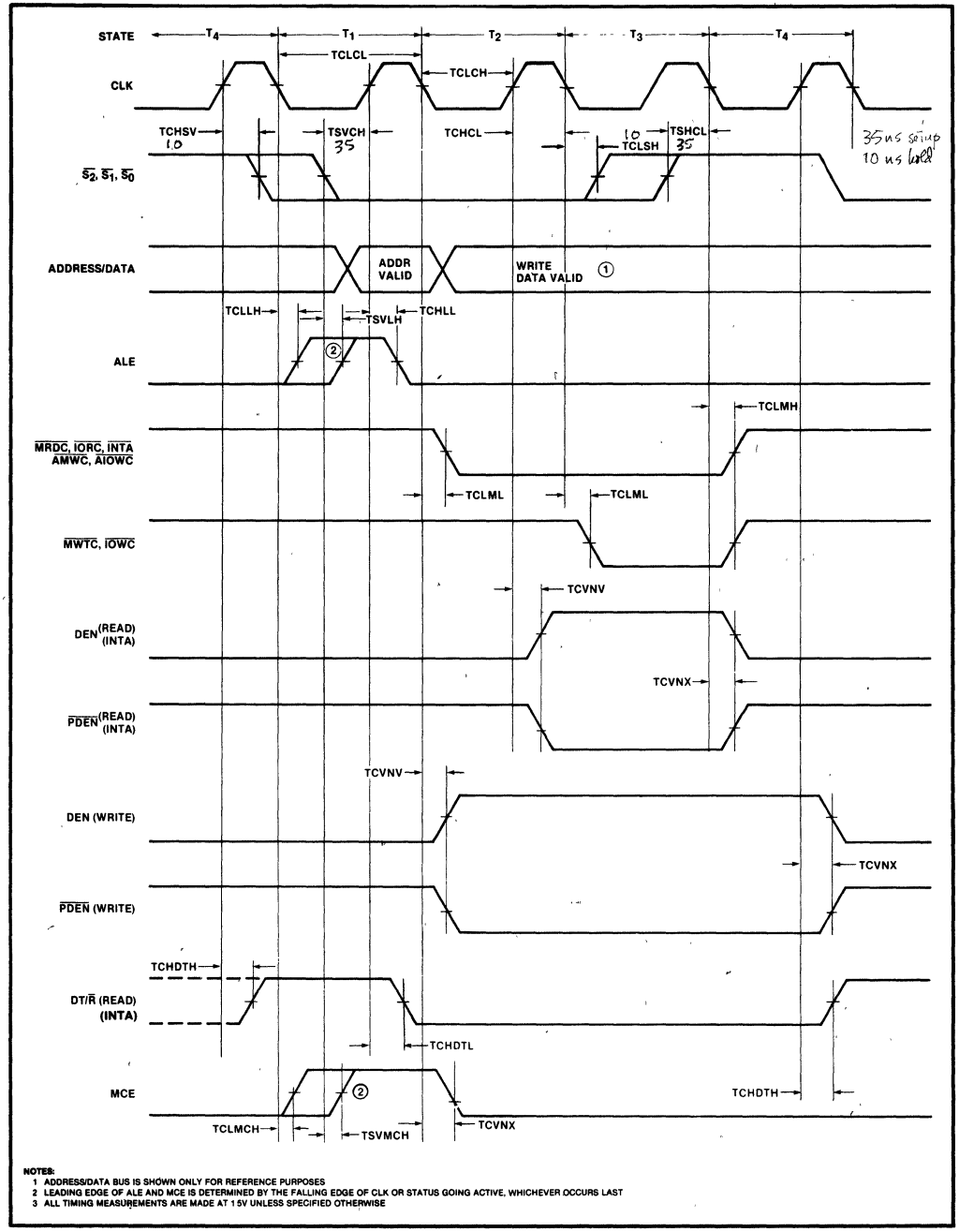


A.C. TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". THE CLOCK IS DRIVEN AT 4.3V AND 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0." INPUT RISE AND FALL TIMES ARE 5 ± 2 NS, MEASURED BETWEEN 0.8V AND 2.0V.

TEST LOAD CIRCUITS—3-STATE COMMAND OUTPUT TEST LOAD



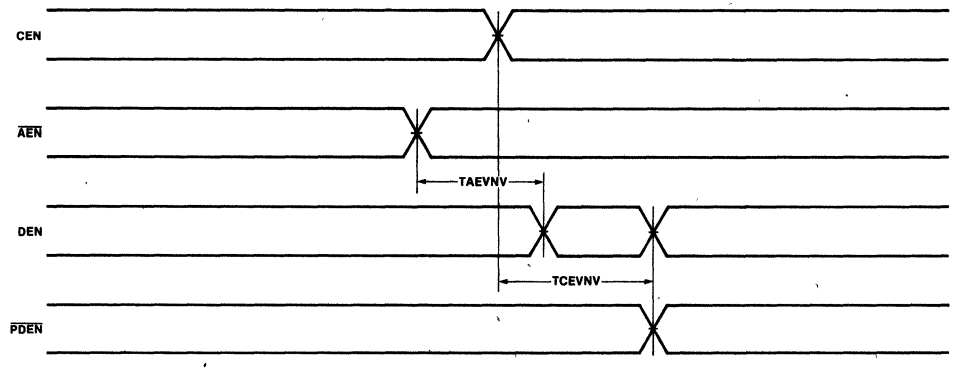
WAVEFORMS



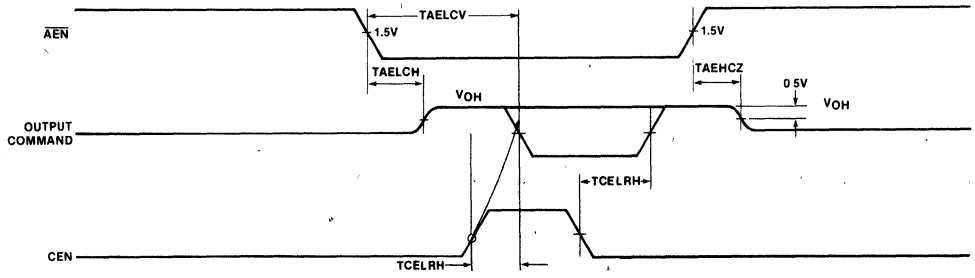
NOTES:
1. ADDRESS/DATA BUS IS SHOWN ONLY FOR REFERENCE PURPOSES
2. LEADING EDGE OF ALE AND MCE IS DETERMINED BY THE FALLING EDGE OF CLK OR STATUS GOING ACTIVE, WHICHEVER OCCURS LAST
3. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS SPECIFIED OTHERWISE

WAVEFORMS (Continued)

DEN, PDEN QUALIFICATION TIMING



ADDRESS ENABLE (AEN) TIMING (3-STATE ENABLE/DISABLE)



NOTE: CEN must be low or valid prior to T2 to prevent the command from being generated.

8289/8289-1
BUS ARBITER

- **Provides Multi-Master System Bus Protocol**
- **Synchronizes iAPX 86, 88 Processors with Multi-Master Bus**
- **10MHz Version, 8289-1, Fully Compatible with 10MHz iAPX 86 or 8MHz iAPX 186 Based Systems**
- **Provides Simple Interface with 8288 Bus Controller**
- **Four Operating Modes for Flexible System Configuration**
- **Compatible with Intel Bus Standard MULTIBUS™**
- **Provides System Bus Arbitration for 8089 IOP in Remote Mode**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The Intel 8289 Bus Arbiter is a 20-pin, 5-volt-only bipolar component for use with medium to large iAPX 86, 88 multi-master/multiprocessing systems. The 8289 provides system bus arbitration for systems with multiple bus masters, such as an 8086 CPU with 8089 IOP in its REMOTE mode, while providing bipolar buffering and drive capability.

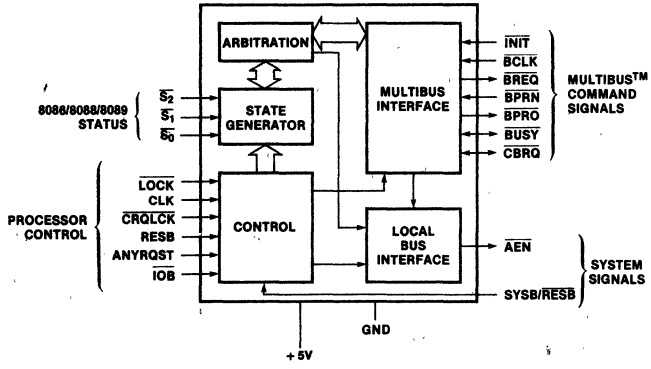


Figure 1. Block Diagram

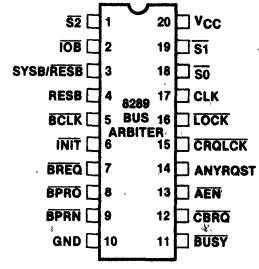


Figure 2. Pin Diagram

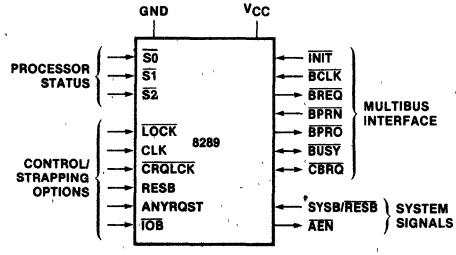


Figure 3. Functional Pinout

Table 1. Pin Description

Symbol	Type	Name and Function
V_{CC}		**Power:** +5V supply ±10%.
GND		**Ground.**
$\overline{S0}$,$\overline{S1}$,$\overline{S2}$	I	**Status Input Pins:** The status input pins from an 8086, 8088 or 8089 processor. The 8289 decodes these pins to initiate bus request and surrender actions. (See Table 2.)
CLK	I	**Clock:** From the 8284 clock chip and serves to establish when bus arbiter actions are initiated.
$\overline{LOCK}$	I	**Lock:** A processor generated signal which when activated (low) prevents the arbiter from surrendering the multi-master system bus to any other bus artiter, regardless of its priority.
$\overline{CRQLCK}$	I	**Common Request Lock:** An active low signal which prevents the arbiter from surrendering the multi-master system bus to any other bus arbiter requesting the bus through the $\overline{CBRQ}$ input pin.
RESB	I	**Resident Bus:** A strapping option to configure the arbiter to operate in systems having both a multi-master system bus and a Resident Bus. Strapped high, the multi-master system bus is requested or surrendered as a function of the SYSB/$\overline{RESB}$ input pin. Strapped low, the SYSB/$\overline{RESB}$ input is ignored.
ANYRQST	I	**Any Request:** A strapping option which permits the multi-master system bus to be surrendered to a lower priority arbiter as if it were an arbiter of higher priority (i.e., when a lower priority arbiter requests the use of the multi-master system bus, the bus is surrendered as soon as it is possible). When ANYRQST is strapped low, the bus is surrendered according to Table 2. If ANYRQST is strapped high and $\overline{CBRQ}$ is activated, the bus is surrendered at the end of the present bus cycle. Strapping $\overline{CBRQ}$ low and ANYRQST high forces the 8289 arbiter to surrender the multi-master system bus after each transfer cycle. Note that when surrender occurs BREQ is driven false (high).
$\overline{IOB}$	I	**IO Bus:** A strapping option which configures the 8289 Arbiter to operate in systems having both an IO Bus (Peripheral Bus) and a multi-master system bus. The arbiter requests and surrenders the use of the multi-master system bus as a function of the status line, $\overline{S2}$. The multi-master system bus is permitted to be surrendered while the processor is performing IO commands and is requested whenever the processor performs a memory command. Interrupt cycles are assumed as coming from the peripheral bus and are treated as an IO command.
$\overline{AEN}$	O	**Address Enable:** The output of the 8289 Arbiter to the processor's address latches, to the 8288 Bus Controller and 8284A Clock Generator. $\overline{AEN}$ serves to instruct the Bus Controller and address latches when to tri-state their output drivers.
SYSB/ $\overline{RESB}$	I	**System Bus/Resident Bus:** An input signal when the arbiter is configured in the S.R. Mode (RESB is strapped high) which determines when the multi-master system bus is requested and multi-master system bus surrendering is permitted. The signal is intended to originate from a form of address-mapping circuitry, as a decoder or PROM attached to the resident address bus. Signal transitions and glitches are permitted on this pin from ϕ1 of T4 to ϕ 1 of T2 of the processor cycle. During the period from ϕ1 of T2 to ϕ 1 of T4, only clean transitions are permitted on this pin (no glitches). If a glitch occurs, the arbiter may capture or miss it, and the multi-master system bus may be requested or surrendered, depending upon the state of the glitch. The arbiter requests the multi-master system bus in the S.R. Mode when the state of the SYSB/$\overline{RESB}$ pin is high and permits the bus to be surrendered when this pin is low.
$\overline{CBRQ}$	I/O	**Common Bus Request:** An input signal which instructs the arbiter if there are any other arbiters of lower priority requesting the use of the multi-master system bus. The $\overline{CBRQ}$ pins (open-collector output) of all the 8289 Bus Arbiters which surrender to the multi-master system bus upon request are connected together. The Bus Arbiter running the current transfer cycle will not itself pull the $\overline{CBRQ}$ line low. Any other arbiter connected to the $\overline{CBRQ}$ line can request the multi-master system bus. The arbiter presently running the current transfer cycle drops its $\overline{BREQ}$ signal and surrenders the bus whenever the proper surrender conditions exist. Strapping $\overline{CBRQ}$ low and ANYRQST high allows the multi-master system bus to be surrendered after each transfer cycle. See the pin definition of ANYRQST.
$\overline{INIT}$	I	**Initialize:** An active low multi-master system bus input signal used to reset all the bus arbiters on the multi-master system bus. After initialization, no arbiters have the use of the multi-master system bus.

Table 1. Pin Descriptions (Continued)

Symbol	Type	Name and Function
$\overline{BCLK}$	I	**Bus Clock:** The multi-master system bus clock to which all multi-master system bus interface signals are synchronized.
$\overline{BREQ}$	O	**Bus Request:** An active low output signal in the parallel Priority Resolving Scheme which the arbiter activates to request the use of the multi-master system bus.
$\overline{BPRN}$	I	**Bus Priority In:** The active low signal returned to the arbiter to instruct it that it may acquire the multi-master system bus on the next falling edge of BCLK. $\overline{BPRN}$ indicates to the arbiter that it is the highest priority requesting arbiter presently on the bus. The loss of $\overline{BPRN}$ instructs the arbiter that it has lost priority to a higher priority arbiter.

Symbol	Type	Name and Function
$\overline{BPRO}$	O	**Bus Priority Out:** An active low output signal used in the serial priority resolving scheme where $\overline{BPRO}$ is daisy-chained to $\overline{BPRN}$ of the next lower priority arbiter.
$\overline{BUSY}$	I/O	**Busy:** An active low open collector multi-master system bus interface signal used to instruct all the arbiters on the bus when the multi-master system bus is available. When the multi-master system bus is available the highest requesting arbiter (determined by $\overline{BPRN}$) seizes the bus and pulls $\overline{BUSY}$ low to keep other arbiters off of the bus. When the arbiter is done with the bus, it releases the $\overline{BUSY}$ signal, permitting it to go high and thereby allowing another arbiter to acquire the multi-master system bus.

FUNCTIONAL DESCRIPTION

The 8289 Bus Arbiter operates in conjunction with the 8288 Bus Controller to interface iAPX 86, 88 processors to a multi-master system bus (both the iAPX 86 and iAPX 88 are configured in their max mode). The processor is unaware of the arbiter's existence and issues commands as though it has exclusive use of the system bus. If the processor does not have the use of the multi-master system bus, the arbiter prevents the Bus Controller (8288), the data transceivers and the address latches from accessing the system bus (e.g. all bus driver outputs are forced into the high impedance state). Since the command sequence was not issued by the 8288, the system bus will appear as "Not Ready" and the processor will enter wait states. The processor will remain in Wait until the Bus Arbiter acquires the use of the multi-master system bus whereupon the arbiter will allow the bus controller, the data transceivers, and the address latches to access the system. Typically, once the command has been issued and a data transfer has taken place, a transfer acknowledge (XACK) is returned to the processor to indicate "READY" from the accessed slave device. The processor then completes its transfer cycle. Thus the arbiter serves to multiplex a processor (or bus master) onto a multi-master system bus and avoid contention problems between bus masters.

Arbitration Between Bus Masters

In general, higher priority masters obtain the bus when a lower priority master completes its present transfer cycle. Lower priority bus masters obtain the bus when a higher priority master is not accessing the system bus. A strapping option (ANYRQST) is provided to allow the arbiter to surrender the bus to a lower priority master as though it were a master of higher priority. If there are no other bus masters requesting the bus, the arbiter maintains the bus so long as its processor has not entered the HALT State. The arbiter will not voluntarily surrender the system bus and has to be forced off by another master's bus request, the HALT State being the only exception. Additional strapping options permit other modes of operation wherein the multi-master system bus is surrendered or requested under different sets of conditions.

Priority Resolving Techniques

Since there can be many bus masters on a multi-master system bus, some means of resolving priority between bus masters simultaneously requesting the bus must be provided. The 8289 Bus Arbiter provides several resolving techniques. All the techniques are based on a priority concept that at a given time one bus master will have priority above all the rest. There are provisions for using parallel priority resolving techniques, serial priority resolving techniques, and rotating priority techniques.

PARALLEL PRIORITY RESOLVING

The parallel priority resolving technique uses a separate bus request line ($\overline{BREQ}$) for each arbiter on the multi-master system bus, see Figure 4. Each $\overline{BREQ}$ line enters into a priority encoder which generates the binary address of the highest priority $\overline{BREQ}$ line which is active. The binary address is decoded by a decoder to select the corresponding $\overline{BPRN}$ (Bus Priority In) line to be returned to the highest priority requesting arbiter. The arbiter receiving priority ($\overline{BPRN}$ true) then allows its associated bus master onto the multi-master system bus as soon as it becomes available (i.e., the bus is no longer busy). When one bus arbiter gains priority over another arbiter it cannot immediately seize the bus, it must wait until the present bus transaction is complete.

Upon completing its transaction the present bus occupant recognizes that it no longer has priority and surrenders the bus by releasing $\overline{BUSY}$. $\overline{BUSY}$ is an active low "OR" tied signal line which goes to every bus arbiter on the system bus. When $\overline{BUSY}$ goes inactive (high), the arbiter which presently has bus priority ($\overline{BPRN}$ true) then seizes the bus and pulls $\overline{BUSY}$ low to keep other arbiters off of the bus. See waveform timing diagram, Figure 5. Note that all multi-master system bus transactions are synchronized to the bus clock (BCLK). This allows the parallel priority resolving circuitry or any other priority resolving scheme employed to settle.

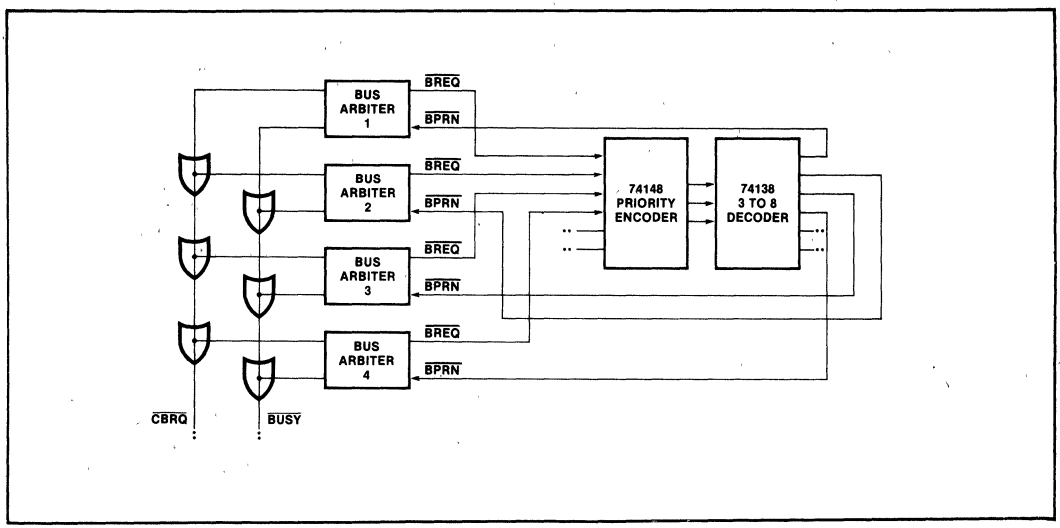


Figure 4. Parallel Priority Resolving Technique

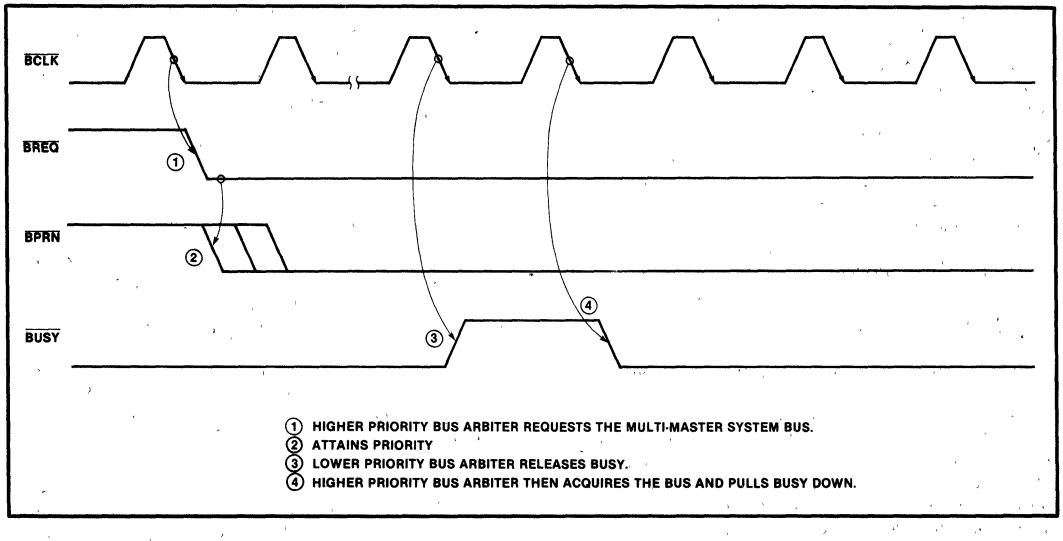


Figure 5. Higher Priority Arbiter obtaining the Bus from a Lower Priority Arbiter

SERIAL PRIORITY RESOLVING

The serial priority resolving technique eliminates the need for the priority encoder-decoder arrangement by daisy-chaining the bus arbiters together, connecting the higher priority bus arbiter's $\overline{BPRO}$ (Bus Priority Out) output to the $\overline{BPRN}$ of the next lower priority. See Figure 6.

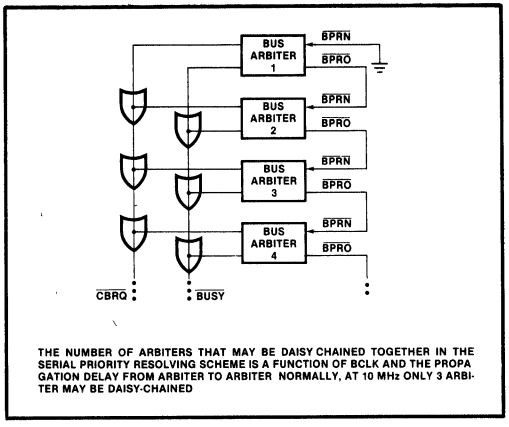


Figure 6. Serial Priority Resolving

ROTATING PRIORITY RESOLVING

The rotating priority resolving technique is similar to that of the parallel priority resolving technique except that priority is dynamically re-assigned. The priority encoder is replaced by a more complex circuit which rotates priority between requesting arbiters thus allowing each arbiter an equal chance to use the multi-master system bus, over time.

Which Priority Resolving Technique To Use

There are advantages and disadvantages for each of the techniques described above. The rotating priority resolving technique requires substantial external logic to implement while the serial technique uses no external logic but can accommodate only a limited number of bus arbiters before the daisy-chain propagation delay exceeds the multi-master's system bus clock ($\overline{BCLK}$). The parallel priority resolving technique is in general a good compromise between the other two techniques. It allows for many arbiters to be present on the bus while not requiring too much logic to implement.

8289 MODES OF OPERATION

There are two types of processors in the iAPX 86 family. An Input/Output processor (the 8089 IOP) and the iAPX 86/10, 88/10 CPUs. Consequently, there are two basic operating modes in the 8289 bus arbiter. One, the IOB (I/O Peripheral Bus) mode, permits the processor access to both an I/O Peripheral Bus and a multi-master system bus. The second, the RESB (Resident Bus mode), permits the processor to communicate over both a Resident Bus and a multi-master system bus. An I/O Peripheral Bus is a bus where all devices on that bus, including memory, are treated as I/O devices and are addressed by I/O commands. All memory commands are directed to another bus, the multi-master system bus. A Resident Bus can issue both memory and I/O commands, but it is a distinct and separate bus from the multi-master system bus. The distinction is that the Resident Bus has only one master, providing full availability and being dedicated to that one master.

The $\overline{IOB}$ strapping option configures the 8289 Bus Arbiter into the $\overline{IOB}$ mode and the strapping option RESB configures it into the RESB mode. It might be noted at this point that if both strapping options are strapped false, the arbiter interfaces the processor to a multi-master system bus only (see Figure 7). With both options strapped true, the arbiter interfaces the processor to a multi-master system bus, a Resident Bus, and an I/O Bus.

In the $\overline{IOB}$ mode, the processor communicates and controls a host of peripherals over the Peripheral Bus. When the I/O Processor needs to communicate with system memory, it does so over the system memory bus. Figure 8 shows a possible I/O Processor system configuration.

The iAPX 86 and iAPX 88 processors can communicate with a Resident Bus and a multi-master system bus. Two bus controllers and only one Bus Arbiter would be needed in such a configuration as shown in Figure 9. In such a system configuration the processor would have access to memory and peripherals of both busses. Memory mapping techniques are applied to select which bus is to be accessed. The SYSB/RESB input on the arbiter serves to instruct the arbiter as to whether or not the system bus is to be accessed. The signal connected to SYSB/RESB also enables or disables commands from one of the bus controllers.

A summary of the modes that the 8289 has, along with its response to its status lines inputs, is summarized in Table 2.

*In some system configurations it is possible for a non-I/O Processor to have access to more than one Multi-Master System Bus, see 8289 Application Note.

Table 2. Summary of 8289 Modes, Requesting and Relinquishing the Multi-Master System Bus

	Status Lines From 8086 or 8088 or 8089			IOB Mode Only	RESB (Mode) Only IOB = High RESB = High		IOB Mode RESB Mode IOB = Low RESB = High		Single Bus Mode IOB = High RESB = Low
	$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	$\overline{IOB}$ = Low	SYSB/RESB = High	SYSB/$\overline{RESB}$ = Low	SYSB/$\overline{RESB}$ = High	SYSB/$\overline{RESB}$ = Low	
I/O COMMANDS	0	0	0	x		x	x	x	
	0	0	1	x		x	x	x	
	0	1	0	x		x	x	x	
HALT	0	1	1	x	x	x	x	x	x
MEM COMMANDS	1	0	0			x		x	
	1	0	1			x		x	
	1	1	0			x		x	
IDLE	1	1	1	x	x	x	x	x	x

NOTES:

1. X = Multi-Master System Bus is allowed to be Surrendered.
2. ↙ = Multi-Master System Bus is Requested.

Mode	Pin Strapping	Multi-Master System Bus	
		Requested**	Surrendered*
Single Bus Multi-Master Mode	$\overline{IOB}$ = High RESB = Low	Whenever the processor's status lines go active	HLT + TI • CBRQ + HPBRQ†
RESB Mode Only	$\overline{IOB}$ = High RESB = High	SYSB/$\overline{RESB}$ = High • ACTIVE STATUS	(SYSB/$\overline{RESB}$ = Low + TI) • CBRQ + HLT + HPBRQ
IOB Mode Only	$\overline{IOB}$ = Low RESB = Low	Memory Commands	(I/O Status + TI) • CBRQ + HLT + HPBRQ
IOB Mode·RESB Mode	$\overline{IOB}$ = Low RESB = High	(Memory Command) • (SYSB/$\overline{RESB}$ = High)	((I/O Status Commands) + SYSB/$\overline{RESB}$ = LOW)) • CBRQ + HPBRQ† + HLT

NOTES:

*$\overline{LOCK}$ prevents surrender of Bus to any other arbiter, $\overline{CRQLCK}$ prevents surrender of Bus to any lower priority arbiter.

**Except for HALT and Passive or IDLE Status.

†HPBRQ, Higher priority Bus request or $\overline{BPRN}$ = 1.

1. $\overline{IOB}$ Active Low.
2. RESB Active High.
3. + is read as "OR" and • as "AND."
4. TI = Processor Idle Status $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ = 111
5. HLT = Processor Halt Status $\overline{S2}$, $\overline{S1}$, $\overline{S0}$ = 011

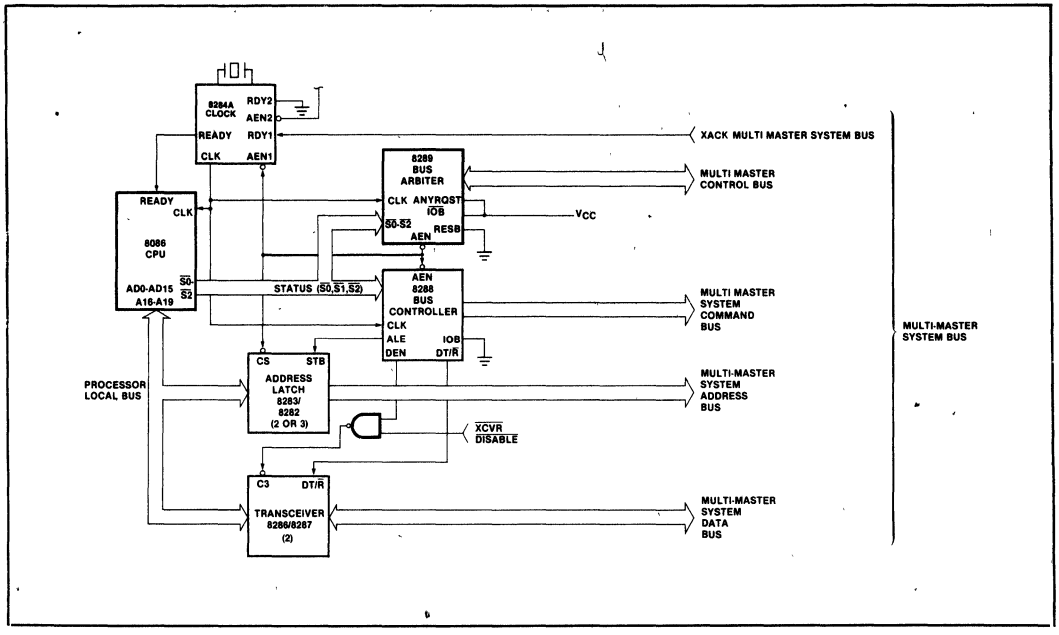


Figure 7. Typical Medium Complexity CPU System

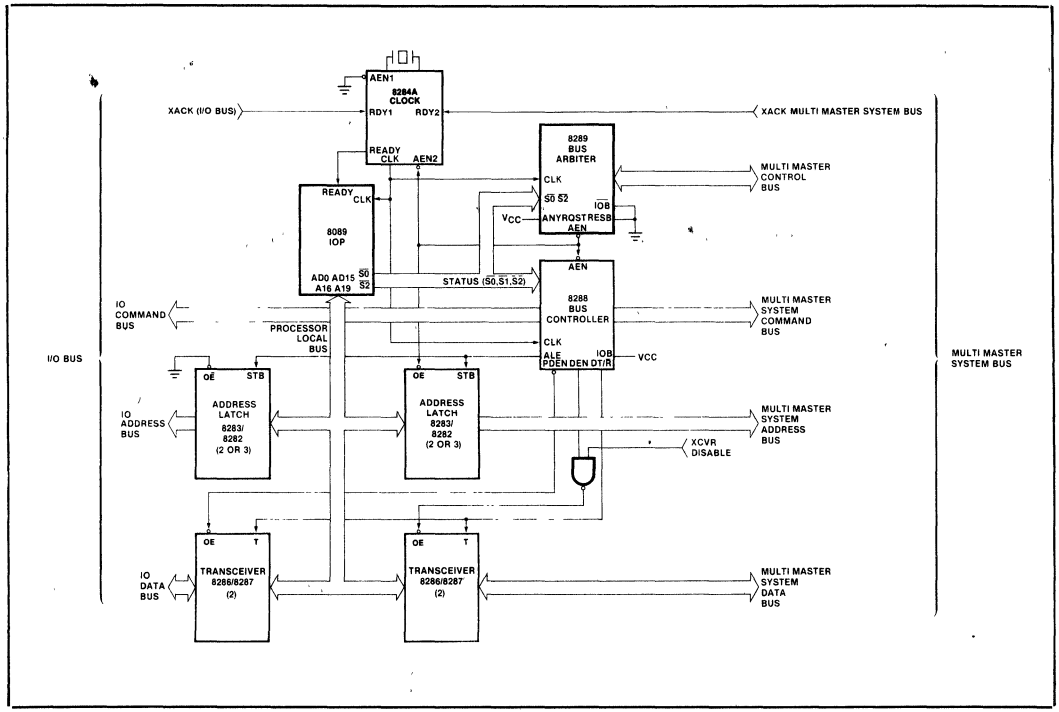
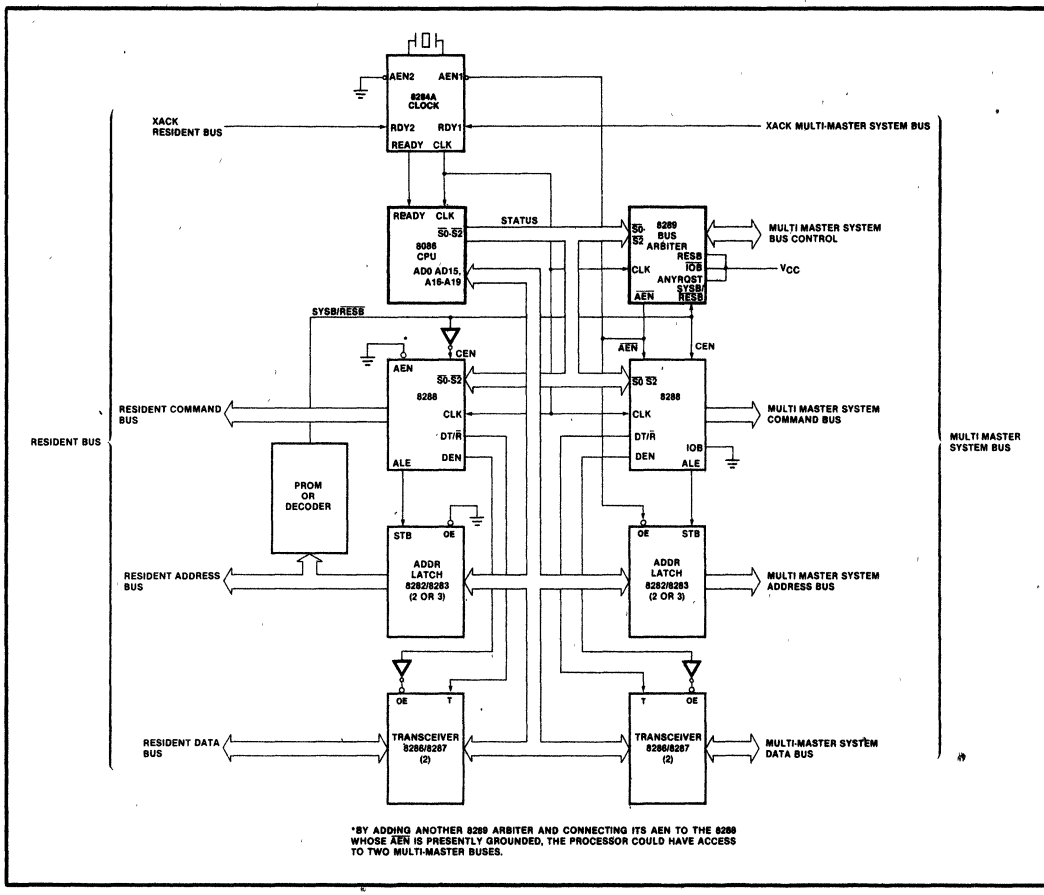


Figure 8. Typical Medium Complexity IOB System



*BY ADDING ANOTHER 8289 ARBITER AND CONNECTING ITS AEN TO THE 8288 WHOSE $\overline{AEN}$ IS PRESENTLY GROUNDED, THE PROCESSOR COULD HAVE ACCESS TO TWO MULTI-MASTER BUSES.

Figure 9. 8289 Bus Arbiter Shown in System-Resident Bus Configuration

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .....0°C to 70°C
Storage Temperature.....–65°C to +150°C
All Output and Supply Voltages.....–0.5V to +7V
All Input Voltages.....–1.0V to +5.5V
Power Dissipation.....1.5 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ±10%)

Symbol	Parameter	Min.	Max.	Units	Test Condition
V_C	Input Clamp Voltage		–1.0	V	V_{CC} = 4.50V, I_C = –5 mA
I_F	Input Forward Current		–0.5	mA	V_{CC} = 5.50V, V_F = 0.45V
I_R	Reverse Input Leakage Current		60	μA	V_{CC} = 5.50, V_R = 5.50
V_{OL}	Output Low Voltage				
	$\overline{BUSY}$, $\overline{CBRQ}$		0.45	V	I_{OL} = 20 mA
	$\overline{AEN}$		0.45	V	I_{OL} = 16 mA
	$\overline{BPRO}$, $\overline{BREQ}$		0.45	V	I_{OL} = 10 mA
V_{OH}	Output High Voltage $\overline{BUSY}$, $\overline{CBRQ}$	Open Collector			
	All Other Outputs	2.4		V	I_{OH} = 400 μA
I_{CC}	Power Supply Current		165	mA	
V_{IL}	Input Low Voltage		.8	V	
V_{IH}	Input High Voltage	2.0		V	
Cin Status	Input Capacitance		25	pF	
Cin (Others)	Input Capacitance		12	pF	

A.C. CHARACTERISTICS (V_{CC} = +5V ±10%, T_A = 0°C to 70°C)

TIMING REQUIREMENTS

Symbol	Parameter	8289 Min.	8289-1 Min.	Max.	Unit	Test Condition
TCLCL	CLK Cycle Period	125	100		ns	
TCLCH	CLK Low Time	65	53		ns	
TCHCL	CLK High Time	35	26		ns	
TSVCH	Status Active Setup	65	55	TCLCL-10	ns	
TSHCL	Status Inactive Setup	50	45	TCLCL-10	ns	
THVCH	Status Active Hold	10	10		ns	
THVCL	Status Inactive Hold	10	10		ns	
TBYSBL	BUSY↑↓Setup to BCLK↓	20	20		ns	
TCBSBL	CBRQ↑↓Setup to BCLK↓	20	20		ns	
TBLBL	BCLK Cycle Time	100	100		ns	
TBHCL	BLCK High Time	30	30	.65[TBLBL]	ns	
TCLLL1	LOCK Inactive Hold	10	10		ns	
TCLLL2	LOCK Active Setup	40	40		ns	
TPNBL	BPRN↑↓ to BCLK Setup Time	15	15		ns	
TCLSR1	SYSB/RESB Setup	0	0		ns	
TCLSR2	SYSB/RESB Hold	20	20		ns	
TIVIH	Initialization Pulse Width	3 TBLBL+ 3 TCLCL	3 TBLBL+ 3 TCLCL		ns	

A.C. CHARACTERISTICS (Continued)

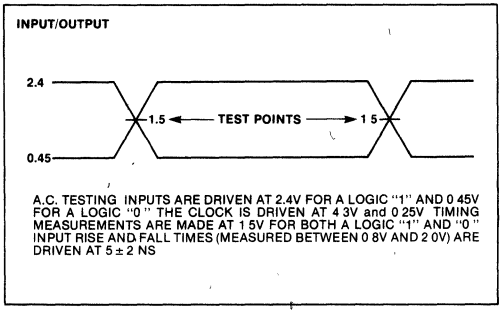
TIMING RESPONSES

Symbol	Parameter	Min.	Max.	Unit	Test Condition
TBLBRL	BCLK to BREQ Delay↓↑		35	ns	
TBLPOH	BCLK to BPRO↓↑ (See Note 1)		40	ns	
TPNPO	BPRN↓↑to BPRO↓↑Delay (See Note 1)		25	ns	
TBLBYL	BCLK to BUSY Low		60	ns	
TBLBYH	BCLK to BUSY Float (See Note 2)		35	ns	
TCLAEH	CLK to AEN High		65	ns	
TBLAEL	BCLK to AEN Low		40	ns	
TBLCBL	BCLK to CBRQ Low		60	ns	
TRLCRH	BCLK to CBRQ Float (See Note 2)		35	ns	
TOLOH	Output Rise Time		20	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12	ns	From 2.0V to 0.8V

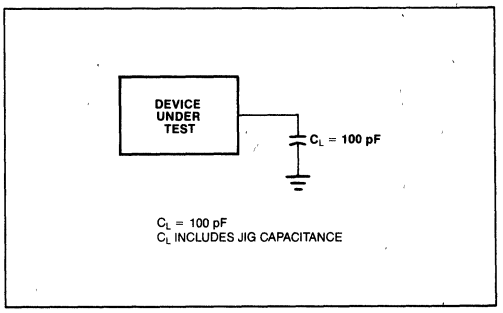
↓↑ Denotes that spec applies to both transitions of the signal.

NOTES:
1. BCLK generates the first BPRO wherein subsequent BPRO changes lower in the chain are generated through BPRON.
2. Measured at .5V above GND.

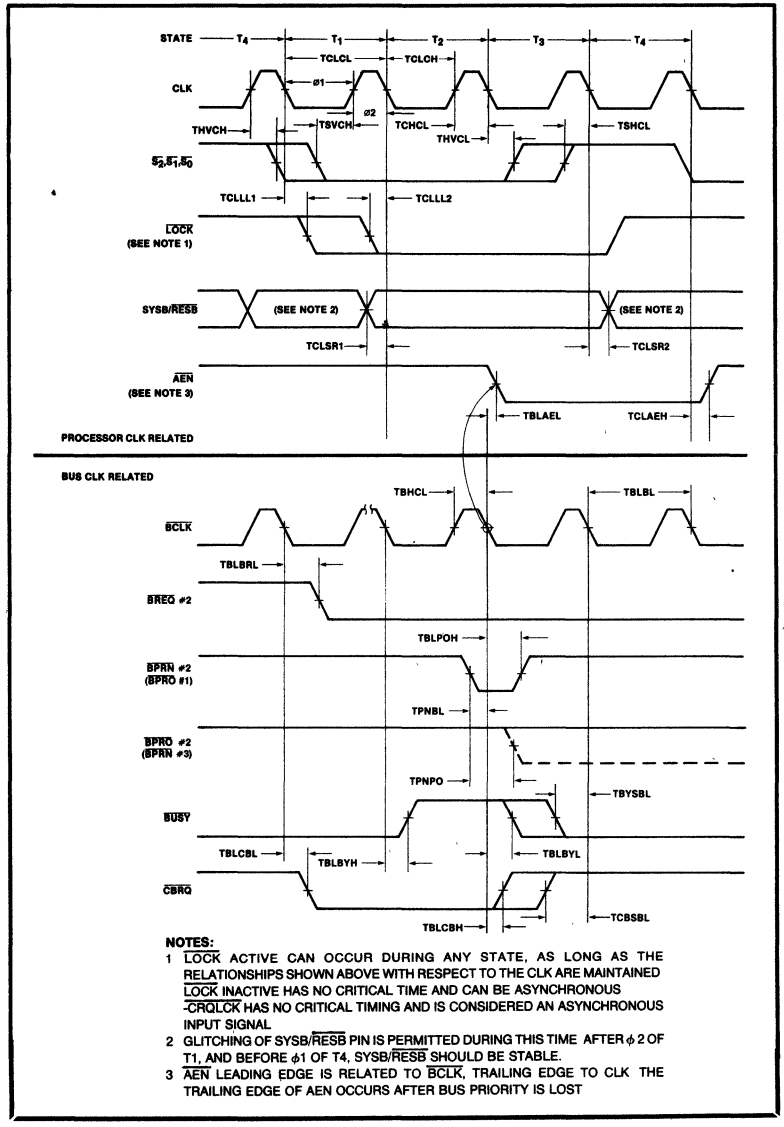
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



WAVEFORMS



NOTES:

1 LOCK ACTIVE CAN OCCUR DURING ANY STATE, AS LONG AS THE RELATIONSHIPS SHOWN ABOVE WITH RESPECT TO THE CLK ARE MAINTAINED
LOCK INACTIVE HAS NO CRITICAL TIME AND CAN BE ASYNCHRONOUS
-CRQLCK HAS NO CRITICAL TIMING AND IS CONSIDERED AN ASYNCHRONOUS INPUT SIGNAL

2 GLITCHING OF SYSB/RESB PIN IS PERMITTED DURING THIS TIME AFTER ϕ2 OF T1, AND BEFORE ϕ1 OF T4, SYSB/RESB SHOULD BE STABLE.

3 AEN LEADING EDGE IS RELATED TO BCLK, TRAILING EDGE TO CLK THE TRAILING EDGE OF AEN OCCURS AFTER BUS PRIORITY IS LOST

ADDITIONAL NOTES:

The signals related to CLK are typical processor signals, and do not relate to the depicted sequence of events of the signals referenced to BCLK. The signals shown related to the BCLK represent a hypothetical sequence of events for illustration. Assume 3 bus arbiters of priorities 1, 2 and 3 configured in serial priority resolving scheme as shown in Figure 6. Assume arbiter 1 has the bus and is holding busy low. Arbiter #2 detects its processor wants the bus and pulls low BREQ#2. If BPRN#2 is high (as shown), arbiter #2 will pull low CBRQ line. CBRQ signals to the higher priority arbiter #1 that a lower priority arbiter wants the bus. [A higher priority arbiter would be granted BPRN when it makes the bus request rather than having to wait for another arbiter to release the bus through CBRQ].** Arbiter #1 will relinquish the multi-master system bus when it enters a state not requiring it (see Table 1), by lowering its BPRO#1 (tied to BPRN#2) and releasing BUSY. Arbiter #2 now sees that it has priority from BPRN#2 being low and releases CBRQ. As soon as BUSY signifies the bus is available (high), arbiter #2 pulls BUSY low on next falling edge of BCLK. Note that if arbiter #2 didn't want the bus at the time it received priority, it would pass priority to the next lower priority arbiter by lowering its BPRO #2 [TPNPO].

**Note that even a higher priority arbiter which is acquiring the bus through BPRN will momentarily drop CBRQ until it has acquired the bus

iAPX 286 Microprocessors

4

Microprocessors Section

intel®

ADVANCE INFORMATION

iAPX 286/10
HIGH PERFORMANCE MICROPROCESSOR WITH MEMORY MANAGEMENT AND PROTECTION

(80286, 80286-6, 80286-4)

- **High Performance Processor (Up to six times iAPX 86)**
- **Large Address Space:**
  - **—16 Megabytes Physical**
  - **—1 Gigabyte Virtual per Task**
- **Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems**
- **Two iAPX 86 Upward Compatible Operating Modes:**
  - **—iAPX 86 Real Address Mode**
  - **—Protected Virtual Address Mode**
- **Range of clock rates**
  - **—8 MHz for 80286**
  - **—6 MHz for 80286-6**
  - **—4 MHz for 80286-4**
- **Optional Processor Extension:**
  - **—iAPX 286/20 High Performance 80-bit Numeric Data Processor**
- **Complete System Development Support:**
  - **—Development Software: Assembler, PL/M, Pascal, FORTRAN, and System Utilities**
  - **—In-Circuit-Emulator (ICE™-286)**
- **High Bandwidth Bus Interface (8 Megabyte/Sec)**
- **Available in EXPRESS:**
  - **—Standard Temperature Range**

The iAPX 286/10 (80286 part number) is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. An 8 MHz iAPX 286/10 provides up to six times greater throughout than the standard 5 MHz iAPX 86/10. The 80286 includes memory management capabilities that map up to 2^{30} (one gigabyte) of virtual address space per task into 2^{24} bytes (16 megabytes) of physical memory.

The iAPX 286 is upward compatible with iAPX 86 and 88 software. Using iAPX 86 real address mode, the 80286 is object code compatible with existing iAPX 86, 88 software. In protected virtual address mode, the 80286 is source code compatible with iAPX 86, 88 software and may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection mechanism. Both modes operate at full 80286 performance and execute a superset of the iAPX 86 and 88's instructions.

The 80286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

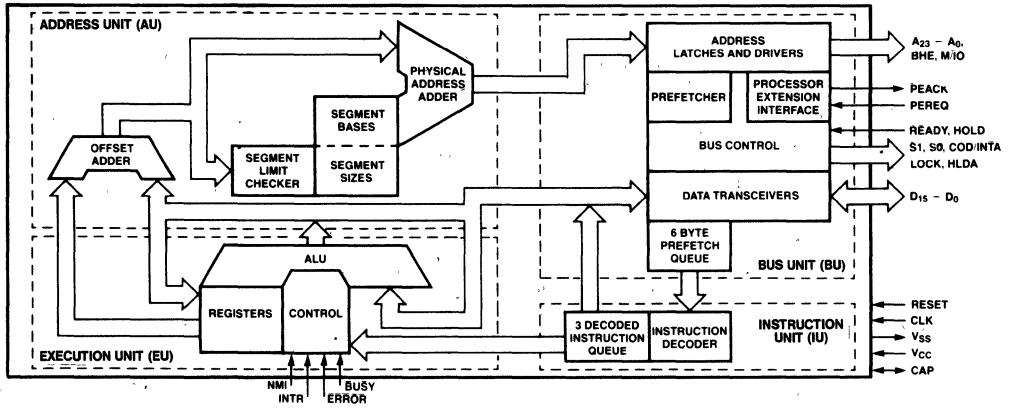
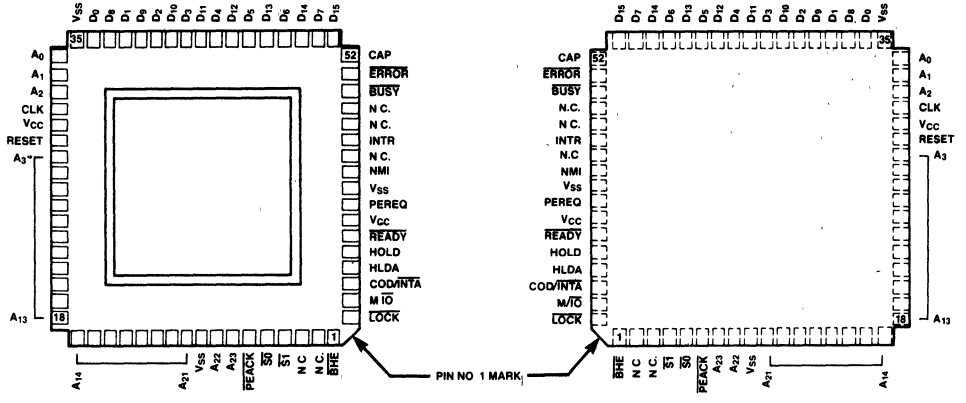


Figure 1. 80286 Internal Block Diagram

NOVEMBER 1983
ORDER NUMBER: 210253-006

Component Pad View—As viewed from underside of component when mounted on the board.

P.C. Board View—As viewed from the component side of the P.C. board.



NOTE: N.C. pads must not be connected.

Figure 2. 80286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80286 microprocessor:

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the fundamental timing for iAPX 286 systems. It is divided by two inside the 80286 to generate the processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.
D_{15}-D_0	I/O	**Data Bus** inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
A_{23}-A_0	O	**Address Bus** outputs physical memory and I/O port addresses. A0 is LOW when data is to be transferred on pins D_{7-0}. A_{23}-A_{16} are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
$\overline{BHE}$	O	**Bus High Enable** indicates transfer of data on the upper byte of the data bus, D_{15-8}. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.

$\overline{BHE}$ and A0 Encodings

$\overline{BHE}$ Value	A0 Value	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus (D_{15-8})
1	0	Byte transfer on lower half of data bus (D_{7-0})
1	1	Reserved

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
$\overline{S1}$, $\overline{S0}$	O	**Bus Cycle Status** indicates initiation of a bus cycle and, along with M/$\overline{IO}$ and COD/$\overline{INTA}$, defines the type of bus cycle. The bus is in a T_S state whenever one or both are LOW. $\overline{S1}$ and $\overline{S0}$ are active LOW and float to 3-state OFF during bus hold acknowledge.

80286 Bus Cycle Status Definition

COD/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Bus cycle initiated
0 (LOW)	0	0	0	Interrupt acknowledge
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	None; not a status cycle
0	1	0	0	IF A1 = 1 then halt; else shutdown
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle
1 (HIGH)	0	0	0	Reserved
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	None, not a status cycle
1	1	0	0	Reserved
1	1	0	1	Memory instruction read
1	1	1	0	Reserved
1	1	1	1	None; not a status cycle

M/$\overline{IO}$	O	**Memory/IO Select** distinguishes memory access from I/O access. If HIGH during T_S, a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/$\overline{IO}$ floats to 3-state OFF during bus hold acknowledge.
COD/$\overline{INTA}$	O	**Code/Interrupt Acknowledge** distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{INTA}$ floats to 3-state OFF during bus hold acknowledge. Its timing is the same as M/$\overline{IO}$.
$\overline{LOCK}$	O	**Bus Lock** indicates that other system bus masters are not to gain control of the system bus following the current bus cycle. The $\overline{LOCK}$ signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. $\overline{LOCK}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.
$\overline{READY}$	I	**Bus Ready** terminates a bus cycle. Bus cycles are extended without limit until terminated by $\overline{READY}$ LOW. $\overline{READY}$ is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. $\overline{READY}$ is ignored during bus hold acknowledge.
HOLD HLDA	I O	**Bus Hold Request and Hold Acknowledge** control ownership of the 80286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80286 will float its bus drivers to 3-state OFF and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.
INTR	I	**Interrupt Request** requests the 80286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	I	**Non-maskable Interrupt Request** interrupts the 80286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
PEREQ PEACK	I O	**Processor Extension Operand Request and Acknowledge** extend the memory management and protection capabilities of the 80286 to processor extensions. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and floats to 3-state OFF during bus hold acknowledge. PEACK may be asynchronous to the system clock. PEACK is active LOW.
BUSY ERROR	I I	**Processor Extension Busy and Error** indicate the operating condition of a processor extension to the 80286. An active BUSY input stops 80286 program execution on WAIT and some ESC instructions until BUSY becomes inactive (HIGH). The 80286 may be interrupted while waiting for BUSY to become inactive. An active ERROR input causes the 80286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock.
RESET	I	**System Reset** clears the internal logic of the 80286 and is active HIGH. The 80286 may be re-initialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80286 enter the state shown below: **80286 Pin State During Reset** **Pin Value** / **Pin Names** 1 (HIGH) / $\overline{S0}$, $\overline{S1}$, PEACK, A23-A0, $\overline{BHE}$, $\overline{LOCK}$ 0 (LOW) / M/$\overline{IO}$, COD/$\overline{INTA}$, HLDA 3-state OFF / D_{15}–D_0 Operation of the 80286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed. A LOW to HIGH transition of RESET synchronous to the system clock will end a processor cycle at the second HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it cannot be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are required only for systems where the processor clock must be phase synchronous to another clock.
V_{SS}	I	**System Ground:** 0 Volts.
V_{CC}	I	**System Power:** +5 Volt Power Supply.
CAP	I	**Substrate Filter Capacitor:** a 0.047μf ± 20% 12V capacitor must be connected between this pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum DC leakage current of 1 μa is allowed through the capacitor. For correct operation of the 80286, the substrate bias generator must charge this capacitor to its operating voltage. The capacitor chargeup time is 5 milliseconds (max.) after V_{CC} and CLK reach their specified AC and DC parameters. RESET may be applied to prevent spurious activity by the CPU during this time. After this time, the 80286 processor clock can be phase synchronized to another clock by pulsing RESET LOW synchronous to the system clock.

FUNCTIONAL DESCRIPTION

Introduction

The 80286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, the 80286's performance is up to six times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's iAPX 86, 88, and 186 family of CPU's.

The 80286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. Both modes execute a superset of the iAPX 86 and 88 instruction set.

In iAPX 86 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80286 architecture common to both modes, second, iAPX 86 real address mode, and third, protected mode.

iAPX 286/10 BASE ARCHITECTURE

The iAPX 86, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's.

Register Set

The 80286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: The 3 16-bit special purpose registers in figure 3A record or control certain aspects of the 80286 processor state including the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

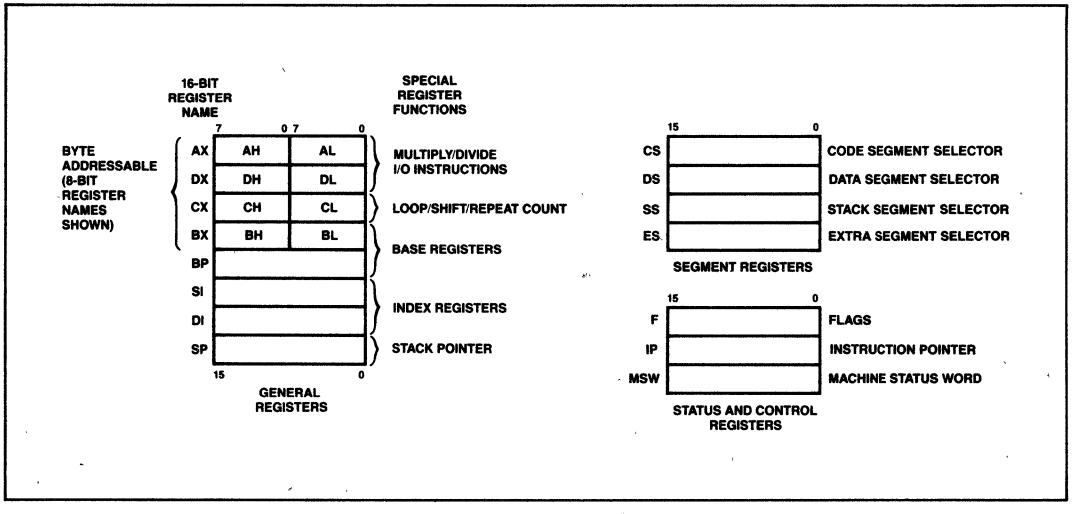


Figure 3. Register Set

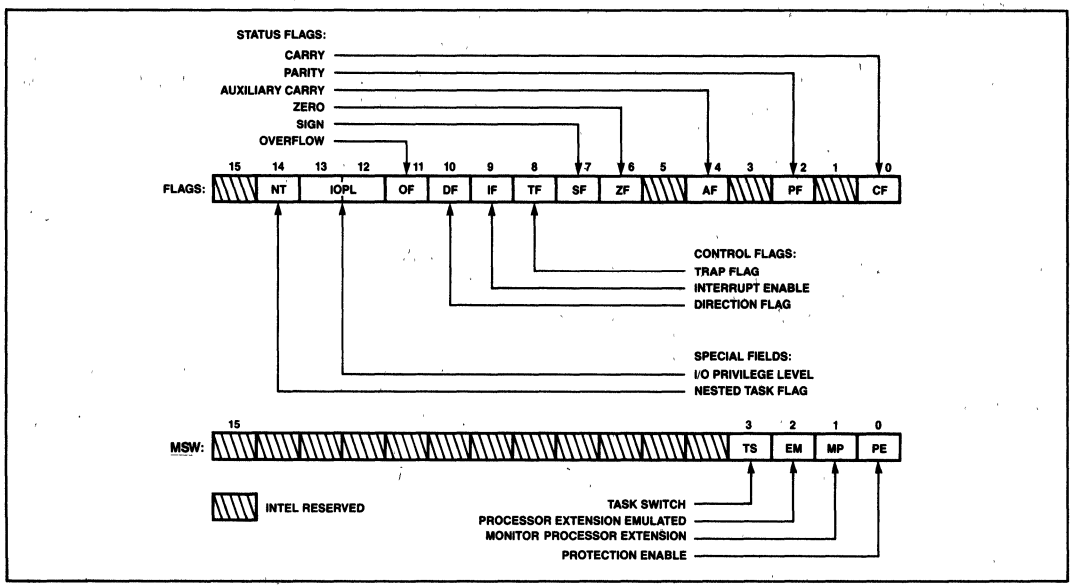


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

—Register to Register
—Memory to Register
—Immediate to Register
—Memory to Memory
—Register to Memory
—Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag–Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto decrement the appropriate index registers when set. Clearing DF causes auto increment.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

For detailed operation and usage of each instruction, see Appendix of iAPX 286 Programmer's Reference Manual (Order No. 210498)

GENERAL PURPOSE	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
INPUT/OUTPUT	
IN	Input byte or word
OUT	Output byte or word
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
FLAG TRANSFER	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

ADDITION	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
MULTIPLICATION	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
DIVISION	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

LOGICALS	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate/Logical Instructions

CONDITIONAL TRANSFERS		UNCONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal	CALL	Call procedure
JAE/JNB	Jump if above or equal/not below	RET	Return from procedure
JB/JNAE	Jump if below/not above nor equal	JMP	Jump
JBE/JNA	Jump if below or equal/not above		
JC	Jump if carry	**ITERATION CONTROLS**	
JE/JZ	Jump if equal/zero		
JG/JNLE	Jump if greater/not less nor equal	LOOP	Loop
JGE/JNL	Jump if greater or equal/not less	LOOPE/LOOPZ	Loop if equal/zero
JL/JNGE	Jump if less/not greater nor equal	LOOPNE/LOOPNZ	Loop if not equal/not zero
JLE/JNG	Jump if less or equal/not greater	JCXZ	Jump if register CX = 0
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal/not zero	**INTERRUPTS**	
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity/parity odd	INT	Interrupt
JNS	Jump if not sign	INTO	Interrupt if overflow
JO	Jump if overflow	IRET	Interrupt return
JP/JPE	Jump if parity/parity even		
JS	Jump if sign		

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{BUSY}}$ not active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

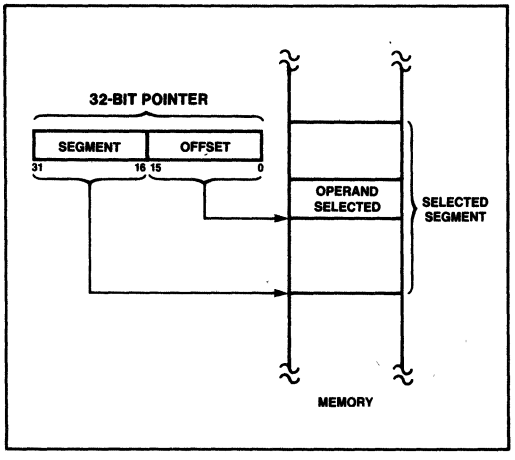


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands not residing in one of the four immediately available segments, a full 32-bit pointer or a new segment selector must be loaded.

Addressing Modes

The 80286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode. The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

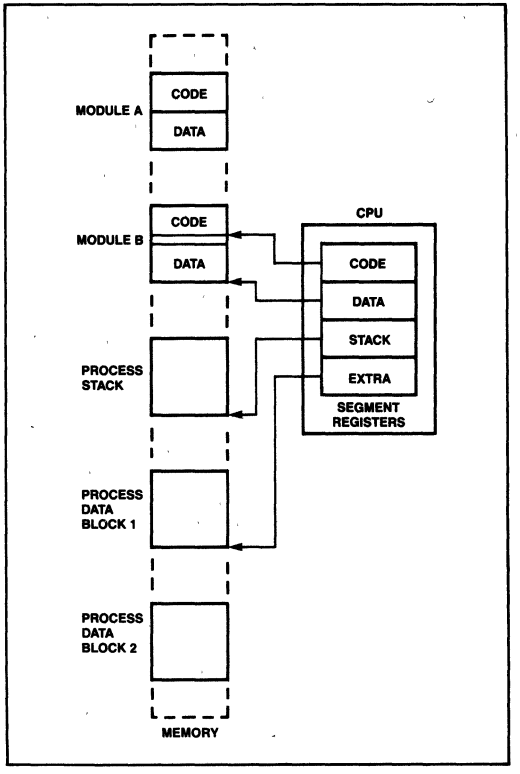


Figure 6. Segmented Memory Helps Structure Software

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Indexed Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80286 directly supports the following data types:

Integer: A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the iAPX 286/20 Numeric Data Processor.

Ordinal: An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.

Pointer: A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.

String: A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.

ASCII: A byte representation of alphanumeric and control characters using the ASCII standard of character representation.

BCD: A byte (unpacked) representation of the decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.

Floating Point: A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the iAPX 286/20 Numeric Processor configuration.)

Figure 7 graphically represents the data types supported by the iAPX 286.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

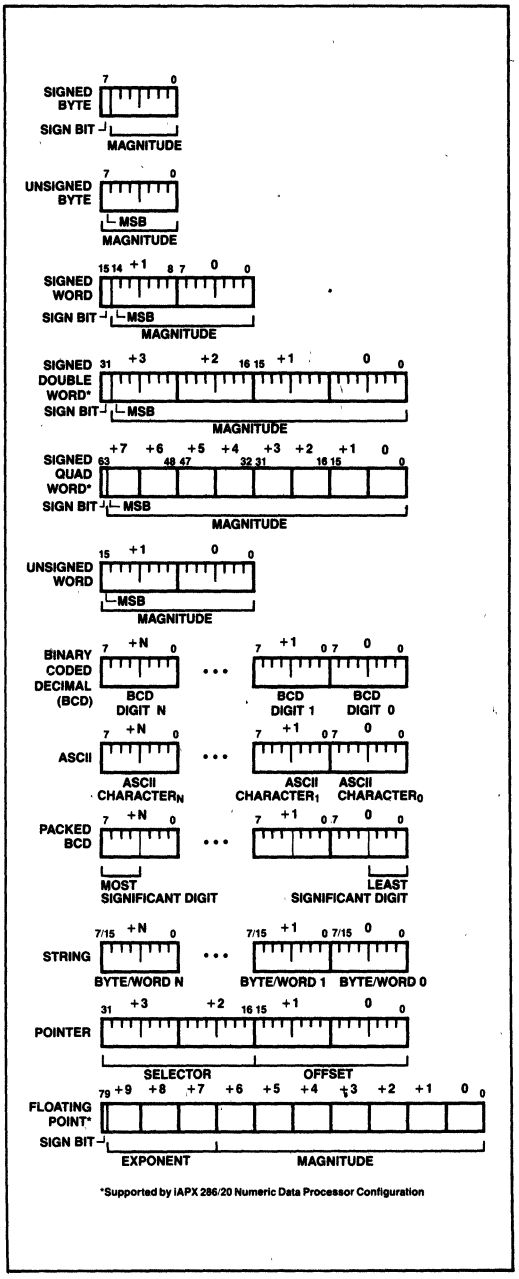


Figure 7. iAPX 286 Supported Data Types

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Does Return Address Point to Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	INT 2 or NMI pin	
Breakpoint interrupt	3	INT 3	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT	Yes
Intel reserved—do not use	8–15		
Processor extension error interrupt	16	ESC or WAIT	
Intel reserved—do not use	17–31		
User defined	32–255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

SINGLE STEP INTERRUPT

The 80286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	Instruction exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR
6	INT instruction

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown as shown in Table 6.

Table 6. 80286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80286 in iAPX 86 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80286 into protected mode and can not be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception 7
0	0	0	Initial encoding after RESET. iAPX 286 operation is identical to iAPX 86,88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The Exception 7 on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

iAPX 86 REAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the iAPX 286/10 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} may be ignored.

Memory Addressing

In real address mode physical memory is a contiguous array of up to 1,048,576 bytres (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} may be ignored.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address formation.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlayed by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80286 reserves two fixed areas of memory in real address mode (see Figure 9); system initialization area and interrupt table area. Locations from addresses FFFF0(H) thorugh FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.

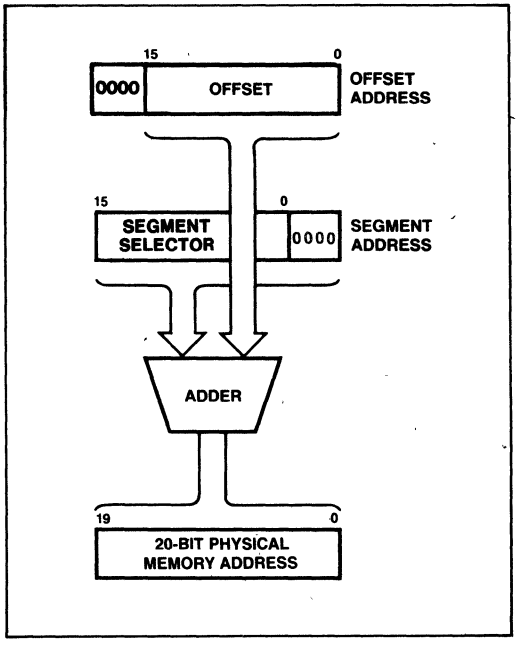


Figure 8. iAPX 86 Real Address Mode Address Calculation

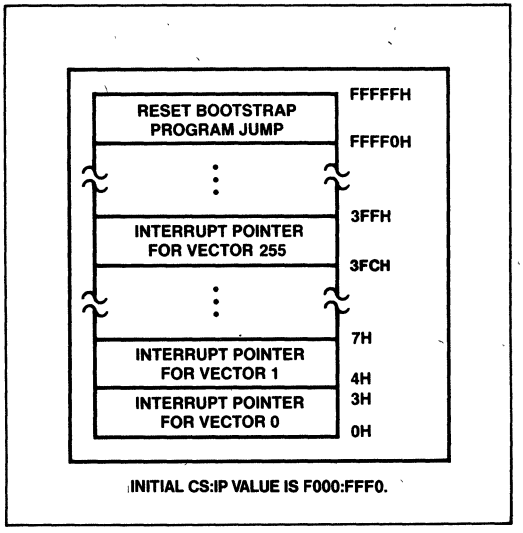


Figure 9. iAPX 86 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with iAPX 86, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the iAPX 286/10 Base Architecture section of this Functional Description remain the same. Programs for the iAPX 86, 88, 186, and real address mode 80286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pins A_{23}–A_0 and $\overline{BHE}$. The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the

desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All iAPX 286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

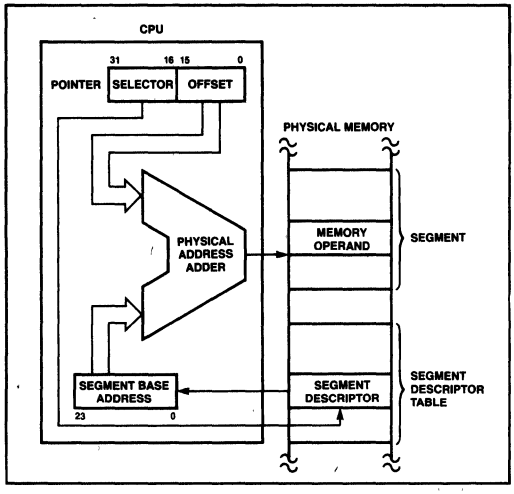


Figure 10. Protected Mode Memory Addressing

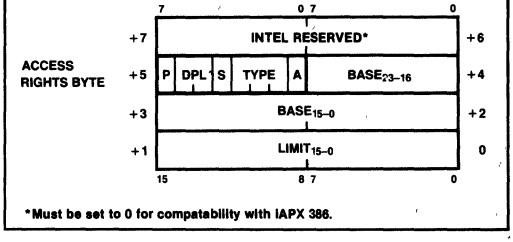
DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS (S = 1)

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.

Code or Data Segment Descriptor



*Must be set to 0 for compatability with IAPX 386.

Access Rights Byte Definition

	Bit Position	Name		Function	
	7	Present (P)	P = 1 P = 0	Segment is mapped into physical memory. No mapping to physical memory exists, base and limit are not used.	
	6-5	Descriptor Privilege Level (DPL)		Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1 S = 0	Code or Data (includes stacks) segment descriptor System Segment Descriptor or Gate Descriptor	
Type Field Definition	3	Executable (E)	E = 0	Data segment descriptor type is:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0 ED = 1	Expand up segment, offsets must be ≤ limit. Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 W = 1	Data segment may not be written into. Data segment may be written into.	
	3	Executable (E)	E = 1	Code Segment Descriptor type is:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0 R = 1	Code segment may not be read. Code segment may be read.	
	0	Accessed (A)	A = 0 A = 1	Segment has not been accessed. Segment selector has been loaded into segment register or used by selector test instructions.	

Figure 11. Code and Data Segment Descriptor Formats

Code and data (including stack data) are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors (S = 1). Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If P = 0, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL controls when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments, and downwards (ED = 1) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

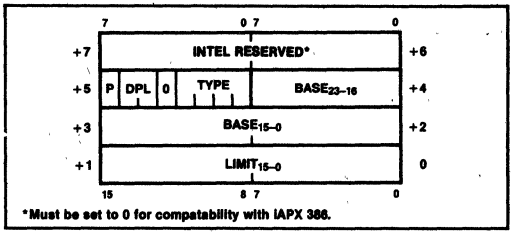
A code segment (S=1, E=1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R=0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below). The limit field identifies the last byte of a code segment.

SYSTEM SEGMENT DESCRIPTORS (S = 0, TYPE = 1-3)

In addition to code and data segment descriptors, the protected mode 80286 defines System Segment Descriptors. These descriptors define special system data segments which contain a table of descriptors (Local Descriptor Table Descriptor) or segments which contain the execution state of a task (Task State Segment Descriptor).

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if P = 1. If P = 0, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descriptor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it

System Segment Descriptor



System Segment Descriptor Fields

Name	Value	Description
TYPE	1 2 3	Available Task State Segment (TSS) Local Descriptor Table Busy Task State Segment (TSS)
P	0 1	Descriptor contents are not valid Descriptor contents are valid
DPL	0–3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. System Segment Descriptor Format

is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

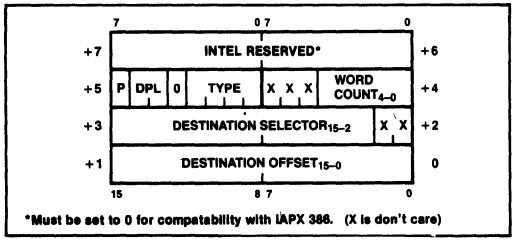
GATE DESCRIPTORS (S = 0, TYPE = 4-7)

Gates are used to control access to entry points within the target code segment. The gate descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct de-

Gate Descriptor



Gate Descriptor Fields

Name	Value	Description
TYPE	4 5 6 7	–Call Gate –Task Gate –Interrupt Gate –Trap Gate
P	0 1	–Descriptor Contents are not valid –Descriptor Contents are valid
DPL	0–3	Descriptor Privilege Level
WORD COUNT	0–31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

scriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0–31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

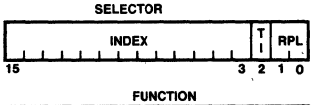
The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing the descriptor. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow high-speed testing of the selector's privilege attribute (refer to privilege discussion below).



BITS	NAME	FUNCTION
1–0	REQUESTED PRIVILEGE LEVEL (RPL)	INDICATES SELECTOR PRIVILEGE LEVEL DESIRED
2	TABLE INDICATOR (TI)	TI = 0 USE GLOBAL DESCRIPTOR TABLE (GDT) TI = 1 USE LOCAL DESCRIPTOR TABLE (LDT)
15–3	INDEX	SELECT DESCRIPTOR ENTRY IN TABLE

Figure 15. Selector Fields

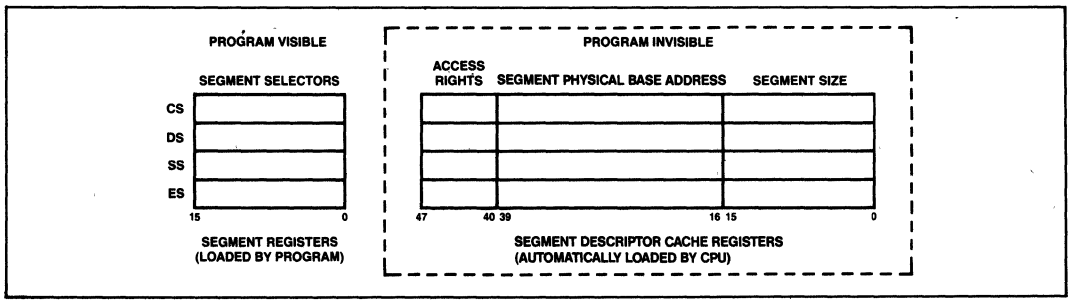


Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor Table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

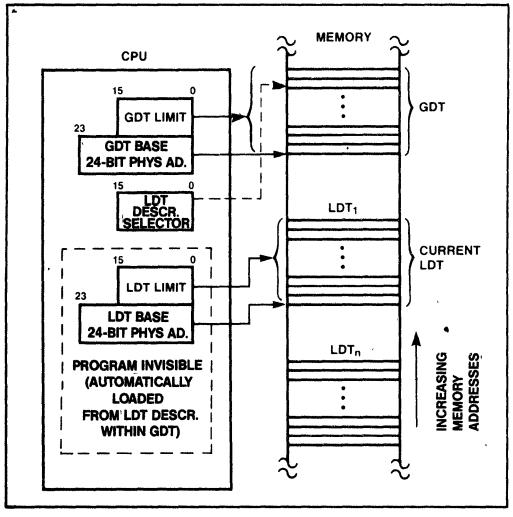


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are privileged, i.e. they may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit physical base address of the Global Descriptor Table as shown in Figure 17. The LDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the base address and limit for an LDT, as shown in Figure 12.

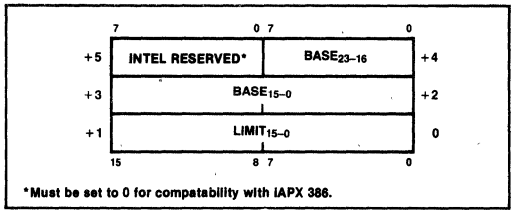


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit physical base and 16-bit limit register in the CPU. The privileged LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

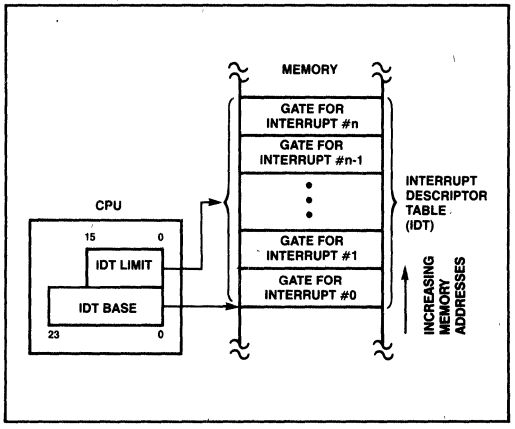


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the most privileged level. Privilege

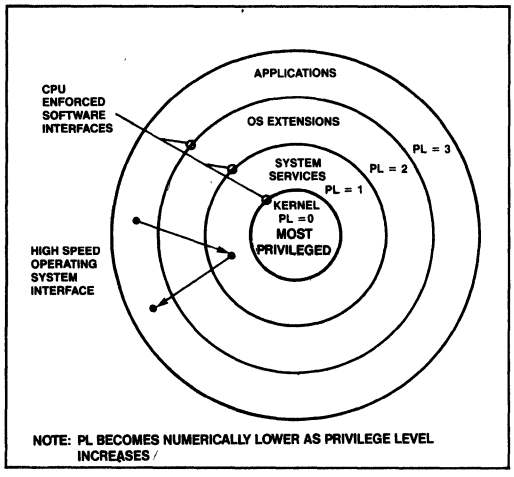


Figure 19. Hierarchical Privilege Levels

levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Each task in the system has a separate stack for each of its privilege levels.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment selector within TSS when the task is initiated via a task switch operation (See Figure 20). A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing a Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate descriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Reference to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

—JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.

—interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.

—return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.

—task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0
**NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These protection mechanisms are grouped into three forms:

Restricted usage of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted access to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). Precisely these are:

- The IF bit is not changed if CPL > IOPL.
- The IOPL field of the flag word is not changed if CPL > 0.

No exceptions or other indication are given when these conditions occur.

Table 11
Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12 Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded[1]	12 or 13

Note 1: Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
CPL ≠ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPL > IOPL when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions can read an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Failing Instruction?	Always Restartable?	Error Code on Stack?
8	Double exception detected	Yes	No[2]	Yes
9	Processor extension segment overrun	No	No[2]	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or stack segment not present	Yes	Yes[1]	Yes
13	General protection	Yes	No[2]	Yes

NOTE 1: When a PUSHA or POPA instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable because stack segment wrap around is not permitted. This condition is identified by the value of the saved SP being eigher 0000(H), 0001(H), FFFE(H), or FFFF(H).

NOTE 2: These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

These exceptions indicate a violation to privilege rules or usage rules has occurred. Restart is generally not attempted under those conditions.

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception causes exception 11 or 12 and is restartable.

Special Operations

TASK SWITCH OPERATION

The 80286 provides a built-in task switch operation which saves the entire 80286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80286 execution state while a task gate descriptor contains a TSS selector. The limit field of the descriptor must be >002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag egister is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task return by popping values off the stack; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL, JMP, or INT instruction initiates a task switch, the old and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The iAPX 286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instructions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag (ZF) indicates whether use of the selector or segment will cause an exception.

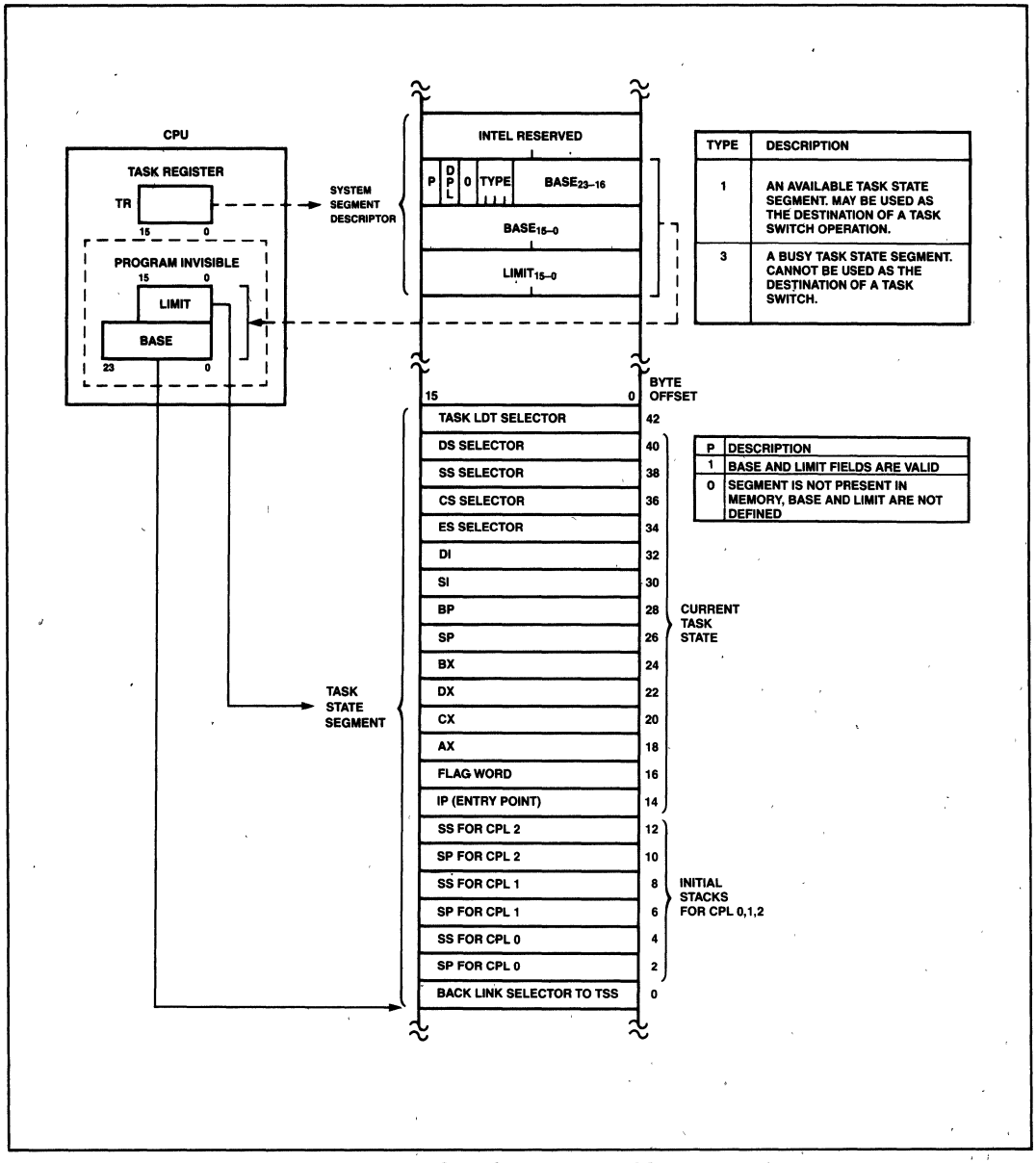


Figure 20. Task State Segment and TSS Registers

Table 15. 80286 Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed by ARPL.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80286 performs the double fault exception (8). If an exception occurs during processing of the double fault exception, the 82086 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 HIGH.

PROTECTED MODE INITIALIZATION

The 80286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, A_{23-20} will be HIGH when the 80286 performs memory references relative to the CS register until CS is changed. A_{23-20} will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A_{23-20} LOW whenever CS is used again. The initial CS:IP value of F000:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Protected mode operation requires several registers to be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80286 must immediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since any task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The iAPX 286 family includes several devices to generate standard system buses such as the IEEE 796 standard Multibus™.

Bus Interface Signals and Timing

The iAPX 286 microsystem local bus interfaces the 80286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80286 CPU, 82284 clock generator, 82288 bus controller, 82289 bus arbiter, 8286/7 transceivers, and 8282/3 latches provide a buffered and decoded system bus interface. The 82284 generates the system clock and synchronizes READY and RESET. The 82288 converts bus operation status encoded by the 80286 into command and bus control signals. The 82289 bus arbiter generates Multibus bus arbitration signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the Multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8}. Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed words require two bus operations. The first transfers data on D_{15-8}, and the second transfers data on D_{7-0}. Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and $\overline{BHE}$, control transfers over the lower and upper halves of the data bus. Even address

byte transfers are indicated by A_0 LOW and $\overline{BHE}$ HIGH. Odd address byte transfers are indicated by A_0 HIGH and $\overline{BHE}$ LOW. Both A_0 and $\overline{BHE}$ are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte (D_{15-8}) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 8259A must be connected to the lower data byte (D_{7-0}) for proper return of the interrupt vector.

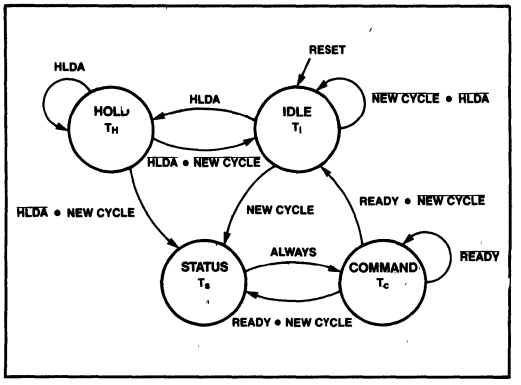


Figure 22. 80286 Bus States

Bus Operation

The 80286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

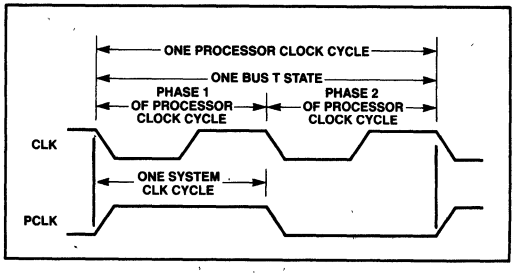


Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The iAPX 286 bus has three basic states: idle (T_i), send status (T_s), and perform command (T_c). The 80286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80286 local bus states and allowed transitions.

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state T_S is signaled by status line $\overline{S1}$ or $\overline{S0}$ going LOW and identifying phase 1 of the processor clock. During T_S, the command encoding, the address, and data (for a write operation) are available on the 80286 output pins. The 82288 bus controler decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_S, the perform command (T_C) state is entered. Memory or I/O devices respond to the bus operation during T_C, either transferring read data to the CPU or accepting write data. T_C states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The $\overline{READY}$ signal determines whether T_C is repeated. A repeated T_C state is called a wait state.

During hold (T_h), the 80286 will float all address, data, and status output pins enabling another bus master to use the local bus. The 80286 HOLD input signal is used to place the 80286 into the T_h state. The 80286 HLDA output signal indicates that the CPU has entered T_h.

Pipelined Addressing

The 80286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows a new bus operation to be initiated every two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in ad-

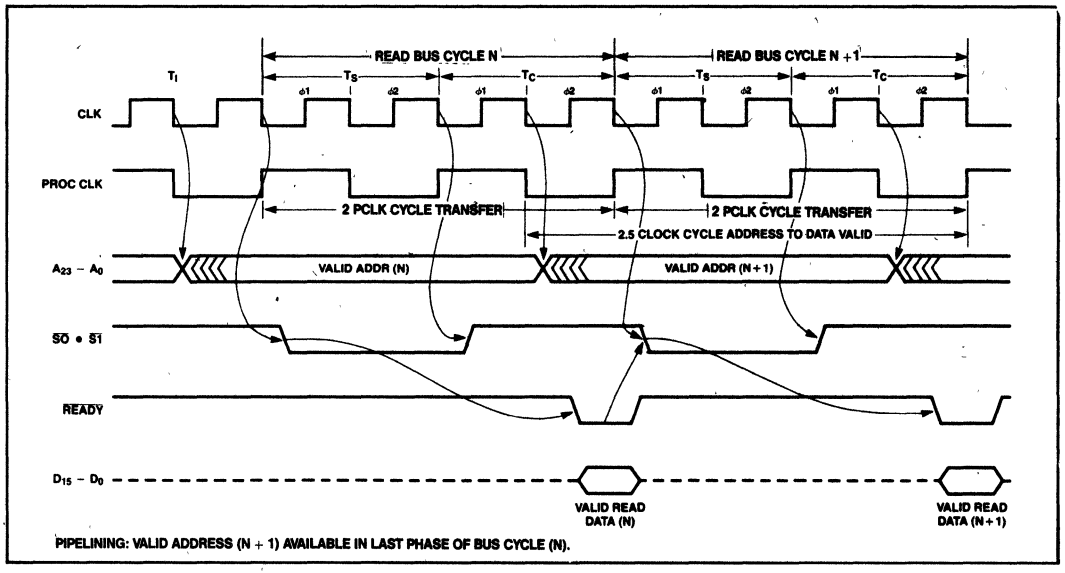


Figure 23. Basic Bus Cycle

vance of the next bus operation. External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80286 does not maintain the address of the current bus operation during all T_C states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_C. The address remains valid during phase 1 of the first T_C to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/$\overline{R}$), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support Multibus® and common memory systems.

The data bus transceivers are controlled by 82288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/$\overline{R}$). DEN enables the data transceivers; while DT/$\overline{R}$ controls transceiver direction. DEN and DT/$\overline{R}$ are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus tranceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the iAPX 286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The $\overline{READY}$ input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82288 CMDLY input. After T_S, the bus controller samples CMDLY at each failing edge of CLK. If CMDLY is HIGH, the 82288 will not activate the command signal. When CMDLY is LOW, the 82288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/$\overline{R}$.

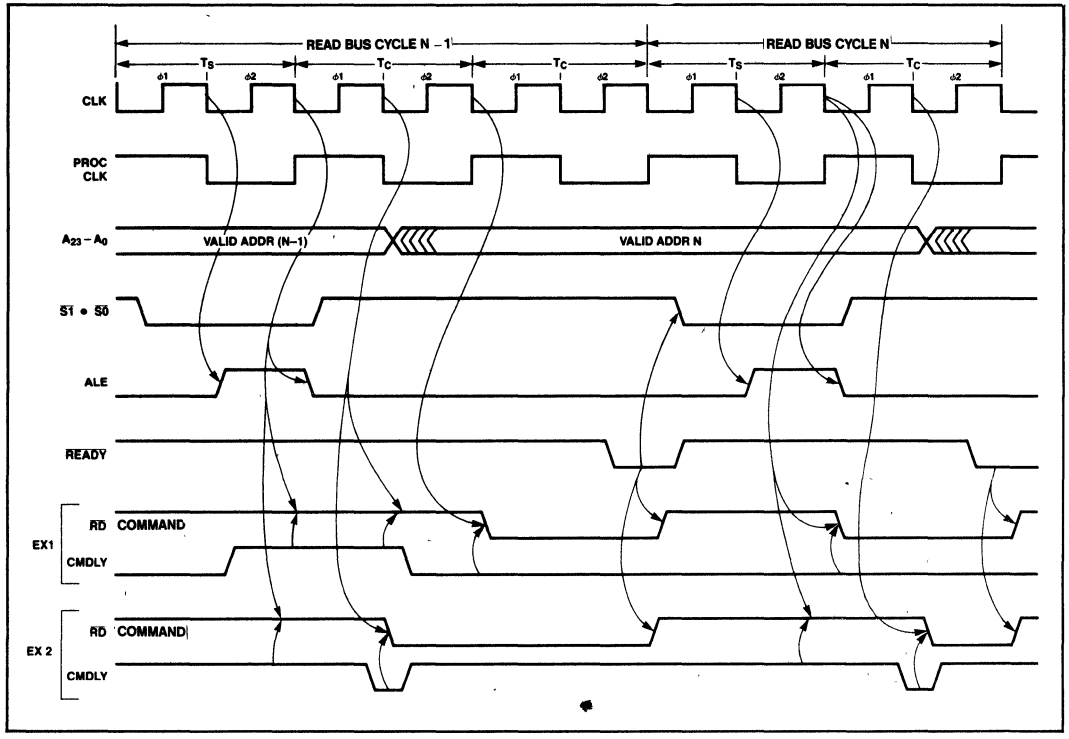


Figure 24. CMDLY Controls the Leading Edge of Command Signal.

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the iAPX 286 bus alternates between the status and command states. The bus status signals become inactive after T_s so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_c exists on the iAPX 286 local bus. The bus master and bus controller enter T_c directly after T_s and continue executing T_c cycles until terminated by $\overline{READY}$.

$\overline{READY}$ Operation

The current bus master and 82288 bus controller terminate each bus operation simultaneously to achieve maximum bus operation bandwidth. Both are informed in advance by $\overline{READY}$ active (open-collector output from 82284) which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{READY}$ signal, thereby requiring $\overline{READY}$ be synchronous to the system clock.

Synchronous Ready

The 82284 clock generator provides $\overline{READY}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{SRDY}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_c. The state of $\overline{SRDY}$ is then broadcast to the bus master and bus controller via the $\overline{READY}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82284 $\overline{SRDY}$ setup and hold time requirements. But the 82284 asynchronous ready input ($\overline{ARDY}$) is designed to accept such signals. The $\overline{ARDY}$ input is sampled at the beginning of each T_C cycle by 82284 synchronization logic. This provides one system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

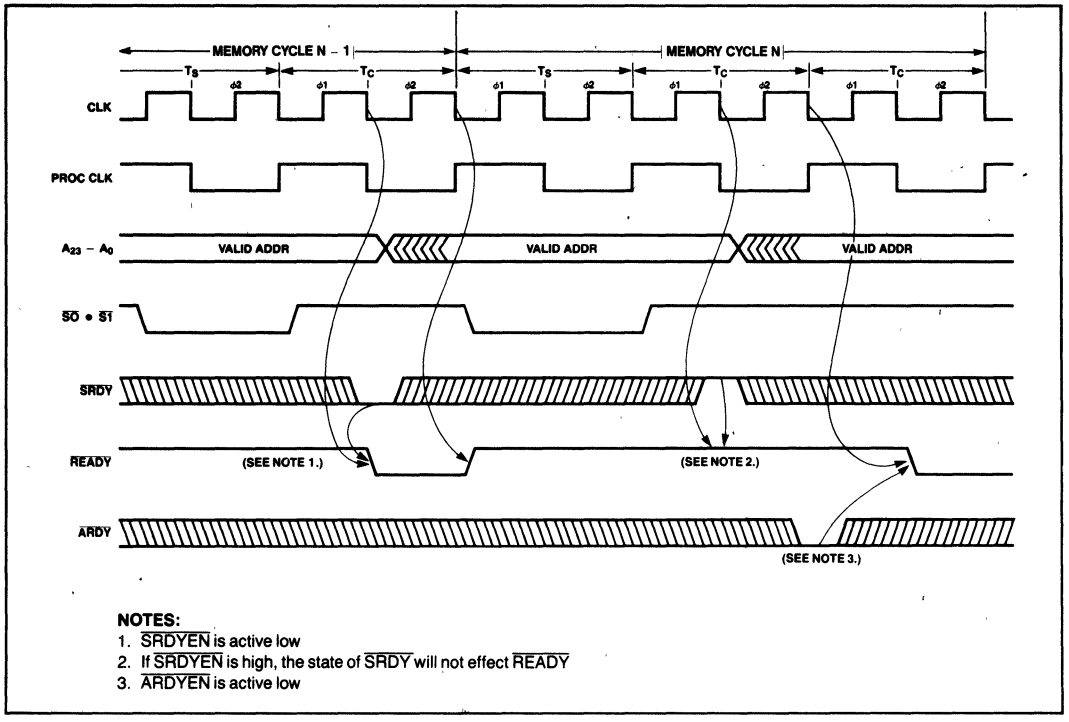


Figure 25. Synchronous and Asynchronous Ready

ARDY or ARDYEN must be HIGH at the end of T_S. ARDY cannot be used to terminate bus cycle with no wait states.

Each ready input of the 82284 has an enable pin (SRDYEN and ARDYEN) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by ARDY or SRDY.

Data Bus Control

Figures 26, 27, and 28 show how the DT/R, DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/R goes active (LOW) for a read operaton. DT/R remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_s. The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF before the 80286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_c to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF during the second phase of the processor cycle after the last T_c. In a write-write sequence the data bus does not enter 3-state OFF between T_c and T_s.

Bus Usage

The 80286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

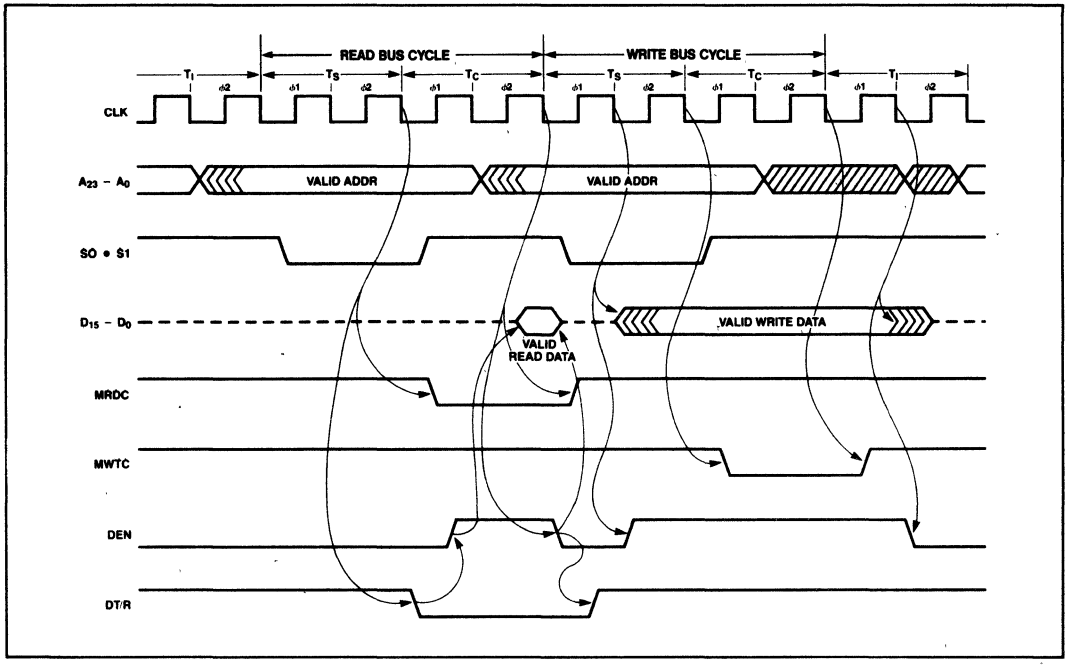


Figure 26. Back to Back Read-Write Cycles

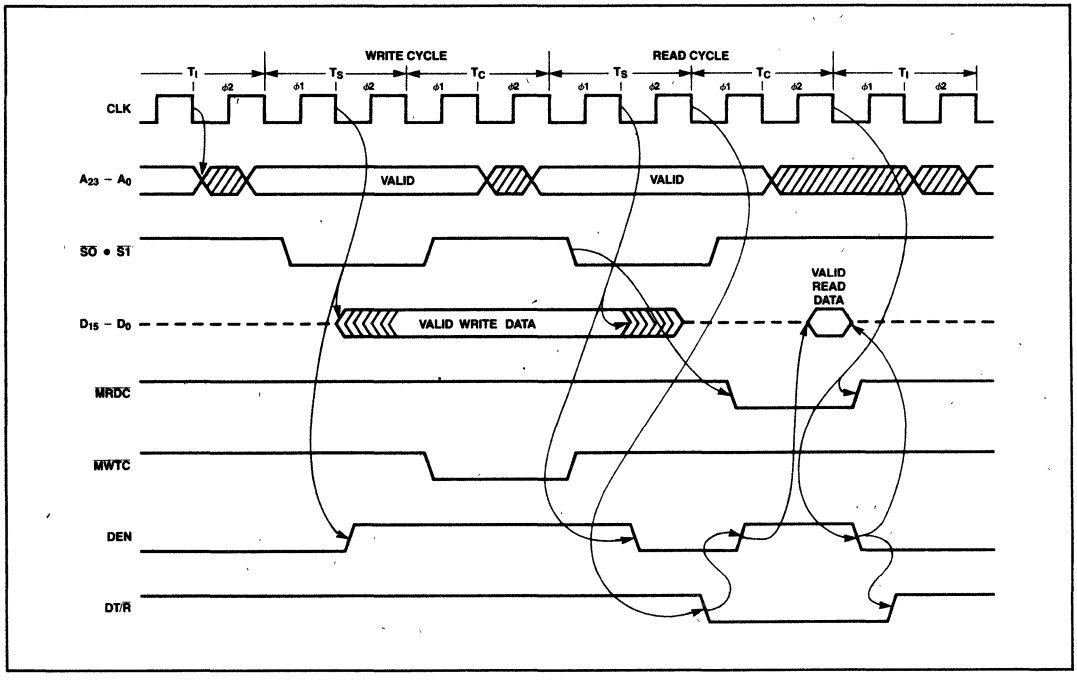


Figure 27. Back to Back Write-Read Cycles

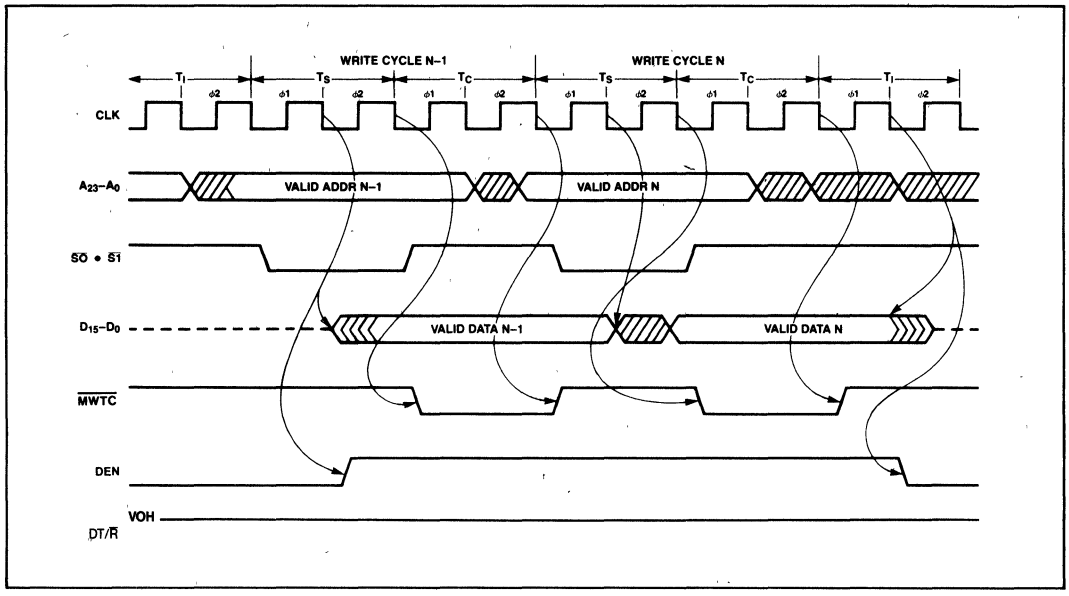


Figure 28. Back to Back Write-Write Cycles

HOLD and HLDA

HOLD and HLDA allow another bus master to gain control of the local bus by placing the 80286 bus into the T_h state. The sequence of events required to pass control between the 80286 and another local bus master are shown in Figure 29.

In this example, the 80286 is initially in the T_h state as signaled by HLDA being active. Upon leaving T_h, as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80286 as shown by the HOLD signal. After completing the write operation, the 80286 performs one T_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The CMDLY signal and $\overline{ARDY}$ ready are used to start and stop the write bus command, respectively. Note that $\overline{SRDY}$ must be inactive or disabled by $\overline{SRDYEN}$ to guarantee $\overline{ARDY}$ will terminate the cycle.

Instruction Fetching

The 80286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

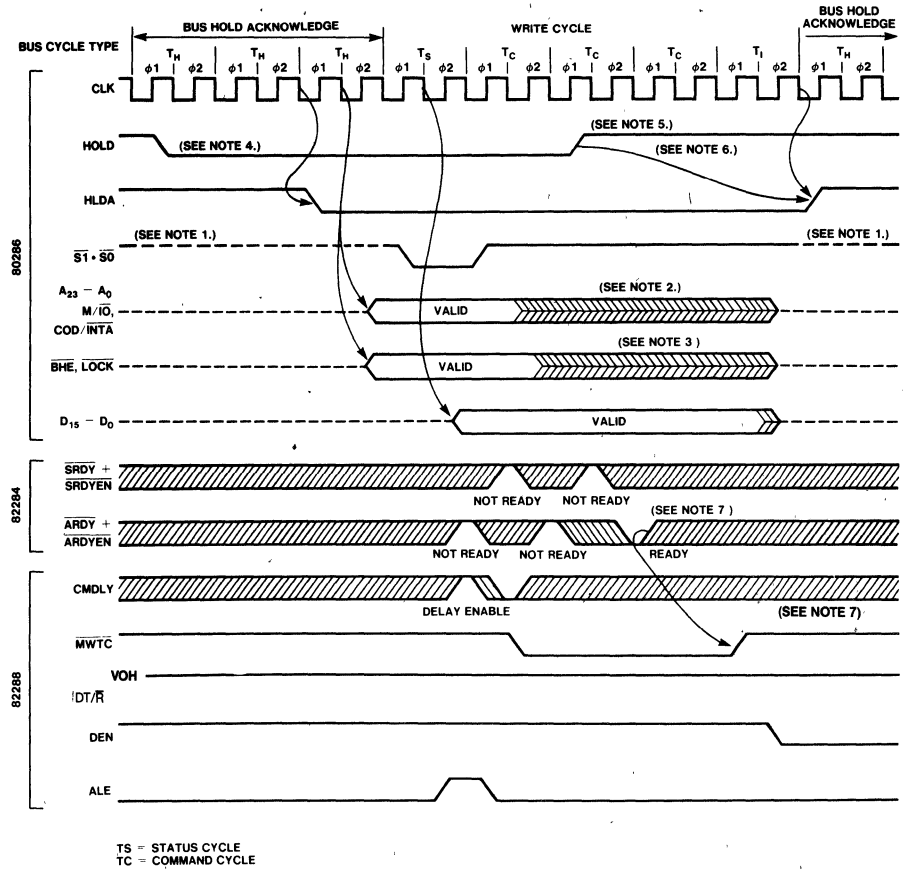
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 6 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



NOTES:

1. Status lines are not driven by 80286, yet remain high due to pullup resistors in 82288 and 82289 during HOLD state
2. Address, M/$\overline{IO}$ and COD/$\overline{INTA}$ may start floating during any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 2$ of TC
3. $\overline{BHE}$ and $\overline{LOCK}$ may start floating after the end of any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 1$ of TC
4. The minimum HOLD to HLDA time is shown. Maximum is one T_H longer
5. The earliest HOLD time is shown It will always allow a subsequent memory cycle if pending is shown
6. The minimum HOLD to HLDA time is shown. Maximum is a function of the instruction, type of bus cycle and other machine status (i e , Interrupts, Waits, Lock, etc.)
7. Asynchronous ready allows termination of the cycle Synchronous ready does not signal ready in this example Synchronous ready state is ignored after ready is signaled via the asynchronous input

Figure 29. Multibus Write Terminated by Asynchronous Ready with Bus Hold

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with Machine Status Word bits EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80286 in response to an INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 8259A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read on D0-D7 of the 80286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80286 emits the $\overline{LOCK}$ signal (active LOW) during T_S of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 8259A. The second INTA bus operation must always have at least one extra T_C state added via logic controlling $\overline{READY}$. A_{23}–A_0 are in 3-state OFF until after the first T_C state of the second INTA bus operation. This prevents bus contention between the cascade address drivers and CPU address drivers. The extra T_C state allows time for the 80286 to resume driving the address lines for subsequent bus operations.

Local Bus Usage Priorities

The 80286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest) Any transfers which assert $\overline{LOCK}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. segment descriptor access, interrupt acknowledge sequence, or an XCHG with memory).

The second of the two byte bus operations required for an odd aligned word operand.

The second or third cycle of a processor extension data transfer.

Local bus request via HOLD input.

Processor extension data operand transfer via PEREQ input.

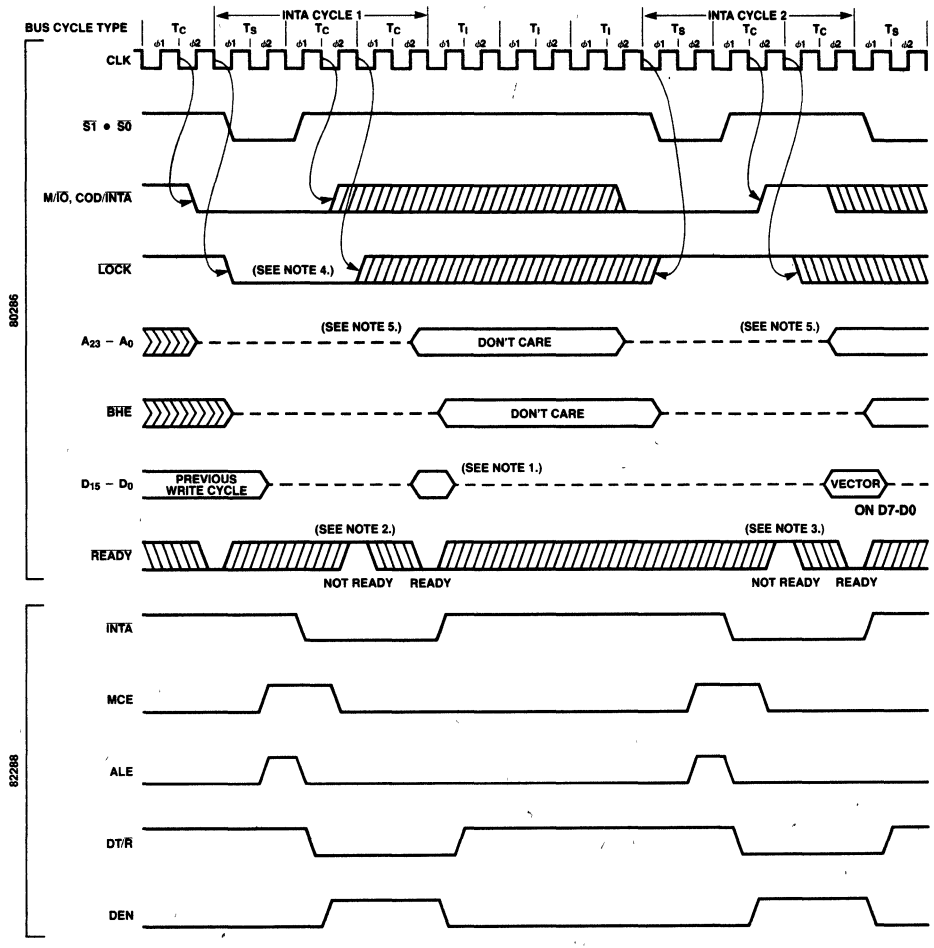
Data transfer performed by EU as part of an instruction.

(Lowest) An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.

Halt or Shutdown Cycles

The 80286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{S1}$, $\overline{S0}$ and COD/$\overline{INTA}$ are LOW and M/$\overline{IO}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The 82288 bus controller does not issue ALE, nor is $\overline{READY}$ required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80286 out of halt.



NOTES:

1. Data is ignored.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle must have at least one wait state inserted since the CPU will not drive $A_{23} - A_0$, $\overline{BHE}$, and $\overline{LOCK}$ until after the first TC state.

   The CPU imposed one/clock delay prevents bus contention between cascade address buffer being disabled by MCE and address outputs.

   Without the wait state, the 80286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The 8259A also requires one wait state for minimum INTA pulse width.
4. $\overline{LOCK}$ is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system.
5. $A_{23} - A_0$ exits 3-state OFF during $\phi 2$ of the second T_C in the INTA cycle.

Figure 30. Interrupt Acknowledge Sequence

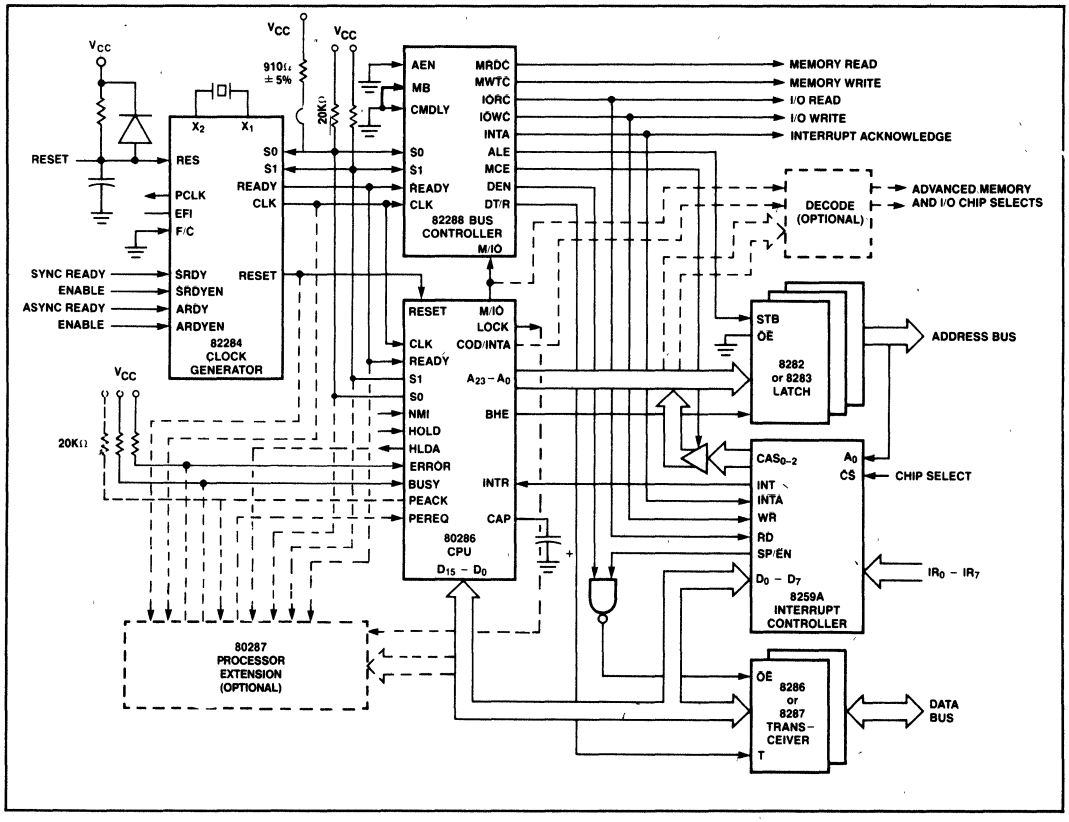


Figure 31. Basic iAPX 286 System Configuration

SYSTEM CONFIGURATIONS

The versatile bus structure of the iAPX 286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 31, is similar to an iAPX 86 maximum mode system. It includes the CPU plus an 8259A interrupt controller, 82284 clock generator, and the 82288 Bus Controller. The iAPX 86 latches (8282 and 8283) and transceivers (8286 and 8287) may be used in an iAPX 286 microsystem.

As indicated by the dashed lines in Figure 31, the ability to add processor extensions is an integral feature of iAPX 286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80286 supervises all data transfers and instruction execution for the processor extension.

The iAPX 286/20 numeric data processor which includes the 80287 numeric processor extension (NPX) uses this interface. The iAPX 286/20 has all the instructions and data types of an iAPX 86/20 or iAPX 88/20. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the iAPX 286 protection mechanism.

The 80286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 8282/3's by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycles address is being decoded and propagated into the system. Decode logic can be implemented with a high speed bipolar PROM.

The optional decode logic shown in Figure 31 takes advantage of the overlap between address and data of the 80286 bus cycle to generate advanced memory and IO-select signals. This minimizes system performance

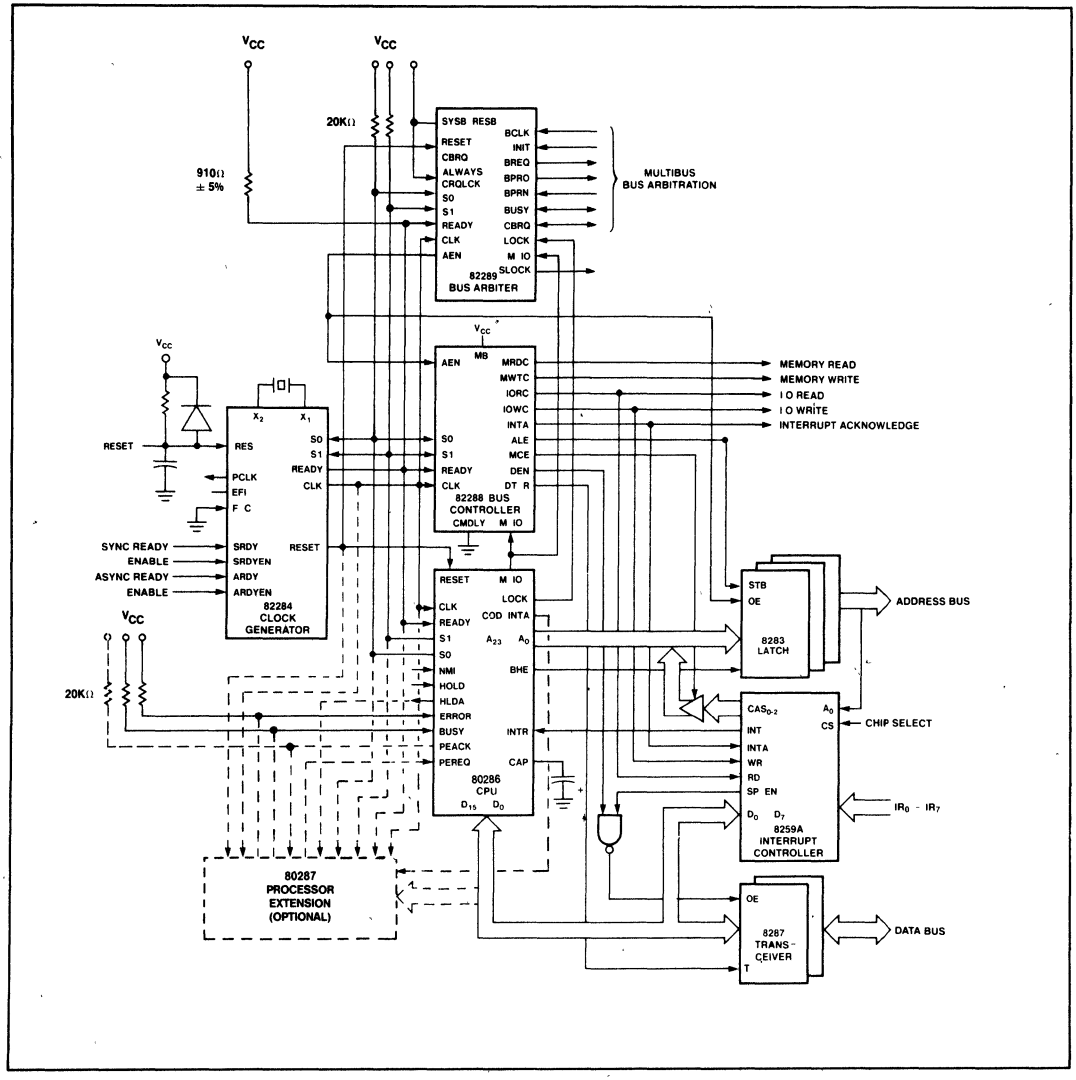


Figure 32. Multibus System Bus Interface

degradation caused by address propogation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/$\overline{INTA}$ and M/$\overline{IO}$ signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding the 82289 bus arbiter chip the 80286 provides a Multibus system bus interface as shown in Figure 32. The ALE output of the 82288 for the Multibus bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet Multibus address and write data setup times. This arrangement will add at least one extra T_C state to each bus operation which uses the Multibus.

A second 82288 bus controller and additional latches and transceivers could be added to the local bus of Figure 32. This configuration allows the 80286 to support an on-board bus for local memory and peripherals, and the Multibus for system bus interfacing.

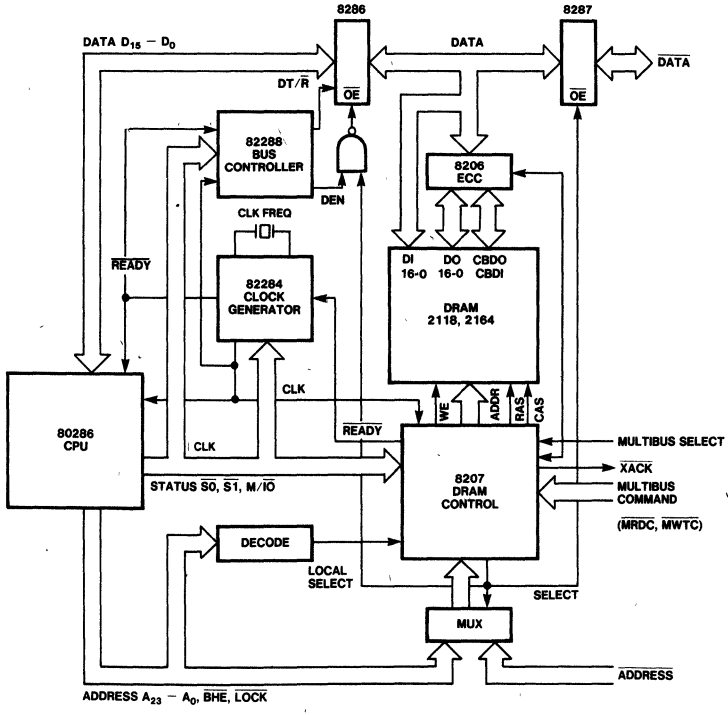


Figure 33. iAPX 286 System Configuration with Dual-Ported Memory

Figure 33 shows the addition of dual ported dynamic memory between the Multibus system bus and the iAPX 286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh, initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard Multibus system bus interface to maximize performance and protection in multiprocessor system configurations.

Table 16. 80286 Systems Recommended Pull up Resistor Values

80286 Pin and Name	Pullup Value	Purpose
4 – $\overline{S1}$	20KΩ ± 10%	Pull $\overline{S0}$, $\overline{S1}$, and $\overline{PEACK}$ inactive during 80286 hold periods
5 – $\overline{S0}$		
6 – $\overline{PEACK}$		
53 – $\overline{ERROR}$	20KΩ ± 10%	Pull $\overline{ERROR}$ and $\overline{BUSY}$ inactive when 80287 not present (or temporarily removed from socket)
54 – $\overline{BUSY}$		
63 – $\overline{READY}$	910Ω ± 5%	Pull $\overline{READY}$ inactive within required minimum time (C_L = 150pF, $I_R \leq 7mA$)

PACKAGE

The 80286 is packaged in a 68-pin, leadless JEDEC type A hermetic leadless chip carrier. Figure 34 illustrates the package, and Figure 2 shows the pinout.

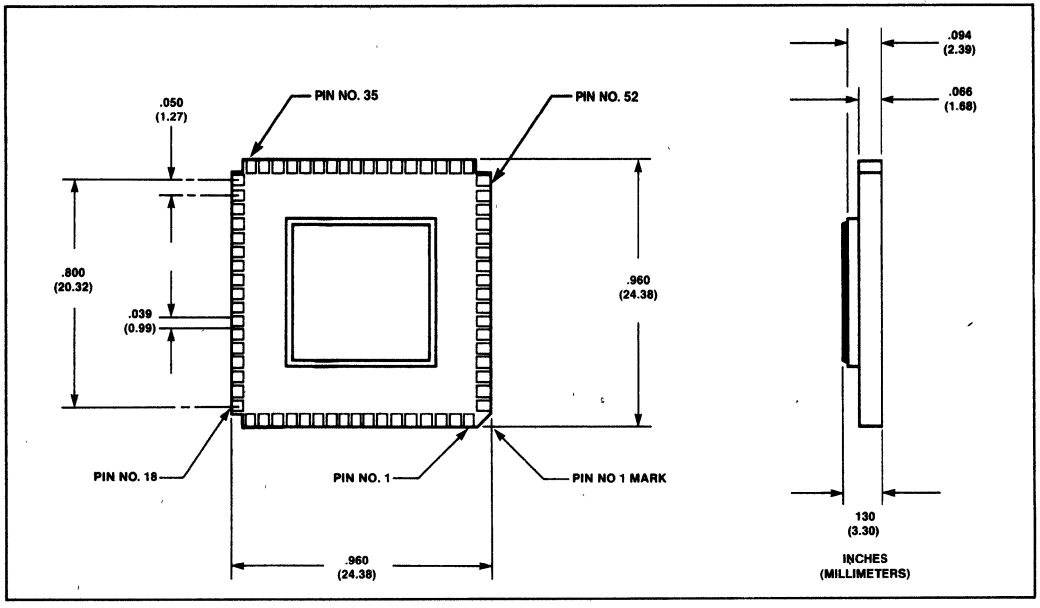


Figure 34. JEDEC Type A Package

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with Respect to Ground . . . . . −1.0 to +7V
Power Dissipation . . . . . 3.6 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 55°C, V_{CC} = 5V, ± 5%)

		4 MHz		6 MHz		8 MHz			
Sym	Parameter	-4 Min	-4 Max	-6 Min	-6 Max	Min	Max	Unit	Test Condition
V_{IL}	Input LOW Voltage	-.5	.8	-.5	.8	-.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	2.0	V_{CC} + .5	2.0	V_{CC} + .5	V	
V_{ILC}	CLK Input LOW Voltage	-.5	.6	-.5	.6	-.5	.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	V_{CC} + .5	3.8	V_{CC} + .5	3.8	V_{CC} + .5	V	
V_{OL}	Output LOW Voltage	.	.45	.	.45	.	.45	V	I_{OL} = 2.0mA
V_{OH}	Output HIGH Voltage	2.4	.	2.4	.	2.4	.	V	I_{OH} = -400μA
I_{LI}	Input Leakage Current	.	+-10	.	+-10	.	+-10	μA	$0V \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current	.	+-10	.	+-10	.	+-10	μA	$.45V \le V_{OUT} \le V_{CC}$
I_{CC}	Supply Current (turn on, 0°C)	.	600	.	600	.	600	μA	Note 1
C_{CLK}	CLK Input Capacitance	.	12	.	12	.	12	pF	F_C = 1MHz
C_{IN}	Other Input Capacitance	.	10	.	10	.	10	pF	F_C = 1MHz
C_O	Input/Output Capacitance	.	20	.	20	.	20	pF	F_C = 1MHz

NOTE 1: Low temperature is worst case.

A.C. CHARACTERISTICS (T_A = 0°C to 55°C, V_{CC} = 5V, ± 5%)

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

		4 MHz		6 MHz		8 MHz			
Sym	Parameter	-4 Min	-4 Max	-6 Min	-6 Max	Min	Max	Unit	Test Condition
1	System Clock (CLK) Period	124	250	83	250	62	250	ns	
2	System Clock (CLK) LOW Time	30	210	20	250	15	225	ns	at 0.6V
3	System Clock (CLK) HIGH Time	40	220	25	230	25	235	ns	at 3.2V
17	System Clock (CLK) Rise Time		10		10		10	ns	1.0V to 3.5V
18	System Clock (CLK) Fall Time		10		10		10	ns	3.5V to 1.0V
4	Asynch. Inputs Setup Time	40		30		20		ns	Note 1
5	Asynch. Inputs Hold Time	40		30		20		ns	Note 1
6	RESET Setup Time	40		25		20		ns	
7	RESET Hold Time	0		0		0		ns	
8	Read Data Setup Time	30		20		10		ns	
9	Read Data Hold Time	10		8		5		ns	
10	$\overline{\text{READY}}$ Setup Time	75		50		38.5		ns	
11	$\overline{\text{READY}}$ Hold Time	50		35		25		ns	
12	Status/$\overline{\text{PEACK}}$ Valid Delay	0	80	0	55	0	37.5	ns	Note 2 Note 3
13	Address Valid Delay	0	120	0	80	0	60	ns	Note 2 Note 3
14	Write Data Valid Delay	0	100	0	65	0	50	ns	Note 2 Note 3
15	Address/Status/Data Float Delay	0	120	0	80	0	60	ns	Note 2 Note 4
16	HLDA Valid Delay	0	120	0	80	0	60	ns	Note 2 Note 3

NOTE: 1: Asychronous inputs are INTR, NMI, HOLD, PEREQ, $\overline{\text{ERROR}}$, and $\overline{\text{BUSY}}$. This specification is given only for testing purposes, to assure recognition at a specific CLK edge

NOTE: 2: Delay from 0 8V on the CLK, to 0 8V or 2 0V or float on the output as appropriate for valid or floating condition

NOTE: 3: Output load C1 = 100pF

NOTE: 4: Float condition occurs when output current is less than I_{LO} in magnitude

82284 Timing Requirements

		82286-6		822284			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
11	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ setup time	25		15		ns	
12	$\overline{\text{SRDY}}$/$\overline{\text{SRDYEN}}$ hold time	0		0		ns	
13	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ setup time	5		0		ns	See note 1
14	$\overline{\text{ARDY}}$/$\overline{\text{ARDYEN}}$ hold time	30		16		ns	See note 1
19	PCLK delay	0	45	0	45	ns	C_L = 75pF I_{OL} = 5 ma I_{OH} = −1 ma

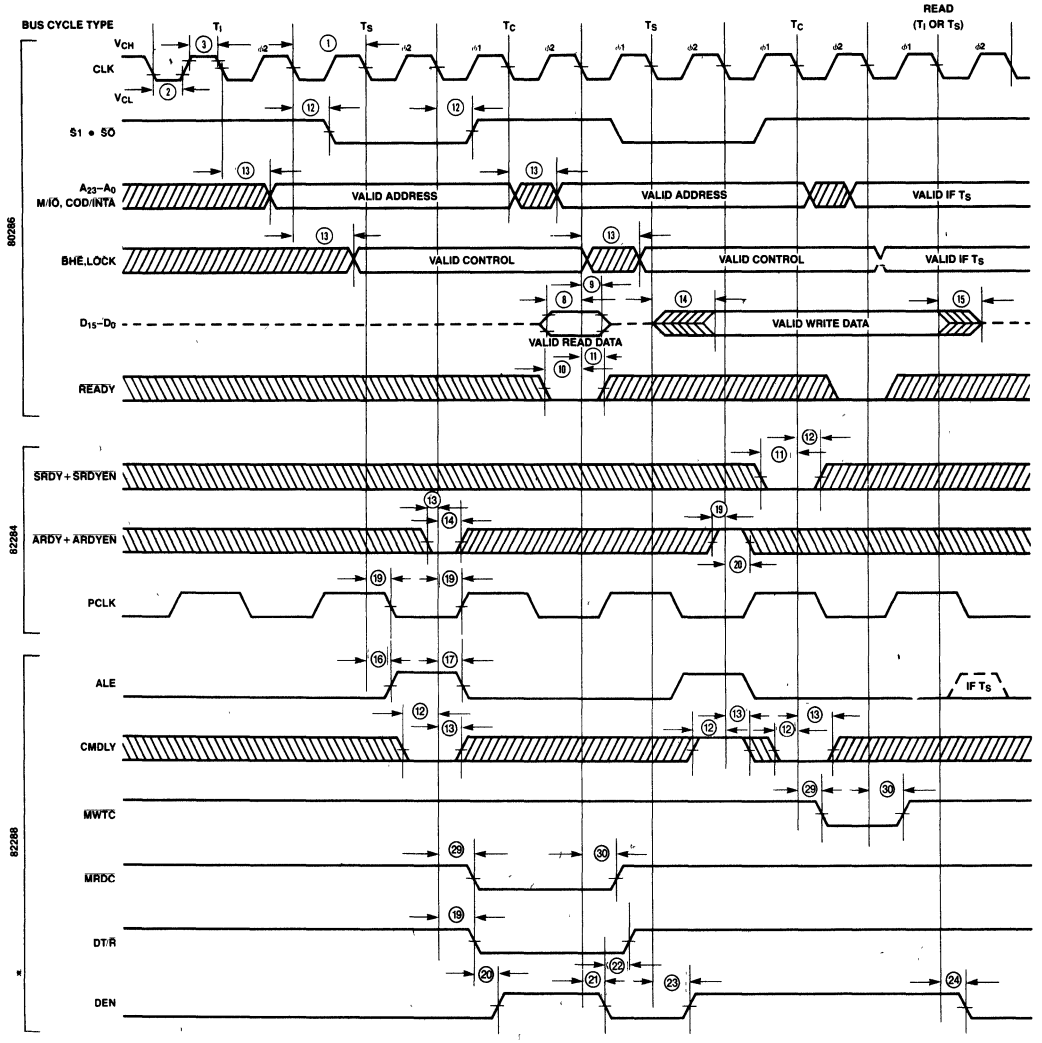
NOTE : These times are given for testing purposes to assure a predetermined action

82288 Timing Requirements

			82288-6		82288			
Symbol	Parameter		Min.	Max.	Min.	Max.	Units	Test Conditions
12	CMDLY setup time		25		20		ns	
13	CMDLT hold time		0		0		ns	
30	Command delay From CLK	Command Inactive	3	30	3	20	ns	C_L = 300 pF max I_{OL} = 32 ma max I_{OH} = 5 ma max
29		Command Active	3	40	3	20		
16	ALE active delay		3	25	3	15	ns	
17	ALE inactive delay			35		20	ns	
19	DT/$\overline{\text{R}}$ read active delay			40	0	20	ns	
22	DT/$\overline{\text{R}}$ read inactive delay		5	45	10	40	ns	C_L = 150 pF
20	DEN read active delay		10	50	10	40	ns	I_{OL} = 16 ma max
21	DEN read inactive delay		3	40	3	15	ns	I_{OH} = -1 ma max
23	DEN write active delay			35		30	ns	
24	DEN write inactive delay		3	35	3	30	ns	

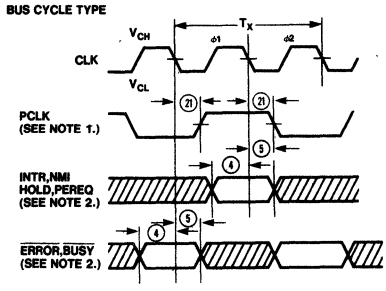
WAVEFORMS

MAJOR CYCLE TIMING



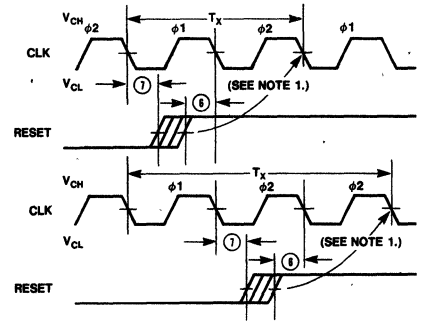
WAVEFORMS (Continued)

80286 ASYNCHRONOUS INPUT SIGNAL TIMING



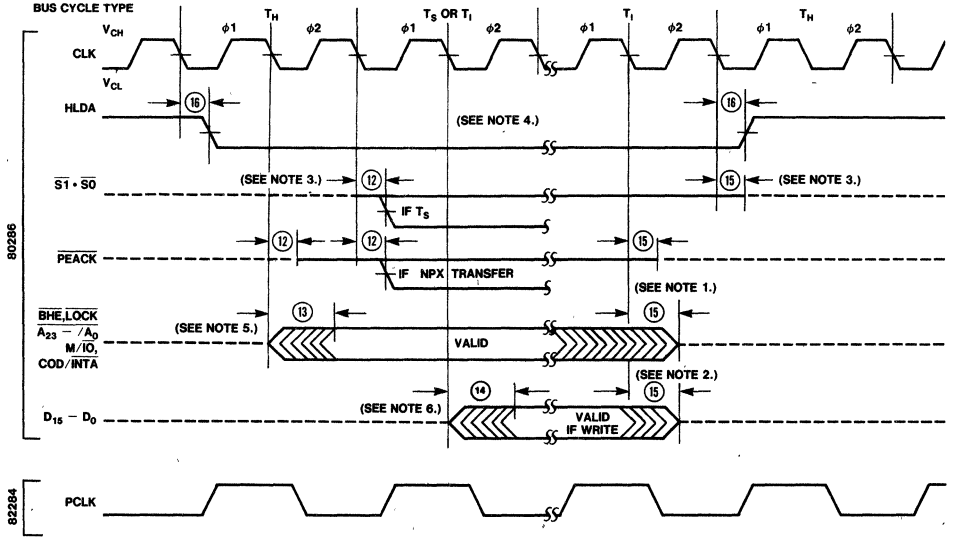
NOTES:
1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

80286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE



NOTE 1: When RESET meets the setup time shown, the next CLK will start or repeat $\phi 2$ of a processor cycle.

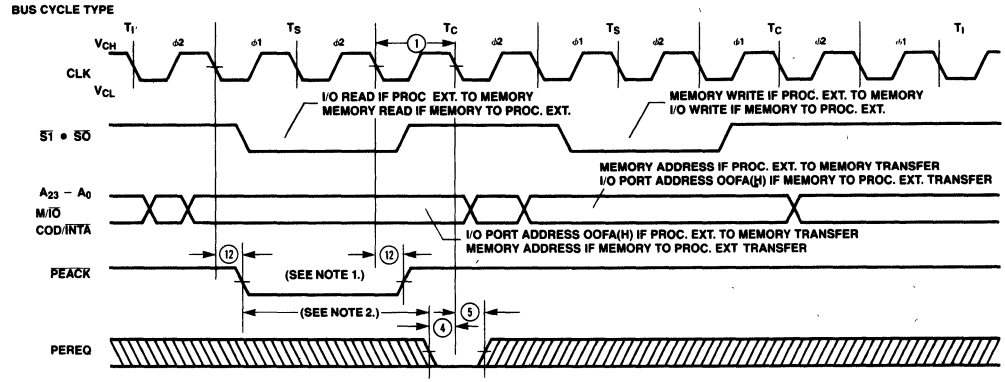
EXITING AND ENTERING HOLD



NOTES:
1. These signals may not be driven by the 80286 during the time shown. The worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_I in the diagram was a write T_C.
3. The 80286 floats its status pins during T_H. External 20KΩ resistors keep these signals high (see Table 16).
4. For HOLD request set up to HLDA, refer to Figure 29.
5. $\overline{BHE}$ and $\overline{LOCK}$ are driven at this time but will not become valid until T_S.
6. The data bus will remain in 3-state OFF if a read cycle is performed

WAVEFORMS (Continued)

80286 PEREQ/PEACK TIMING FOR ONE TRANSFER ONLY

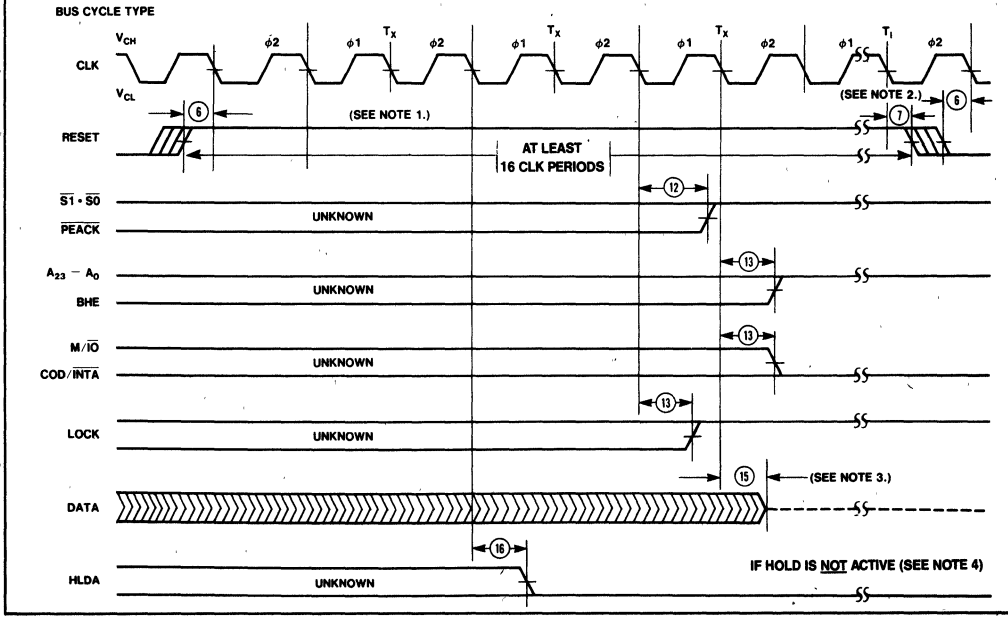


ASSUMING WORD-ALIGNED MEMORY OPERAND. IF ODD ALIGNED, 80286 TRANSFERS TO/FROM MEMORY BYTE-AT-A-TIME WITH TWO MEMORY CYCLES.

NOTES:

1. $\overline{\text{PEACK}}$ always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address OOFA(H).
2. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is: $3X(1)-(11)_{max}-(4)_{min}$. The actual, configuration dependent, maximum time is: $3X(1)-(11)_{max}-(4)_{min.}+AX2X(1)$.
   A is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.

INITIAL 80286 PIN STATE DURING RESET



NOTES:

1. Setup time for RESET ↑ may be violated with the consideration that ø1 of the processor clock may begin one system CLK period later
2. Setup and hold times for RESET ↓ must be met for proper operation, but RESET ↓ may occur during ø1 or ø2.
3. The data bus is only guaranteed to be in 3-state OFF at the time shown.
4. HOLD is acknowledged during RESET, causing HLDA to go active and the appropriate pins to float If HOLD remains active while RESET goes inactive, the 80286 remains in HOLD state and will not perform any bus accesses until HOLD is de-activiated.

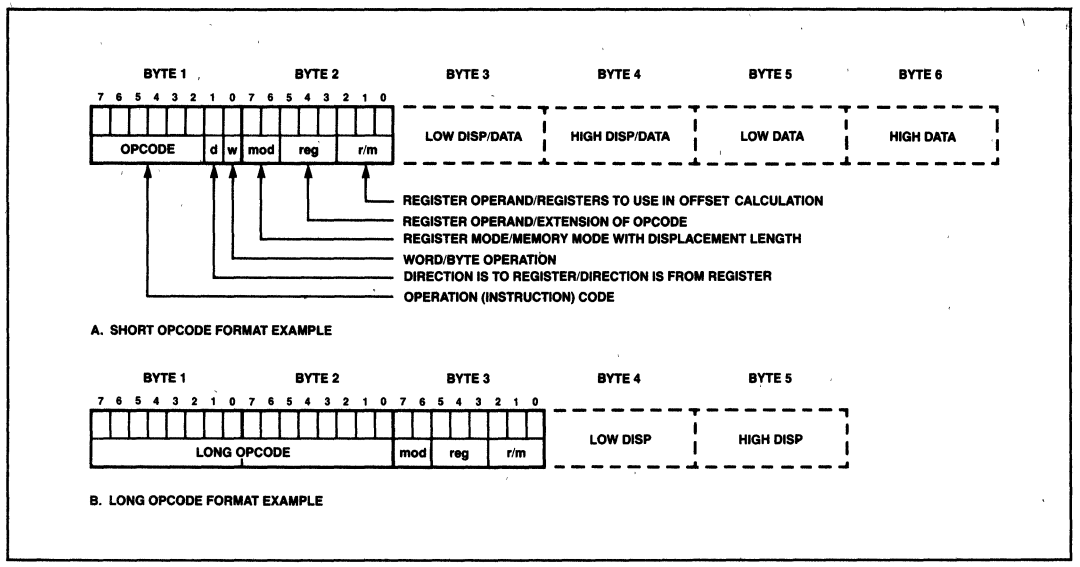


Figure 35. 80286 Instruction Format Examples

80286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80286. With no delays in bus cycles, the actual clock count of an 80286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 8 MHz processor clock has a clock period of 125 nanoseconds and requires an 80286 system clock (CLK input) of 16 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value

Greater refers to positive signed value

Less refers to less positive (more negative) signed values

if d = 1 then to register; if d = 0 then from register

if w = 1 then word instruction; if w = 0 then byte instruction

if s = 0 then 16-bit immediate data form the operand

if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.
8. $\overline{\text{LOCK}}$ does not remain active between all operand transfers.

PROTECTED VIRTUAL ADDRESS MODE ONLY

9. A general protection exception (13) will occur if the memory operand can not be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.
11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if CPL $\neq$ 0.
14. A general protection exception (13) occurs if CPL > IOPL.
15. The IF field of the flag word is not updated if CPL > IOPL. The IOPL field is updated only if CPL = 0.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.
18. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) will occur.

80286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1 0 0 0 1 0 0 w | mod reg r/m	2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w | mod reg r/m	2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg | data | data if w = 1	2	2		
Memory to accumulator	1 0 1 0 0 0 0 w | addr-low | addr-high	5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w | addr-low | addr-high	3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0 | mod 0 reg r/m	2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0 | mod 0 reg r/m	2,3*	2,3*	2	9
PUSH = Push:					
Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	5*	5*	2	9
Register	0 1 0 1 0 reg	3	3	2	9
Segment register	0 0 0 reg 1 1 0	3	3	2	9
Immediate	0 1 1 0 1 0 s 0 | data | data if s = 0	3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0	17	17	2	9
POP = Pop:					
Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	5*	5*	2	9
Register	0 1 0 1 1 reg	5	5	2	9
Segment register	0 0 0 reg 1 1 1 | (reg ≠ 01)	5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1	19	19	2	9
XCHG = Exchange:					
Register/memory with register	1 0 0 0 0 1 1 w | mod reg r/m	3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg	3	3		
IN = Input from:					
Fixed port	1 1 1 0 0 1 0 w | port	5	5		14
Variable port	1 1 1 0 1 1 0 w	5	5		14
OUT = Output to:					
Fixed port	1 1 1 0 0 1 1 w | port	3	3		14
Variable port	1 1 1 0 1 1 1 w	3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1	5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1 | mod reg r/m	3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1 | mod reg r/m | (mod ≠ 11)	7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0 | mod reg r/m | (mod ≠ 11)	7*	21*	2	9,10,11
LAHF = Load AH with flags	1 0 0 1 1 1 1 1	2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0	2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0	3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1	5	5	2,4	9,15

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
ARITHMETIC					
ADD = Add:					
Reg/memory with register to either	0 0 0 0 0 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 0 0 r/m | data | data if s w = 0 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w | data | data if w = 1	3	3		
ADC = Add with carry:					
Reg/memory with register to either	0 0 0 1 0 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w | mod 0 1 0 r/m | data | data if s w = 0 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w | data | data if w = 1	3	3		
INC = Increment:					
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 0 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 0 reg	2	2		
SUB = Subtract:					
Reg/memory and register to either	0 0 1 0 1 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w | mod 1 0 1 r/m | data | data if s w = 0 1	3,7*	3,7*	2	9
Immediate from accumulator	0 0 1 0 1 1 0 w | data | data if w = 1	3	3		
SBB = Subtract with borrow:					
Reg/memory and register to either	0 0 0 1 1 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | data | data if s w = 0 1	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w | data | data if w = 1	3	3		
DEC = Decrement:					
Register/memory	1 1 1 1 1 1 1 w | mod 0 0 1 r/m	2,7*	2,7*	2	9
Register	0 1 0 0 1 reg	2	2		
CMP = Compare:					
Register/memory with register	0 0 1 1 1 0 1 w | mod reg r/m	2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w | mod reg r/m	2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | data | data if s w = 0 1	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w | data | data if w = 1	3	3		
NEG = Change sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	2	7*	2	7
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1	3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1	3	3		
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1	3	3		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1	3	3		
MUL = Multiply (unsigned)	1 1 1 1 0 1 1 w | mod 1 0 0 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w | mod 1 0 1 r/m				
Register-Byte		13	13		
Register-Word		21	21		
Memory-Byte		16*	16*	2	9
Memory-Word		24*	24*	2	9
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1 | mod reg r/m | data | data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w | mod 1 1 0 r/m				
Register-Byte		14	14	6	6
Register-Word		22	22	6	6
Memory-Byte		17*	17*	2,6	6,9
Memory-Word		25*	25*	2,6	6,9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
ARITHMETIC (Continued):					
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w | mod 1 1 1 r/m				
Register-Byte		17	17	6	6
Register-Word		25	25	6	6
Memory-Byte		20*	20*	2,6	6,9
Memory-Word		28*	28*	2,6	6,9
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	16	16		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	14	14		
CBW = Convert byte to word	1 0 0 1 1 0 0 0	2	2		
CWD = Convert word to double word	1 0 0 1 1 0 0 1	2	2		
LOGIC					
Shift/Rotate Instructions:					
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	2,7*	2,7*	2	9
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	5+n,8+n*	5+n,8+n*	2	9
Register/Memory by Count	1 1 0 0 0 0 w | mod TTT r/m | count	5+n,8+n*	5+n,8+n*	2	9
	TTT Instruction: 0 0 0 ROL; 0 0 1 ROR; 0 1 0 RCL; 0 1 1 RCR; 1 0 0 SHL/SAL; 1 0 1 SHR; 1 1 1 SAR				
AND = **And:**					
Reg/memory and register to either	0 0 1 0 0 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | data | data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 0 0 1 0 w | data | data if w = 1	3	3		
TEST = **And function to flags, no result:**					
Register/memory and register	1 0 0 0 0 1 0 w | mod reg r/m	2,6*	2,6*	2	9
Immediate data and register/memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | data | data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1 0 1 0 1 0 0 w | data | data if w = 1	3	3		
OR = **Or:**					
Reg/memory and register to either	0 0 0 0 1 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | data | data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 1 1 0 w | data | data if w = 1	3	3		
XOR = **Exclusive or:**					
Reg/memory and register to either	0 0 1 1 0 0 d w | mod reg r/m	2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | data | data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 1 0 1 0 w | data | data if w = 1	3	3		
NOT = Invert register/memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	2,7*	2,7*	2	9
STRING MANIPULATION:					
MOVS = Move byte/word	1 0 1 0 0 1 0 w	5	5	2	9
CMPS = Compare byte/word	1 0 1 0 0 1 1 w	8	8	2	9
SCAS = Scan byte/word	1 0 1 0 1 1 1 w	7	7	2	9
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w	5	5	2	9
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w	3	3	2	9
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w	5	5	2	9,14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w	5	5	2	9,14

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
STRING MANIPULATION (Continued):					
Repeated by count in CX					
MOVS = Move string	1 1 1 1 0 0 1 0 | 1 0 1 0 0 1 0 w	5+4n	5+4n	2	9
CMPS = Compare string	1 1 1 1 0 0 1 z | 1 0 1 0 0 1 1 w	5+9n	5+9n	2,8	8,9
SCAS = Scan string	1 1 1 1 0 0 1 z | 1 0 1 0 1 1 1 w	5+8n	5+8n	2,8	8,9
LODS = Load string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 1 0 w	5+4n	5+4n	2,8	8,9
STOS = Store string	1 1 1 1 0 0 1 0 | 1 0 1 0 1 0 1 w	4+3n	4+3n	2,8	8,9
INS = Input string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	5+4n	5+4n	2	9,14
OUTS = Output string	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	5+4n	5+4n	2	9,14
CONTROL TRANSFER					
CALL = Call:					
Direct within segment	1 1 1 0 1 0 0 0 | disp-low | disp-high	7+m	7+m	2	18
Register/memory indirect within segment	1 1 1 1 1 1 1 1 | mod 0 1 0 r/m	7+m,11+m*	7+m,11+m*	2,8	8,9,18
Direct intersegment	1 0 0 1 1 0 1 0 | segment offset | segment selector	13+m	26+m	2	11,12,18
Protected Mode Only (Direct intersegment):					
Via call gate to same privilege level			41+m		8,11,12,18
Via call gate to different privilege level, no parameters			82+m		8,11,12,18
Via call gate to different privilege level, x parameters			86+4x+m		8,11,12,18
Via TSS			177+m		8,11,12,18
Via task gate			182+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 0 1 1 r/m (mod ≠ 11)	16+m	29+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):					
Via call gate to same privilege level			44+m*		8,9,11,12,18
Via call gate to different privilege level, no parameters			83+m*		8,9,11,12,18
Via call gate to different privilege level, x parameters			90+4x+m*		8,9,11,12,18
Via TSS			180+m*		8,9,11,12,18
Via task gate			185+m*		8,9,11,12,18
JMP = Unconditional jump:					
Short/long	1 1 1 0 1 0 1 1 | disp-low	7+m	7+m		18
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	7+m	7+m		18
Register/memory indirect within segment	1 1 1 1 1 1 1 1 | mod 1 0 0 r/m	7+m,11+m*	7+m,11+m*	2	9,18
Direct intersegment	1 1 1 0 1 0 1 0 | segment offset | segment selector	11+m	23+m		11,12,18
Protected Mode Only (Direct intersegment):					
Via call gate to same privilege level			38+m		8,11,12,18
Via TSS			175+m		8,11,12,18
Via task gate			180+m		8,11,12,18
Indirect intersegment	1 1 1 1 1 1 1 1 | mod 1 0 1 r/m (mod ≠ 11)	15+m*	26+m*	2	8,9,11,12,18
Protected Mode Only (Indirect intersegment):					
Via call gate to same privilege level			41+m*		8,9,11,12,18
Via TSS			178+m*		8,9,11,12,18
Via task gate			183+m*		8,9,11,12,18
RET = Return from CALL:					
Within segment	1 1 0 0 0 0 1 1	11+m	11+m	2	8,9,18
Within seg adding immed to SP	1 1 0 0 0 0 1 0 | data-low | data-high	11+m	11+m	2	8,9,18
Intersegment	1 1 0 0 1 0 1 1	15+m	25+m	2	8,9,11,12,18
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0 | data-low | data-high	15+m		2	8,9,11,12,18
Protected Mode Only (RET):					
To different privilege level			55+m		9,11,12,18

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT: Real Address Mode	CLOCK COUNT: Protected Virtual Address Mode	COMMENTS: Real Address Mode	COMMENTS: Protected Virtual Address Mode
CONTROL TRANSFER (Continued):					
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0 | disp	7+m or 3	7+m or 3		18
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0 | disp	7+m or 3	7+m or 3		18
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0 | disp	7+m or 3	7+m or 3		18
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0 | disp	7+m or 3	7+m or 3		18
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0 | disp	7+m or 3	7+m or 3		18
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0 | disp	7+m or 3	7+m or 3		18
JO = Jump on overflow	0 1 1 1 0 0 0 0 | disp	7+m or 3	7+m or 3		18
JS = Jump on sign	0 1 1 1 1 0 0 0 | disp	7+m or 3	7+m or 3		18
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1 | disp	7+m or 3	7+m or 3		18
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1 | disp	7+m or 3	7+m or 3		18
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1 | disp	7+m or 3	7+m or 3		18
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1 | disp	7+m or 3	7+m or 3		18
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1 | disp	7+m or 3	7+m or 3		18
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1 | disp	7+m or 3	7+m or 3		18
JNO = Jump on not overflow	0 1 1 1 0 0 0 1 | disp	7+m or 3	7+m or 3		18
JNS = Jump on not sign	0 1 1 1 1 0 0 1 | disp	7+m or 3	7+m or 3		18
LOOP = Loop CX times	1 1 1 0 0 0 1 0 | disp	8+m or 4	8+m or 4		18
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1 | disp	8+m or 4	8+m or 4		18
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0 | disp	8+m or 4	8+m or 4		18
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1 | disp	8+m or 4	8+m or 4		18
ENTER = Enter Procedure	1 1 0 0 1 0 0 0 | data-low | data-high | L				
L = 0		11	15	2,8	8,8
L = 1		15	25	2,8	8,8
L > 1		18+4(L − 1)	38+4(L − 1)	2,8	8,8
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	5	5	2,8	8,8
INT = Interrupt:					
Type specified	1 1 0 0 1 1 0 1 | type	23+m		2,7,8	
Type 3	1 1 0 0 1 1 0 0	23+m		2,7,8	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	24+m or 3 (3 if no interrupt)	(3 if no interrupt)	2,6,8	
Protected Mode Only:					
Via interrupt or trap gate to same privilege level			40+m		7,8,11,12,18
Via interrupt or trap gate to fit different privilege level			78+m		7,8,11,12,18
Via Task Gate			167+m		7,8,11,12,18
IRET = Interrupt return	1 1 0 0 1 1 1 1	17+m	31+m	2,4	8,9,11,12,15,18
Protected Mode Only:					
To different privilege level			55+m		8,9,11,12,15,18
To different task (NT = 1)			169+m		8,9,11,12,18
BOUND = Detect value out of range	0 1 1 0 0 0 1 0 | mod reg r/m	13*	13* (Use INT clock count if exception 5)	2,6	8,8,11,12,18

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT	CLOCK COUNT		COMMENTS	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROCESSOR CONTROL					
CLC = Clear carry	1 1 1 1 1 0 0 0	2	2		
CMC = Complement carry	1 1 1 1 0 1 0 1	2	2		
STC = Set carry	1 1 1 1 1 0 0 1	2	2		
CLD = Clear direction	1 1 1 1 1 1 0 0	2	2		
STD = Set direction	1 1 1 1 1 1 0 1	2	2		
CLI = Clear interrupt	1 1 1 1 1 0 1 0	3	3		14
STI = Set interrupt	1 1 1 1 1 0 1 1	2	2		14
HLT = Halt	1 1 1 1 0 1 0 0	2	2		13
WAIT = Wait	1 0 0 1 1 0 1 1	3	3		
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0	0	0		14
CTS = Clear task switched flag	0 0 0 0 1 1 1 1 | 0 0 0 0 0 1 1 0	2	2	3	13
ESC = Processor Extension Escape	1 1 0 1 1 T T T | mod LLL r/m (TTT LLL are opcode to processor extension)	9-20*	9-20*	5,8	8,17
SEG = Segment Override Prefix	001 reg 110	0	0		
PROTECTION CONTROL					
LGDT = Load global descriptor table register	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 0 1 0 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 0 0 0 r/m	11*	11*	2,3	9
LIDT = Load interrupt descriptor table register	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 0 1 1 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 0 0 1 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 1 0 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 0 0 r/m		2,3*	1	9
LTR = Load task register from register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 1 1 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 0 0 1 r/m		2,3*	1	9
LMSW = Load machine status word from register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 1 1 0 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 1 | mod 1 0 0 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 1 0 | mod reg r/m		14,16*	1	9,11,16
LSL = Load segment limit from register memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 1 1 | mod reg r/m		14,16*	1	9,11,16
ARPL = Adjust requested privilege level from register memory	0 1 1 0 0 0 1 1 | mod reg r/m		10*,11*	2	8,9
VERR = Verify read access: register/memory	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 1 0 0 r/m		14,16*	1	9,11,16
VERR = Verify write access	0 0 0 0 1 1 1 1 | 0 0 0 0 0 0 0 0 | mod 1 0 1 r/m		14,16*	1	9,11,16

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Footnotes

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

SEGMENT OVERRIDE PREFIX

0	0	1	reg	1	1	0

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.

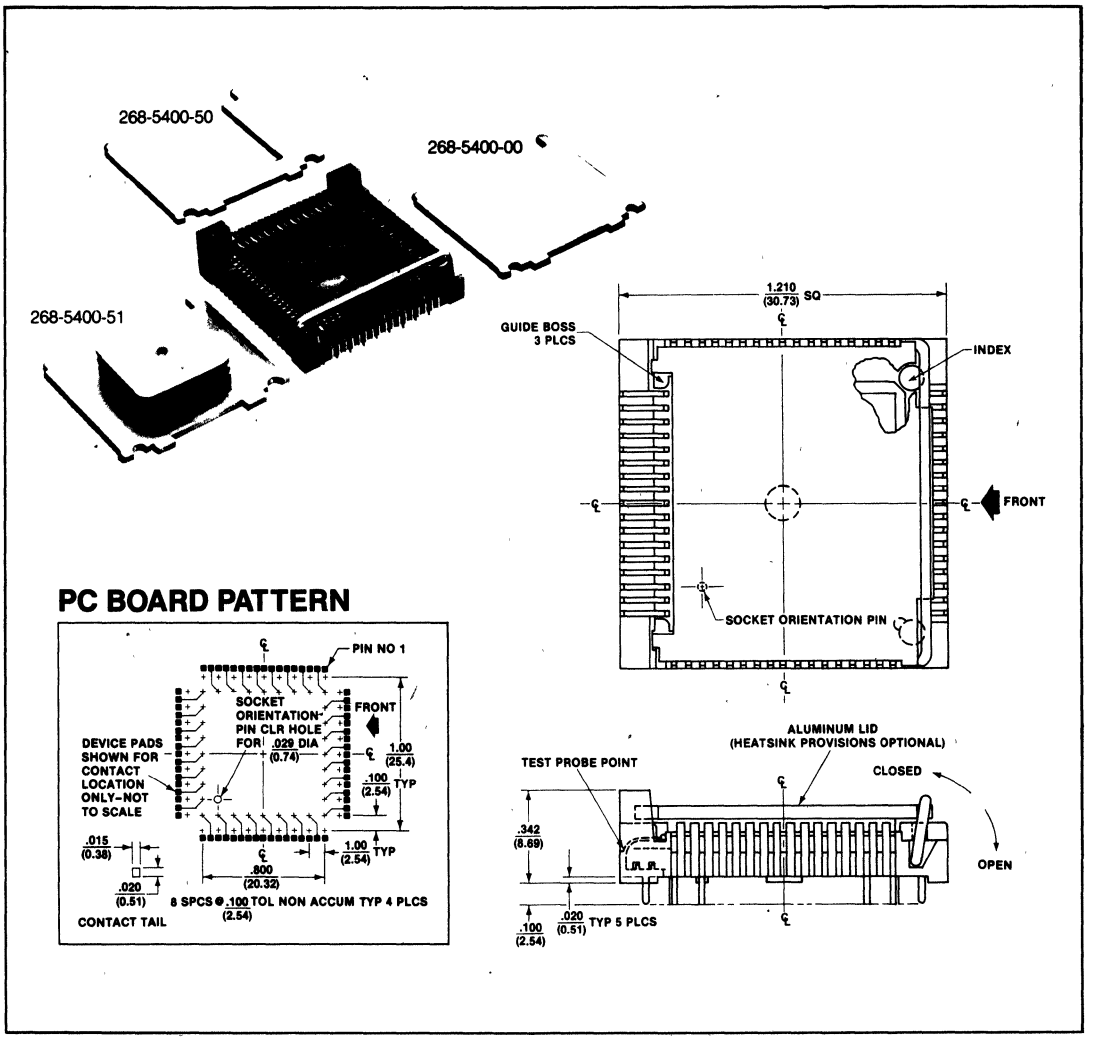


Figure 36. Textool 68 Lead Chip Carrier Socket

80287
80-Bit HMOS
NUMERIC PROCESSOR EXTENSION
80287-3

- **High Performance 80-Bit Internal Architecture**
- **Implements Proposed IEEE Floating Point Standard 754**
- **Expands iAPX 286/10 Datatypes to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Object Code Compatible with 8087**
- **Built-in Exception Handling**
- **Operates in Both Real and Protected Mode iAPX 286 Systems**
- **Protected Mode Operation Completely Conforms to the iAPX 286 Memory Management and Protection Mechanisms**
- **Directly Extends iAPX 286/10 Instruction Set to Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Datatypes**
- **8x80-Bit, Individually Addressable, Numeric Register Stack**
- **Available in EXPRESS—Standard Temperature Range**

The Intel® 80287 is a high performance numerics processor extension that extends the iAPX 286/10 architecture with floating point, extended integer and BCD data types. The iAPX 286/20 computing system (80286 with 80287) fully conforms to the proposed IEEE Floating Point Standard. Using a numerics oriented architecture, the 80287 adds over fifty mnemonics to the iAPX 286/20 instruction set, making the iAPX 286/20 a complete solution for high performance numeric processing. The 80287 is implemented in N-channel, depletion load, silicon gate technology (HMOS) and packaged in a 40-pin ceramic package. The iAPX 286/20 is object code compatible with the iAPX 86/20 and iAPX 88/20.

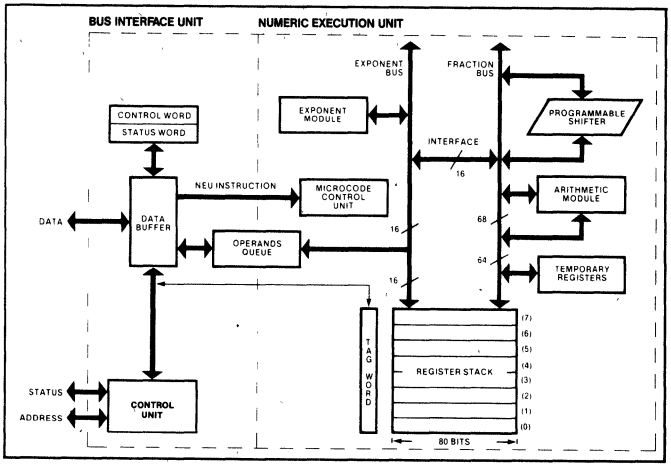
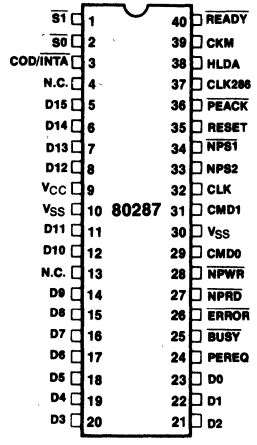


Figure 1. 80287 Block Diagram



NOTE:
N.C. PINS MUST NOT BE CONNECTED.

Figure 2. 80287 Pin Configuration

OCTOBER 1983
ORDER NUMBER: 210920-002

Table 1. 80287 Pin Description

Symbols	Type	Name and Function
CLK	I	Clock input: this clock provides the basic timing for internal 80287 operations. Special MOS level inputs are required. The 82284 or 8284A CLK outputs are compatible to this input.
CKM	I	Clock Mode signal: indicates whether CLK input is to be divided by 3 or used directly. A HIGH input will cause CLK to be used directly. This input may be connected to V_{CC} or V_{SS} as appropriate. This input must be either HIGH or LOW 20 CLK cycles before RESET goes LOW.
RESET	I	System Reset: causes the 80287 to immediately terminate its present activity and enter a dormant state. RESET is required to be HIGH for more than 4 80287 CLK cycles. For proper initialization the HIGH-LOW transition must occur no sooner than 50 μs after V_{CC} and CLK meet their D.C. and A.C. specifications.
D15-D0	I/O	Data: 16-bit bidirectional data bus. Inputs to these pins may be applied asynchronous to the 80287 clock.
$\overline{\text{BUSY}}$	O	Busy status: asserted by the 80287 to indicate that it is currently executing a command.
$\overline{\text{ERROR}}$	O	Error status: reflects the ES bit of the status word. This signal indicates that an unmasked error condition exists.
PEREQ	O	Processor Extension Data Channel operand transfer request: a HIGH on this output indicates that the 80287 is ready to transfer data. PEREQ will be disabled upon assertion of $\overline{\text{PEACK}}$ or upon actual data transfer, whichever occurs first, if no more transfers are required.
$\overline{\text{PEACK}}$	I	Processor Extension Data Channel operand transfer ACKnowledge: acknowledges that the request signal (PEREQ) has been recognized. Will cause the request (PEREQ) to be withdrawn in case there are no more transfers required. $\overline{\text{PEACK}}$ may be asynchronous to the 80287 clock.
$\overline{\text{NPRD}}$	I	Numeric Processor Read: Enables transfer of data from the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPWR}}$	I	Numeric Processor Write: Enables transfer of data to the 80287. This input may be asynchronous to the 80287 clock.
$\overline{\text{NPS1}}$, NPS2	I	Numeric Processor Selects: indicate the CPU is performing an ESCAPE instruction. Concurrent assertion of these signals (i.e., $\overline{\text{NPS1}}$ is LOW and NPS2 is HIGH) enables the 80287 to perform floating point instructions. No data transfers involving the 80287 will occur unless the device is selected via these lines. These inputs may be asynchronous to the 80287 clock.
CMD1, CMD0	I	Command lines: These, along with select inputs, allow the CPU to direct the operation of the 80287. These inputs may be asynchronous to the 80287 clock.

Table 1. 80287 Pin Description (cont.)

Symbols	Type	Name and Function
CLK286	I	CPU Clock: This input provides a sampling edge for the 80287 inputs $\overline{S1}$, $\overline{S0}$, COD/$\overline{INTA}$, $\overline{READY}$, and HLDA. It must be connected to the 80286 CLK input.
$\overline{S1}$, $\overline{S0}$ COD/$\overline{INTA}$	I	Status: These inputs must be connected to the corresponding 80286 pins.
HLDA	I	Hold Acknowledge: This input informs the 80287 when the 80286 controls the local bus. It must be connected to the 80286 HLDA output.
$\overline{READY}$	I	Ready: The end of a bus cycle is signaled by this input. It must be connected to the 80286 $\overline{READY}$ input.
V_{SS}	I	System ground, both pins must be connected to ground.
V_{CC}	I	+5V supply

FUNCTIONAL DESCRIPTION

The 80287 Numeric Processor Extension (NPX) provides arithmetic instructions for a variety of numeric data types in iAPX 286/20 systems. It also executes numerous built-in transcendental functions (e.g., tangent and log functions). The 80287 executes instructions in parallel with a 80286. It effectively extends the register and instruction set of an iAPX 286/10 system for existing iAPX 286 data types and adds several new data types as well. Figure 3 presents the program visible register model of the iAPX 286/20. Essentially, the 80287 can be treated as an additional resource or an extension to the iAPX 286/10 that can be used as a single unified system, the iAPX 286/20.

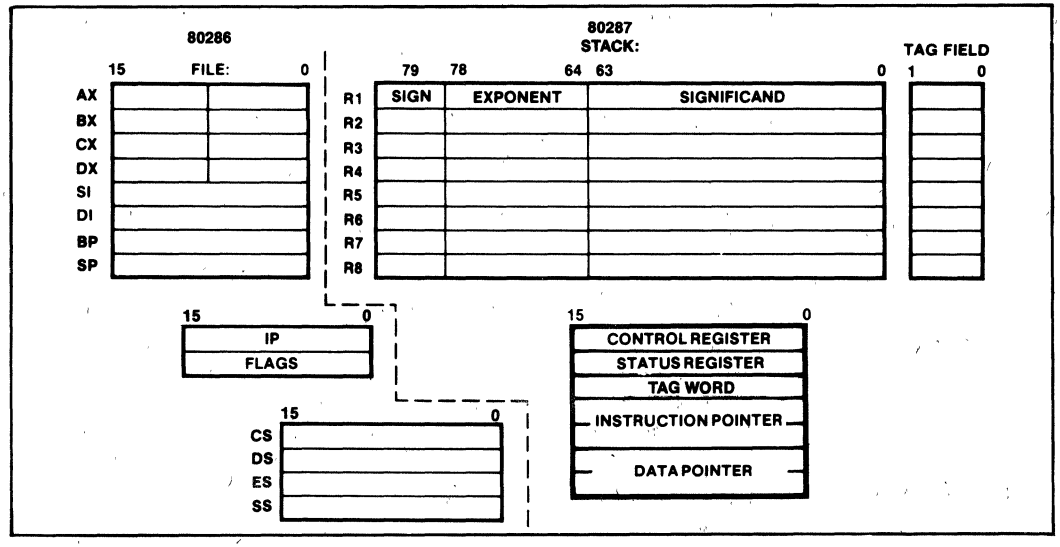


Figure 3. iAPX 286/20 Architecture

The 80287 has two operating modes similar to the two modes of the 80286. When reset, 80287 is in the real address mode. It can be placed in the protected virtual address mode by executing the SETPM ESC instruction. The 80287 cannot be switched back to the real address mode except by reset. In the real address mode, the iAPX 286/20 is completely software compatible with iAPX 86/20, 88/20.

Once in protected mode, all references to memory for numerics data or status information, obey the iAPX 286 memory management and protection rules giving a fully protected extension of the 80286 CPU. In the protected mode, iAPX 286/20 numerics software is also completely compatible with iAPX 86/20 and iAPX 88/20.

SYSTEM CONFIGURATION

As a processor extension to an 80286, the 80287 can be connected to the CPU as shown in Figure 4. The data channel control signals (PEREQ, PEACK), the BUSY signal and the NPRD, NPWR signals, allow the NPX to receive instructions and data from the CPU. When in the protected mode, all information received by the NPX is validated by the 80286 memory management and protection unit. Once started, the 80287 can process in parallel with and independent of the host CPU. When the NPX detects an error or exception, it will indicate this to the CPU by asserting the ERROR signal.

The NPX uses the processor extension request and acknowledge pins of the 80286 CPU to implement data transfers with memory under the protection model of the CPU. The full virtual and physical address space of the 80286 is available. Data for the 80287 in memory is addressed and represented in the same manner as for an 8087.

The 80287 can operate either directly from the CPU clock or with a dedicated clock. For operation with the CPU clock (CKM=0), the 80287 works at one-third the frequency of the system clock (i.e., for an 8 MHz 80286, the 16 MHz system clock is divided down to 5.3 MHz). The 80287 provides a capability to internally divide the CPU clock by three to produce the required internal clock (33% duty cycle). To use a higher performance 80287 (8 MHz), an 8284A clock driver and appropriate crystal may be used to directly drive the 80287 with a 1/3 duty cycle clock on the CLK input (CKM=1).

HARDWARE INTERFACE

Communication of instructions and data operands between the 80286 and 80287 is handled by the CMD0, CMD1, NPS1, NPS2, NPRD, and NPWR signals. I/O port addresses 00F8H, 00FAH, and 00FCH are used by the 80286 for this communication. When any of these addresses are used, the NPS1 input must be LOW and NPS2 input HIGH. The IORC and IOWC outputs of the 82288 identify I/O space transfers (see Figure 4). CMD0 should be connected to latched 80286 A1 and CMD1 should be connected to latched 80286 A2. The S1, S0, COD/INTA, READY, HLDA, and CLK pins of the 80286 are connected to the same named pins on the 80287.

I/O ports 00F8H to 00FFH are reserved for the 80286/80287 interface. To guarantee correct operation of the 80287, programs must not perform any I/O operations to these ports.

The PEREQ, PEACK, BUSY, and ERROR signals of the 80287 are connected to the same-named 80286 input. The data pins of the 80287 should be directly connected to the 80286 data bus. Note that all bus drivers connected to the 80286 local bus must be inhibited when the 80286 reads from the 80287. The use of COD/INTA and M/IO in the decoder prevents INTA bus cycles from disabling the data transceivers.

PROGRAMMING INTERFACE

Table 2 lists the seven data types the 80287 supports and presents the format for each type. These values are stored in memory with the least significant digits at the lowest memory address. Programs retrieve these values by generating the lowest address. All values should start at even addresses for maximum system performance.

Internally the 80287 holds all numbers in the temporary real format. Load instructions automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating point number or 18-digit packed BCD numbers into temporary real format. Store instructions perform the reverse type conversion.

80287 computations use the processor's register stack. These eight 80-bit registers provide the equivalent capacity of 40 16-bit registers. The 80287 register set can be accessed as a stack, with

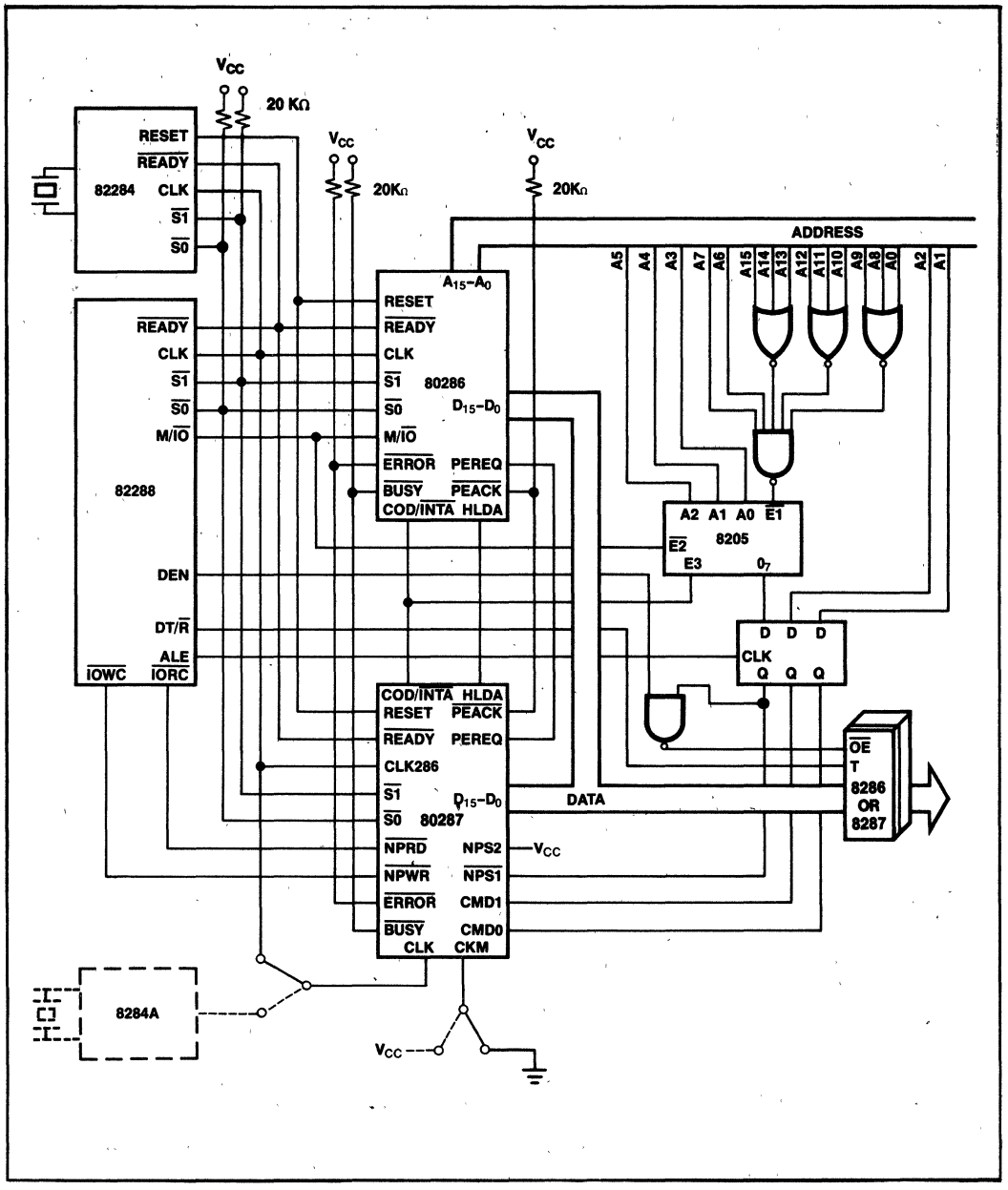
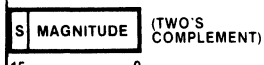
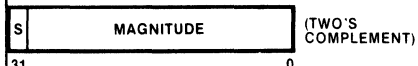
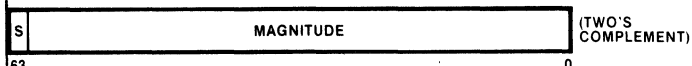
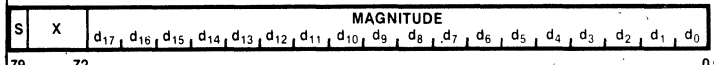
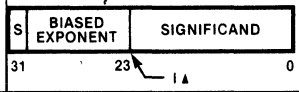
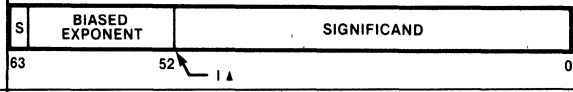
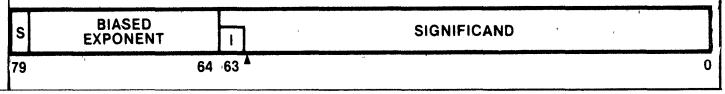


Figure 4. iAPX 286/20 System Configuration

Table 2. 80287 Datatype Representation In Memory

Data Formats	Range	Precision	Most Significant Byte ... HIGHEST ADDRESSED BYTE
Word Integer	10^4	16 Bits	
Short Integer	10^9	32 Bits	
Long Integer	10^{19}	64 Bits	
Packed BCD	10^{18}	18 Digits	
Short Real	$10^{\pm 38}$	24 Bits	
Long Real	$10^{\pm 308}$	53 Bits	
Temporary Real	$10^{\pm 4932}$	64 Bits	

NOTES:
(1) S = Sign bit (0 = positive, 1 = negative)
(2) d_n = Decimal digit (two per byte)
(3) X = Bits have no significance; 8087 ignores when loading, zeros when storing.
(4) ▲ = Position of implicit binary point
(5) I = Integer bit of significand; stored in temporary real, implicit in short and long real
(6) Exponent Bias (normalized values):
Short Real: 127 (7FH)
Long Real: 1023 (3FFH)
Temporary Real: 16383 (3FFFH)
(7) Packed BCD: $(-1)^S(D_{17}...D_0)$
(8) Real: $(-1)^S(2^{E\text{-}BIAS})(F_0\ F_1...)$

instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers.

Table 6 lists the 80287's instructions by class. No special programming tools are necessary to use the 80287 since all new instructions and data types are directly supported by the iAPX 286 assembler and appropriate high level languages. All iAPX 86/88 development tools which support the 8087 can also be used to develop software for the iAPX 286/20 in real address mode.

Table 3 gives the execution times of some typical numeric instructions.

Table 3. Execution Time for Selected 80287 Instructions

Floating Point Instruction	Approximate Execution Time (μs)
	80287 (5 MHz Operation)
Add/Subtract	14/18
Multiply (single precision)	19
Multiply (extended precision)	27
Divide	39
Compare	9
Load (double precision)	10
Store (double precision)	21
Square Root	36
Tangent	90
Exponentiation	100

SOFTWARE INTERFACE

The iAPX 286/20 is programmed as a single processor. All communication between the 80286 and the 80287 is transparent to software. The CPU automatically controls the 80287 whenever a numeric instruction is executed. All memory addressing modes, physical memory, and virtual memory of the CPU are available for use by the NPX.

Since the NPX operates in parallel with the CPU, any errors detected by the NPX may be reported after the CPU has executed the ESCAPE instruction which caused it. To allow identification of the failing numeric instruction, the NPX contains two pointer registers which identify the address of the failing numeric instruction and the numeric memory operand if appropriate for the instruction encountering this error.

INTERRUPT DESCRIPTION

Several interrupts of the iAPX 286 are used to report exceptional conditions while executing numeric programs in either real or protected mode. The interrupts and their functions are shown in Table 4.

PROCESSOR ARCHITECTURE

As shown in Figure 1, the NPX is internally divided into two processing elements, the bus interface unit (BIU) and the numeric execution unit (NEU). The NEU executes all numeric instructions, while the BIU receives and decodes instructions, requests operand transfers to and from memory and executes processor control instructions. The two units are able to operate independently of one another allowing the BIU to maintain asynchronous communication with the CPU while the NEU is busy processing a numeric instruction.

BUS INTERFACE UNIT

The BIU decodes the ESC instruction executed by the CPU. If the ESC code defines a math instruction, the BIU transmits the formatted instruction to the NEU. If the ESC code defines an administrative instruction, the BIU executes it independently of the NEU. The parallel operation of the NPX with the CPU is normally transparant to the user. The BIU generates the $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ signals for 80826/80287 processor synchronization and error notification, respectively.

The 80287 executes a single numeric instruction at a time. When executing most ESC instructions, the

Table 4. 80286 Interrupt Vectors Reserved for NPX

Interrupt Number	Interrupt Function
7	An ESC instruction was encountered when EM or TS of the 80286 MSW was set. EM=1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction will cause interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	The second or subsequent words of a numeric operand in memory exceeded a segment's limit. This interrupt occurs after executing an ESC instruction. The saved return address will not point at the numeric instruction causing this interrupt. After processing the addressing error, the iAPX 286 program can be restarted at the return address with IRET. The address of the failing numeric instruction and numeric operand are saved in the 80287. An interrupt handler for this interrupt *must* execute FNINIT before *any* other ESC or WAIT instruction.
13	The starting address of a numeric operand is not in the segment's limit. The return address will point at the ESC instruction, including prefixes, causing this error. The 80287 has not executed this instruction. The instruction and data address in 80287 refer to a previous, correctly executed, instruction.
16	The previous numeric instruction caused an unmasked numeric error. The address of the faulty numeric instruction or numeric data operand is stored in the 80287. Only ESC or WAIT instructions can cause this interrupt. The 80286 return address will point at a WAIT or ESC instruction, including prefixes, which may be restarted after clearing the error condition in the NPX.

80286 tests the $\overline{\text{BUSY}}$ pin and waits until the 80287 indicates that it is not busy before initiating the command. Once initiated, the 80286 continues program execution while the 80287 executes the ESC instruction. In iAPX 86/20 systems, this synchronization is achieved by placing a WAIT instruction before an ESC instruction. For most ESC instructions, the iAPX 286/20 does not require a WAIT instruction before the ESC opcode. However, the iAPX 286/20 will operate correctly with these WAIT instructions. In all cases, a WAIT or ESC instruction should be inserted after any 80287 store to memory (except FSTSW and FSTCW) or load from memory (except FLDENV or FRSTOR) before the 80286 reads or changes the value to be sure the numeric value has already been written or read by the NPX.

Data transfers between memory and the 80287, when needed, are controlled by the PEREQ $\overline{\text{PEACK}}$, $\overline{\text{NPRD}}$, $\overline{\text{NPWR}}$, $\overline{\text{NPS1}}$, NPS2 signals. The 80286 does the actual data transfer with memory through its processor extension data channel. Numeric data transfers with memory performed by the 80286 use the same timing as any other bus cycle. Control signals for the 80287 are generated by the 80826 as shown in Figure 4, and meet the timing requirements shown in the AC requirements section.

NUMERIC EXECUTION UNIT

The NEU executes all instructions that involve the register stack; these include arithmetic, logical, transcendental, constant and data transfer instructions. The data path in the NEU is 84 bits wide (68 significand bits, 15 exponent bits and a sign bit) which allows internal operand transfers to be performed at very high speeds.

When the NEU begins executing an instruction, it activates the BIU $\overline{\text{BUSY}}$ signal. This signal is used in conjunction with the CPU WAIT instruction or automatically with most of the ESC instructions to synchronize both processors.

REGISTER SET

The 80287 register set is shown in Figure 5. Each of the eight data registers in the 8087's register stack

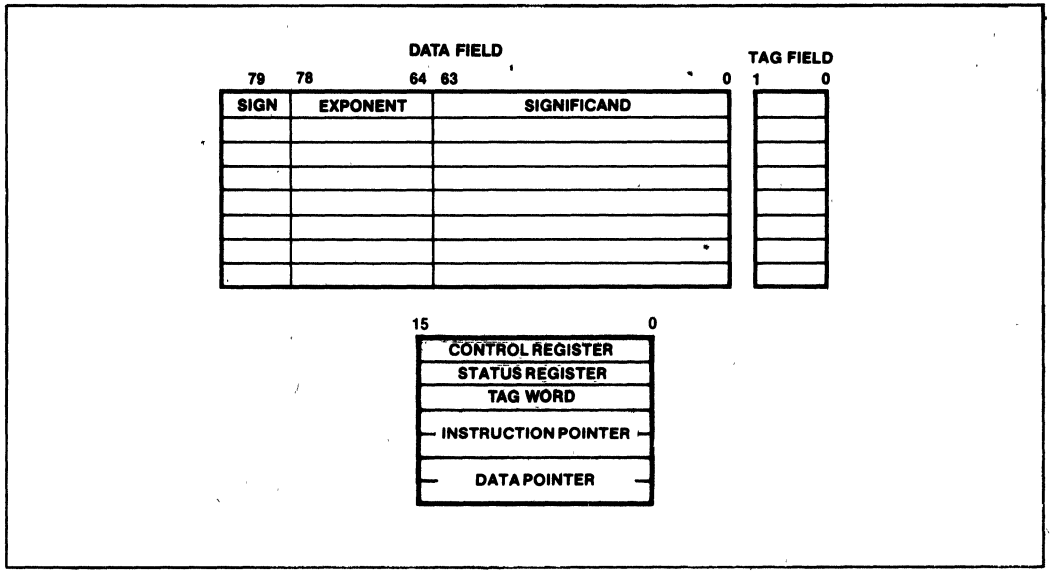


Figure 5. 80287 Register Set

is 80 bits wide and is divided into "fields" corresponding to the NPX's temporary real data type.

At a given point in time the TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like 80286 stacks in memory, the 80287 register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register pointed by the TOP. Other instructions allow the programmer to explicitly specify the register which is to be used. This explicit register addressing is also "top-relative."

STATUS WORD

The 16-bit status word (in the status register) shown in Figure 6 reflects the overall state of the 80287. It may be read and inspected by CPU code. The busy bit (bit 15) indicates whether the NEU is executing an instruction (B = 1) or is idle (B = 0).

The instructions FSTSW, FSTSW AX, FSTENV, and FSAVE which store the status word are executed exclusively by the BIU and do not set the busy bit themselves or require the Busy bit be cleared in order to be executed.

The four numeric condition code bits (C_0-C_3) are similar to the flags in a CPU: instructions that perform arithmetic operations update these bits to reflect the outcome of NPX operations. The effect of these instructions on the condition code bits is summarized in Tables 5a and 5b.

Bits 14-12 of the status word point to the 80287 register that is the current top-of-stack (TOP) as described above. Figure 6 shows the six error flags in bits 5-0 of the status word. Bits 5-0 are set to indicate that the NEU has detected an exception while executing an instruction. The section on exception handling explains how they are set and used.

Bit 7 is the error summary status bit. This bit is set if any unmasked exception bit is set and cleared otherwise. If this bit is set, the ERROR signal is asserted.

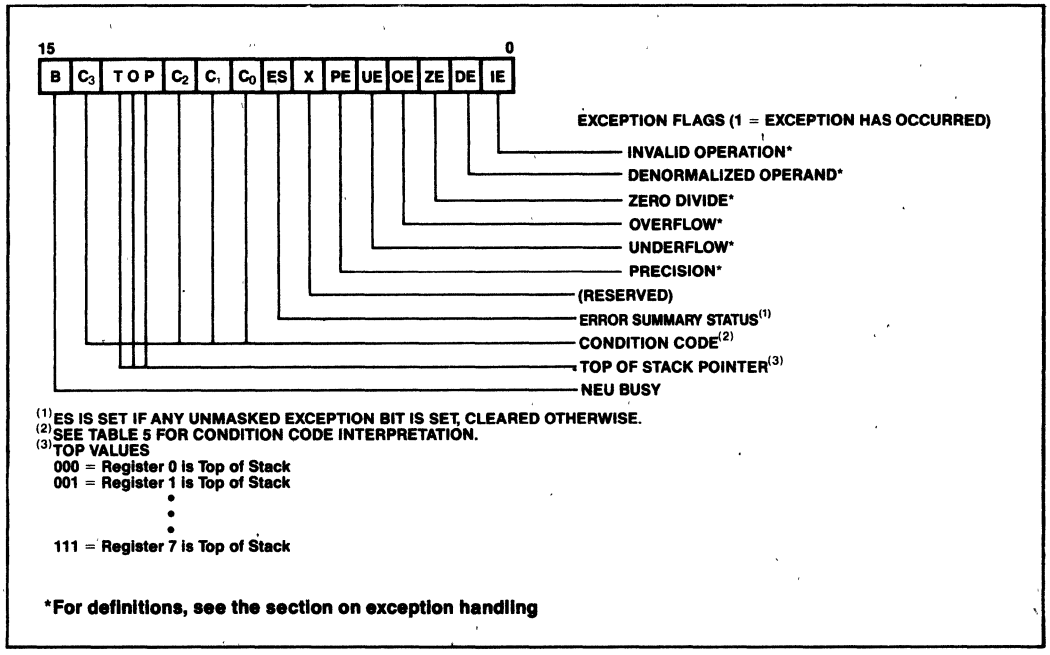


Figure 6. 80287 Status Word

TAG WORD

The tag word marks the content of each register as shown in Figure 7. The principal function of the tag word is to optimize the NPX's performance. The eight two-bit tags in the tag word can be used, however, to interpret the contents of 80287 registers.

INSTRUCTION AND DATA POINTERS

The instruction and data pointers (See Figures 8a and 8b) are provided for user-written error handlers. Whenever the 80287 executes a new instruction, the BIU saves the instruction address, the operand address (if present) and the instruction opcode. 80287 instructions can store this data into memory.

The instruction and data pointers appear in one of two formats depending on the operating mode of the 80287. In real mode, these values are the 20-bit physical address and 11-bit opcode formatted like the 8087. In protected mode, these values are the 32-bit virtual addresses used by the program which executed an ESC instruction. The same FLDENV/FSTENV/FSAVE/FRSTOR instructions as those of the 8087 are used to transfer these values between the 80287 registers and memory.

The saved instruction address in the 80287 will point at any prefixes which preceded the instruction. This is different than in the 8087 which only pointed at the ESCAPE instruction opcode.

CONTROL WORD

The NPX provides several processing options which are selected by loading a word from memory into the control word. Figure 9 shows the format and encoding of fields in the control word.

The low order byte of this control word configures the 80287 error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the 80287 recognizes. The high order byte of the control word configures the 80287 operating mode including precision,

Table 5a. Condition Code Interpretation

Instruction Type	C_3	C_2	C_1	C_0	Interpretation
Compare, Test	0	0	X	0	ST > Source or 0 (FTST)
	0	0	X	1	ST < Source or 0 (FTST)
	1	0	X	0	ST = Source or 0 (FTST)
	1	1	X	1	ST is not comparable
Remainder	Q_1	0	Q_0	Q_2	Complete reduction with three low bits of quotient (See Table 5b)
	U	1	U	U	Incomplete Reduction
Examine	0	0	0	0	Valid, positive unnormalized
	0	0	0	1	Invalid, positive, exponent =0
	0	0	1	0	Valid, negative, unnormalized
	0	0	1	1	Invalid, negative, exponent =0
	0	1	0	0	Valid, positive, normalized
	0	1	0	1	Infinity, positive
	0	1	1	0	Valid, negative, normalized
	0	1	1	1	Infinity, negative
	1	0	0	0	Zero, positive
	1	0	0	1	Empty
	1	0	1	0	Zero, negative
	1	0	1	1	Empty
	1	1	0	0	Invalid, positive, exponent = 0
	1	1	0	1	Empty
	1	1	1	0	Invalid, negative, exponent = 0
	1	1	1	1	Empty

NOTES:
1. ST = Top of stack
2. X = value is not affected by instruction
3. U = value is undefined following instruction
4. Q_n = Quotient bit n

Table 5b. Condition Code Interpretation after FPREM Instruction As a Function of Dividend Value

Dividend Range	Q_2	Q_1	Q_0
Dividend < 2 * Modulus	C_3	C_1	Q_0
Dividend < 4 * Modulus	C_3	Q_1	Q_0
Dividend ≥ 4 * Modulus	Q_2	Q_1	Q_0

NOTE:
1. Previous value of indicated bit, not affected by FPREM instruction execution.

rounding, and infinity control. The precision control bits (bits 9–8) can be used to set the 80287 internal operating precision at less than the default of temporary real (80-bit) precision. This can be useful in providing compatibility with the early generation arithmetic processors of smaller precision than the 80287. The rounding control bits (bits 11–10) provide for directed rounding and true chop as well as the unbiased round to nearest even mode specified in the IEEE standard. Control over closure of the number space at infinity is also provided (either affine closure: ± ∞, or projective closure: ∞, is treated as unsigned, may be specified).

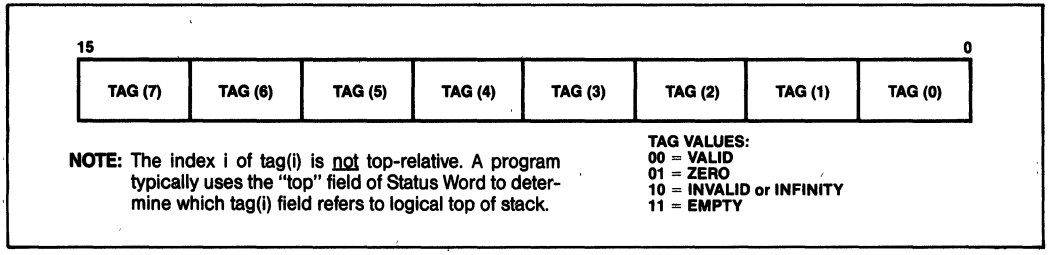


Figure 7. 80287 Tag Word

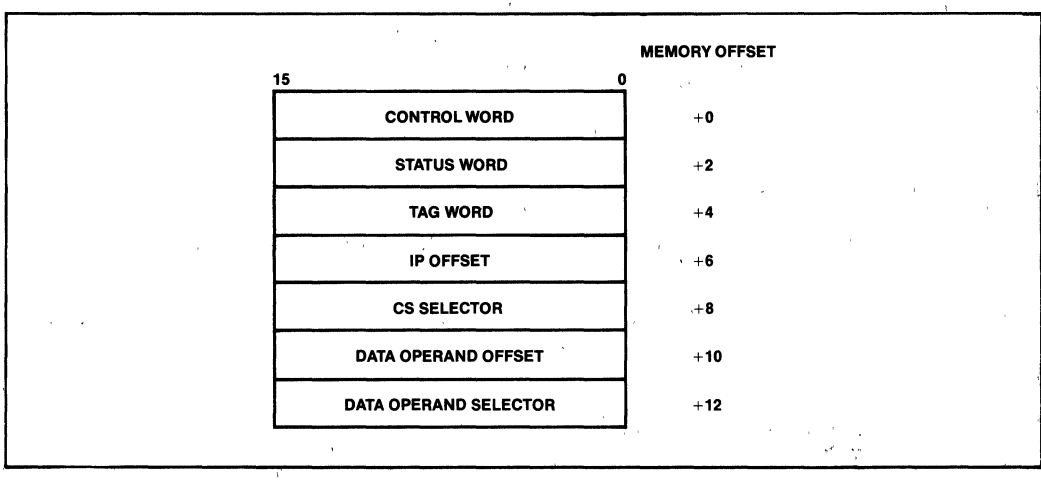


Figure 8a. Protected Mode 80287 Instruction and Data Pointer Image in Memory

EXCEPTION HANDLING

The 80287 detects six different exception conditions that can occur during instruction execution. Any or all exceptions will cause the assertion of external ERROR signal and ES bit of the Status Word if the appropriate exception masks are not set.

The exceptions that the 80287 detects and the 'default' procedures that will be carried out if the exception is masked, are as follows:

Invalid Operation: Stack overflow, stack underflow, indeterminate form (0/0, ∞, $-\infty$, etc) or the use of a Non-Number (NAN) as an operand. An exponent value of all ones and non-zero significand is reserved to identify NANs. If this exception is masked, the 80287 default response is to generate a specific NAN called INDEFINITE, or to propogate already existing NANs as the calculation result.

Overflow: The result is too large in magnitude to fit the specified format. The 80287 will generate an encoding for infinity if this exception is masked.

Zero Divisor: The divisor is zero while the dividend is a non-infinite, non-zero number. Again, the 80287 will generate an encoding for infinity if this exception is masked.

Underflow: The result is non-zero but too small in magnitude to fit in the specified format. If this exception is masked the 82087 will denormalize (shift right) the fraction until the exponent is in range. The process is called gradual underflow.

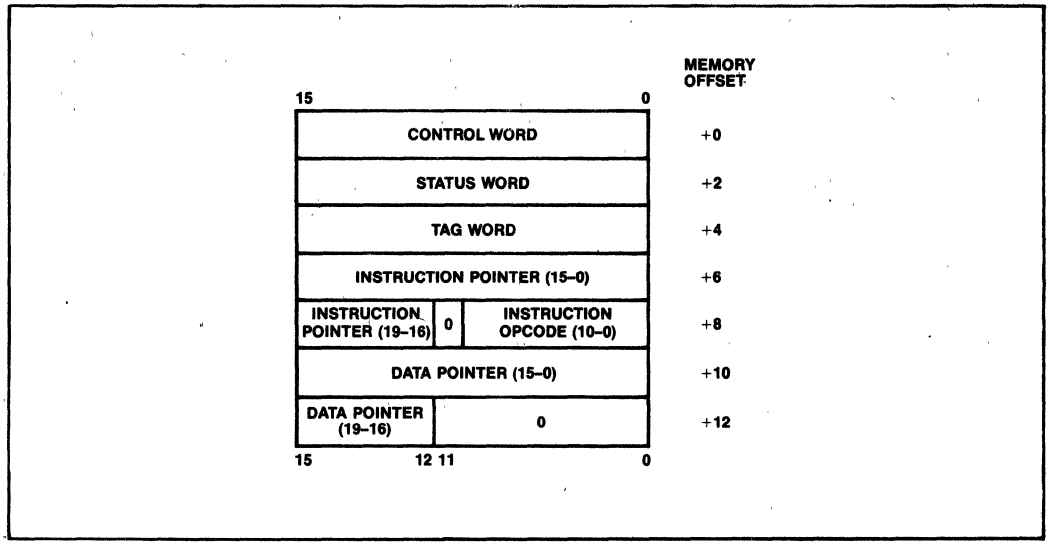


Figure 8b. Real Mode 80287 Instruction and Data Pointer Image in Memory

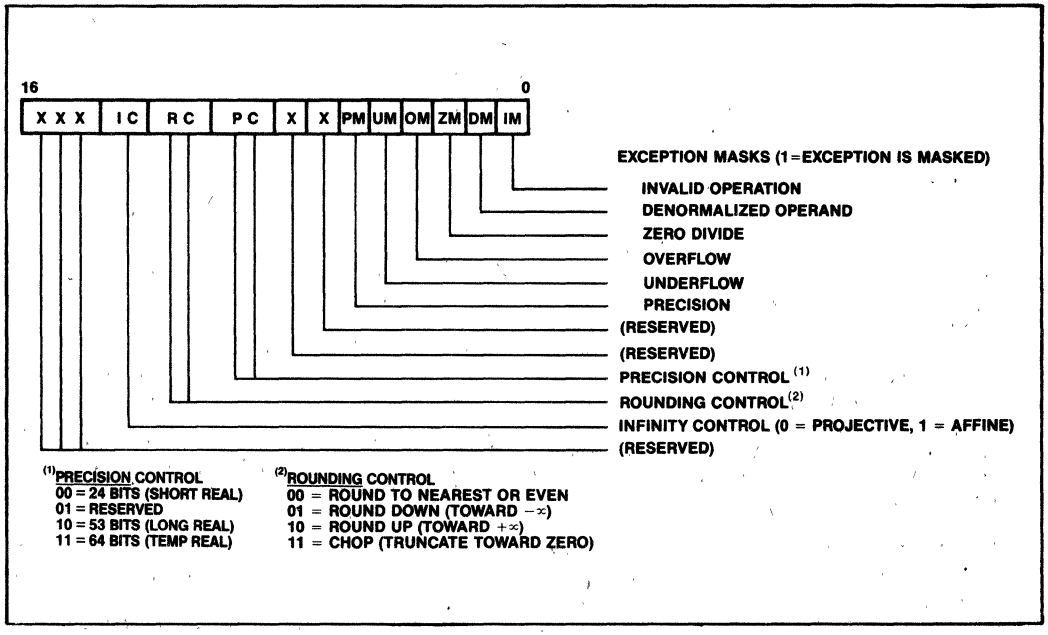


Figure 9. 80287 Control Word

Denormalized Operand: At least one of the operands is denormalized; it has the smallest exponent but a non-zero significand. Normal processing continues if this exception is masked off.

Inexact Result: The true result is not exactly representable in the specified format, the result is rounded according to the rounding mode, and this flag is set. If this exception is masked, processing will simply continue.

If the error is not masked, the corresponding error bit and the error status bit (ES) in the control word will be set, and the $\overline{\text{ERROR}}$ output signal will be asserted. If the CPU attempts to execute another ESC or WAIT instruction, exception 7 will occur.

The error condition must be resolved via an interrupt service routine. The 80287 saves the address of the floating point instruction causing the error as well as the address of the lowest memory location of any memory operand required by that instruction.

iAPX 86/20 COMPATIBILITY:

iAPX 286/20 supports portability of iAPX 86/20 programs when it is in the real address mode. However, because of differences in the numeric error handing techniques, error handling routines *may* need to be changed. The differences between an iAPX 286/20 and iAPX 86/20 are:

1. The NPX error signal does not pass through an interrupt controller (8087 INT signal does). Therefore, any interrupt controller oriented instructions for the iAPX 86/20 may have to be deleted.
2. Interrupt vector 16 must point at the numeric error handler routine.
3. The saved floating point instruction address in the 80287 includes any leading prefixes before the ESCAPE opcode. The corresponding saved address of the 8087 does not include leading prefixes.
4. In protected mode, the format of the saved instruction and operand pointers is different than for the 8087. The instruction opcode is not saved—it must be read from memory if needed.
5. Interrupt 7 will occur when executing ESC instructions with either TS or EM of MSW=1. If TS of MSW=1 then WAIT will also cause interrupt 7. An interrupt handler should be added to handle this situation.
6. Interrupt 9 will occur if the second or subsequent words of a floating point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An interrupt handler should be added to report these programming errors.

In the protected mode, iAPX 86/20 application code can be directly ported via recompilation if the 286 memory protection rules are not violated.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0 to +7V
Power Dissipation ..... 3.0 Watt

**NOTICE: Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS $T_A = 0°C$ to 70°C, $V_{CC} = 5V$, +/-5%

5 MHz

Symbol	Parameter	-3 Min	-3 max	Unit	Test Conditions
V_{IL}	Input LOW Voltage	-.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	V	
V_{ILC}	Clock Input LOW Voltage CKM = 1: CKM = 0:	 -.5 -.5	 .8 .6	 V V	
V_{IHC}	Clock Input HIGH Voltage CKM = 1: CKM = 0:	 2.0 3.8	 V_{CC} + 1 V_{CC} + 1	 V V	
V_{OL}	Output LOW Voltage		.45	V	I_{OL} = 3.0 mA
V_{OH}	Output HIGH Voltage	2.4		V	I_{OH} = -400 μA
I_{LI}	Input Leakage Current	.	±10	μA	OV ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current	.	±10	μA	.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	Power Supply Current	.	475	mA	
C_{IN}	Input Capacitance	.	10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance (D0-D15)	.	20	pF	F_C = 1 MHz
C_{CLK}	CLK Capacitance	.	12	pF	F_C = 1 MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} + 5V,=/-5%)

TIMING REQUIREMENTS

A.C. timings are referenced to 0.8V and 2.0V points on signals unless otherwise noted.

		5 MHz			
Symbol	**Parameter**	**-3 Min**	**-3 max**	**Unit**	**Test Conditions**
T_{CLCL}	CLK Period CKM = 1: CKM = 0:	 200 62.5	 500 250	 ns ns	
T_{CLCH}	CLK LOW Time CKM = 1: CKM = 0:	 118 15	 230	 ns ns	 At 0.8V At 0.6V
T_{CHCL}	CLK HIGH Time CKM = 1: CKM = 0:	 69 20	 235	 ns ns	 At 2.0V At 3.8V
T_{CH1CH2}	CLK Rise Time	.	10	ns	1.0V to3.5V if CKM = 1.
T_{CL2CL1}	CLK Fall Time	.	10	ns	3.5V to 1.0V if CKM = 1.
T_{DVWH}	Data Setup to $\overline{NPWR}$ Inactive	75	.	ns	
T_{WHDX}	Data Hold from $\overline{NPWR}$ Inactive	30	.	ns	
T_{WLWH}, T_{RLRH}	$\overline{NPWR}$, $\overline{NPRD}$ Active Time	95	.	ns	At 0.8V
T_{AVRL}, T_{AVWL}	Command Valid to $\overline{NPWR}$ or $\overline{NPRD}$ Active	0	.	ns	
T_{MHRL}	Minimum Delay from PEREQ Active to $\overline{NPRD}$ Active	130	.	ns	
T_{KLKH}	$\overline{PEACK}$ Active Time	85	.	ns	At 0.8V
T_{KHKL}	$\overline{PEACK}$ Inactive Time	250	.	ns	At 2.0V
T_{KHCH}	$\overline{PEACK}$ Inactive to $\overline{NPWR}$, $\overline{NPRD}$ Inactive	50	.	ns	
T_{CHKL}	$\overline{NPWR}$, $\overline{NPRD}$ Inactive to $\overline{PEACK}$ Active	-30	.	ns	
T_{WHAX}, T_{RHAX}	Command Hold from $\overline{NPWR}$, $\overline{NPRD}$ Inactive	30	.	ns	
T_{KLCL}	$\overline{PEACK}$ Active Setup to $\overline{NPWR}$, $\overline{NPRD}$ Active	50	.	ns	
T_{2CLCL}	CLK286 Period	62.5	.	ns	
T_{2CLCH}	CLK286 LOW Time	15	.	ns	At 0.8V
T_{2CHCL}	CLK286 HIGH Time	20	.	ns	At 2.0V
T_{2SVCL}	$\overline{S0}$, $\overline{S1}$ Setup Time to CLK286	22.5	.	ns	
T_{2CLSH}	$\overline{S0}$, $\overline{S1}$ Hold Time from CLK286	0	.	ns	

A.C. CHARACTERISTICS, continued
TIMING REQUIREMENTS

		5 MHz			
Symbol	**Parameter**	**-3 Min**	**-3 max**	**Unit**	**Test Conditions**
T_{CIVCL}	COD/$\overline{INTA}$ Setup Time to CLK286	0	.	ns	
T_{CLCIH}	COD/$\overline{INTA}$ Hold Time from CLK286	0	.	ns	
T_{RVCL}	$\overline{READY}$ Setup Time to CLK286	38.5	.	ns	
T_{CLRH}	$\overline{READY}$ Hold Time from CLK286	25	.	ns	
T_{HVCL}	HLDA Setup Time to CLK286	0	.	ns	
T_{CLHH}	HLDA Hold Time from CLK286	0	.	ns	
T_{IVCL}	$\overline{NPWR}$, $\overline{NPRD}$ to CLK Setup Time	70		ns	NOTE 1
T_{CLIH}	$\overline{NPWR}$, $\overline{NPRD}$ from CLK Hold Time	45		ns	NOTE 1
T_{RSCL}	RESET to CLK Setup Time	20		ns	NOTE 1
T_{CLRS}	RESET from CLK Hold Time	20		ns	NOTE 1

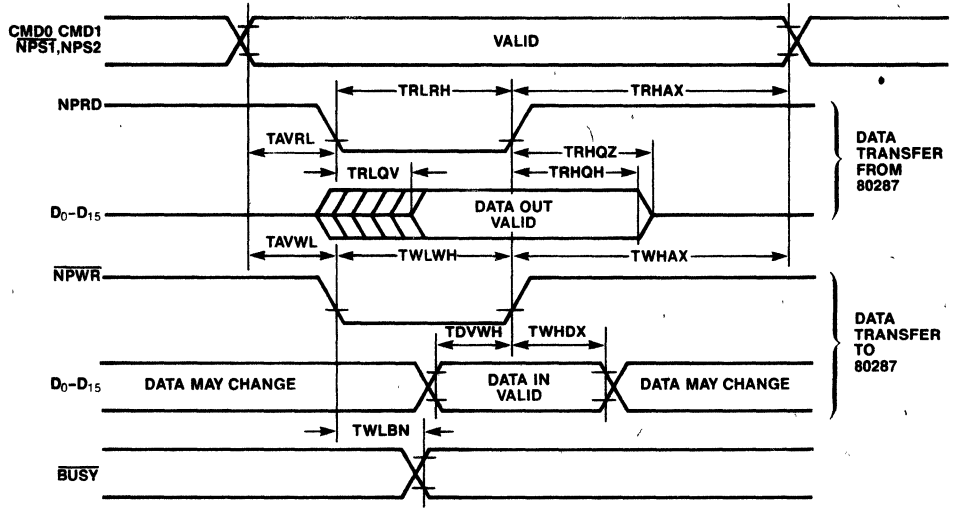
A.C. CHARACTERISTICS,
TIMING RESPONSES

		5 MHz			
Symbol	**Parameter**	**-3 Min**	**-3 max**	**Unit**	**Test Conditions**
T_{RHQZ}	$\overline{NPRD}$ Inactive to Data Float	.	37.5	ns	NOTE 2
T_{RLQV}	$\overline{NPRD}$ Active to Data Valid	.	60	ns	NOTE 3
T_{ILBH}	$\overline{ERROR}$ Active to $\overline{BUSY}$ Inactive	100	.	ns	NOTE 4
T_{WLBV}	$\overline{NPWR}$ Active to $\overline{BUSY}$ Active	.	100	ns	NOTE 5
T_{KLML}	$\overline{PEACK}$ Active to $\overline{PEREQ}$ Inactive	.	127	ns	NOTE 6
T_{CMDI}	Command Inactive Time				
	Write-to-Write	95	.	ns	At 2.0V
	Read-to-Read	250	.	ns	At 2.0V
	Write-to-Read	105	.	ns	At 2.0V
	Read-to-Write	95	.	ns	At 2.0V
T_{RHQH}	Data Hold from $\overline{NPRD}$ Inactive	5	.	ns	NOTE 7

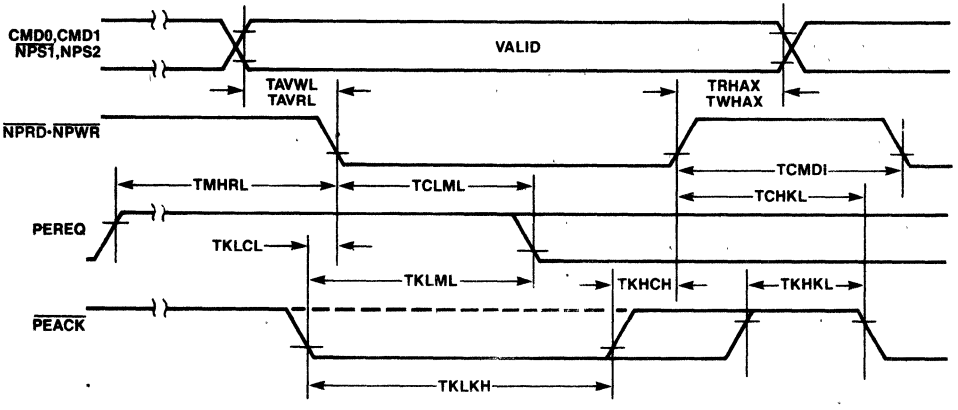
NOTES:
1. This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at a specific CLK edge.
2. Float condition occurs when output current is less than I_{LO} on D0-D15.
3. D0-D15 loading: CL = 100pF.
4. $\overline{BUSY}$ loading: CL = 100pF.
5. $\overline{BUSY}$ loading: CL = 100pF.
6. On last data transfer of numeric instruction.
7. D0-D15 loading: CL = 100pF.

WAVEFORMS (cont.)

DATA TRANSFER TIMING (INITIATED BY 80286)

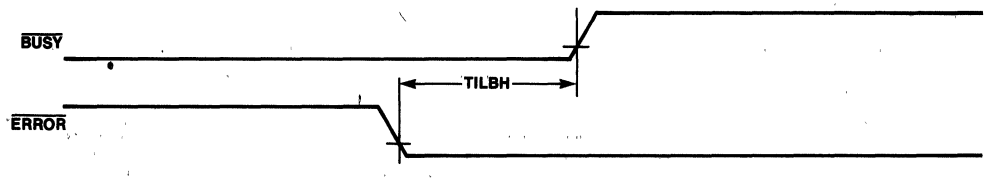


DATA CHANNEL TIMING (INITIATED BY 80287)

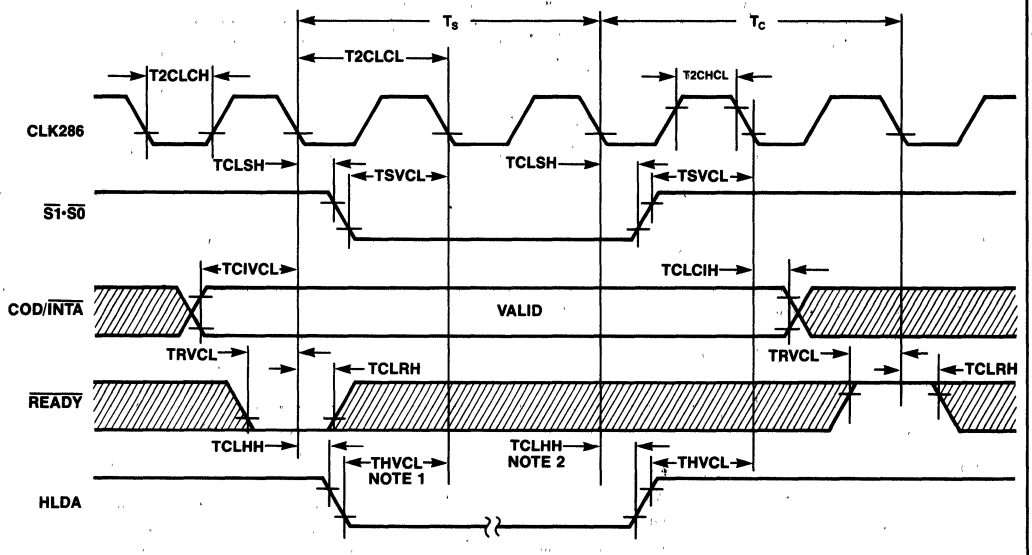


WAVEFORMS (cont.)

ERROR OUTPUT TIMING



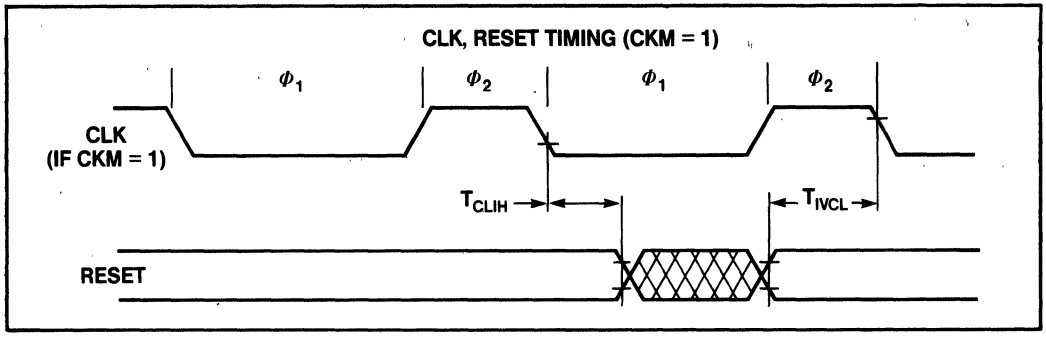
80286 STATUS TIMING

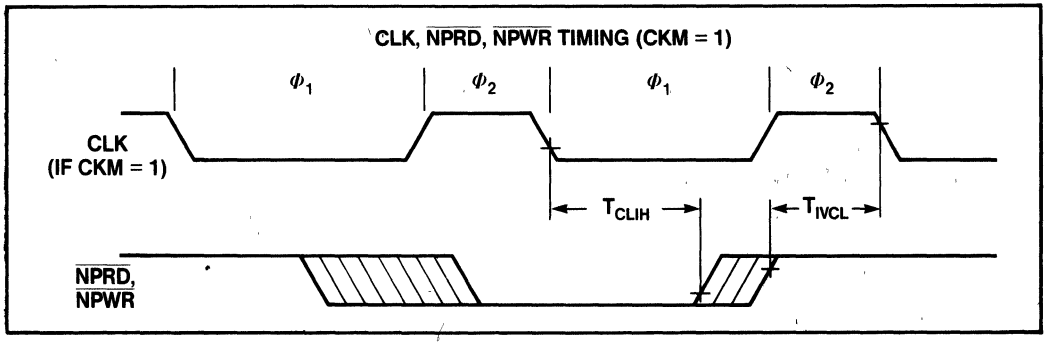


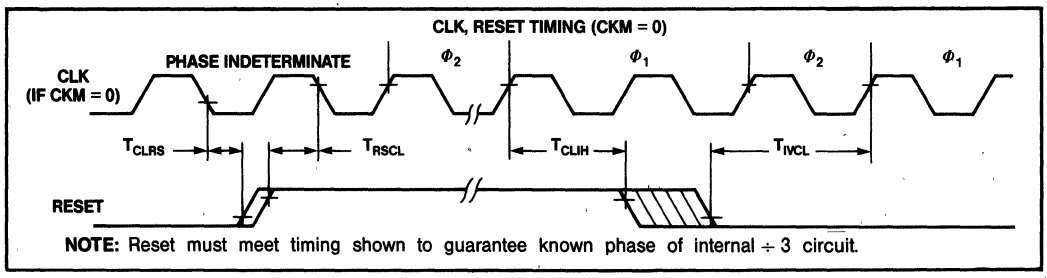
NOTES:
1. This input transition occurs before T_S.
2. This input transition occurs after T_C.

WAVEFORMS

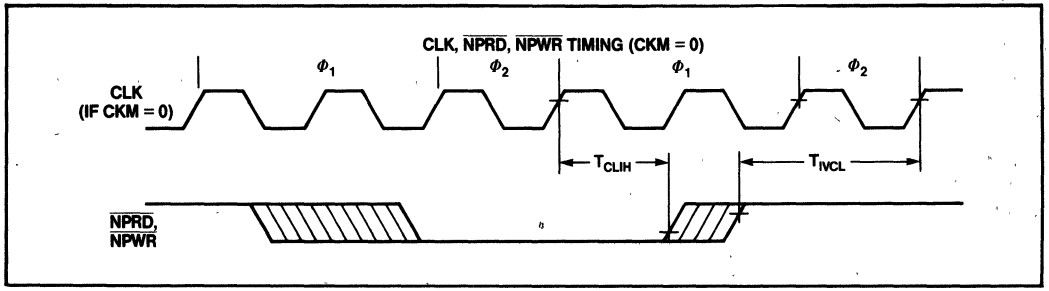
(Reset, $\overline{NPWR}$, $\overline{NPRD}$ are inputs asynchronous to CLK. Timing requirements on this page are given for testing purposes only, to assure recognition at a specific CLK edge.)







NOTE: Reset must meet timing shown to guarantee known phase of internal ÷ 3 circuit.



210920-002

Table 6. 80287 Extensions to the 80286 Instruction Set

Data Transfer		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
FLD = LOAD	MF =		00	01	10	11
Integer/Real Memory to ST(0)	ESCAPE MF 1 | MOD 0 0 0 R/M	DISP	38–56	52–60	40–60	46–54
Long Integer Memory to ST(0)	ESCAPE 1 1 1 | MOD 1 0 1 R/M	DISP	60–68			
Temporary Real Memory to ST(0)	ESCAPE 0 1 1 | MOD 1 0 1 R/M	DISP	53–65			
BCD Memory to ST(0)	ESCAPE 1 1 1 | MOD 1 0 0 R/M	DISP	290–310			
ST(i) to ST(0)	ESCAPE 0 0 1 | 1 1 0 0 0 ST(i)		17–22			
FST = STORE						
ST(0) to Integer/Real Memory	ESCAPE MF 1 | MOD 0 1 0 R/M	DISP	84–90	82–92	96–104	80–90
ST(0) to ST(i)	ESCAPE 1 0 1 | 1 1 0 1 0 ST(i)		15–22			
FSTP = STORE AND POP						
ST(0) to Integer/Real Memory	ESCAPE MF 1 | MOD 0 1 1 R/M	DISP	86–92	84–94	98–106	82–92
ST(0) to Long Integer Memory	ESCAPE 1 1 1 | MOD 1 1 1 R/M	DISP	94–105			
ST(0) to Temporary Real Memory	ESCAPE 0 1 1 | MOD 1 1 1 R/M	DISP	52–58			
ST(0) to BCD Memory	ESCAPE 1 1 1 | MOD 1 1 0 R/M	DISP	520–540			
ST(0) to ST(i)	ESCAPE 1 0 1 | 1 1 0 1 1 ST(i)		17–24			
FXCH – Exchange ST(i) and ST(0)	ESCAPE 0 0 1 | 1 1 0 0 1 ST(i)		10–15			
Comparison						
FCOM = Compare						
Integer/Real Memory to ST(0)	ESCAPE MF 0 | MOD 0 1 0 R/M	DISP	60–70	78–91	65–75	72–86
ST(i) to ST (0)	ESCAPE 0 0 0 | 1 1 0 1 0 ST(i)		40–50			
FCOMP = Compare and Pop						
Integer/Real Memory to ST(0)	ESCAPE MF 0 | MOD 0 1 1 R/M	DISP	63–73	80–93	67–77	74–88
ST(i) to ST(0)	ESCAPE 0 0 0 | 1 1 0 1 1 ST(i)		45–52			
FCOMPP = Compare ST(1) to ST(0) and Pop Twice	ESCAPE 1 1 0 | 1 1 0 1 1 0 0 1		45–55			
FTST = Test ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 1 0 0		38–48			
FXAM = Examine ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 1 0 1		12–23			

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

Constants		Optional 8,16 Bit Displacement	Clock Count Range 32 Bit Real	32 Bit Integer	64 Bit Real	16 Bit Integer
	MF =		00	01	10	11
FLDZ = LOAD + 0.0 into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 1 0		11–17			
FLD1 = LOAD + 1.0 into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 0 0		15–21			
FLDPI = LOAD π into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 1 1		16–22			
FLDL2T = LOAD $\log_2 10$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 0 1		16–22			
FLDL2E = LOAD $\log_2 e$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 0 1 0		15–21			
FLDLG2 = LOAD $\log_{10} 2$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 0 0		18–24			
FLDLN2 = LOAD $\log_e 2$ into ST(0)	ESCAPE 0 0 1 | 1 1 1 0 1 1 0 1		17–23			
Arithmetic						
FADD = Addition						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 0 0 0 R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 0 0 0 ST(i)		70–100 (Note 1)			
FSUB = Subtraction						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 1 0 R R/M	DISP	90–120	108–143	95–125	102–137
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 1 0 R R/M		70–100 (Note 1)			
FMUL = Multiplication						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 0 0 1 R/M	DISP	110–125	130–144	112–168	124–138
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 0 0 1 R/M		90–145 (Note 1)			
FDIV = Division						
Integer/Real Memory with ST(0)	ESCAPE MF 0 | MOD 1 1 R R/M	DISP	215–225	230–243	220–230	224–238
ST(i) and ST(0)	ESCAPE d P 0 | 1 1 1 1 R R/M		193–203 (Note 1)			
FSQRT = Square Root of ST(0)	ESCAPE 0 0 1 | 1 1 1 1 1 0 1 0		180–186			
FSCALE = Scale ST(0) by ST(1)	ESCAPE 0 0 1 | 1 1 1 1 1 1 0 1		32–38			
FPREM = Partial Remainder of ST(0) ÷ ST(1)	ESCAPE 0 0 1 | 1 1 1 1 1 0 0 0		15–190			
FRNDINT = Round ST(0) to Integer	ESCAPE 0 0 1 | 1 1 1 1 1 1 0 0		16–50			

NOTE:
1. If P=1 then add 5 clocks.

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

		Optional 8,16 Bit Displacement	Clock Count Range
FXTRACT = Extract Components of St(0)	ESCAPE 0 0 1 | 1 1 1 1 0 1 0 0		27–55
FABS = Absolute Value of ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 0 0 1		10–17
FCHS = Change Sign of ST(0)	ESCAPE 0 0 1 | 1 1 1 0 0 0 0 0		10–17
Transcendental			
FPTAN = Partial Tangent of ST(0)	ESCAPE 0 0 1 | 1 1 1 1 0 0 1 0		30–540
FPATAN = Partial Arctangent of ST(0) –ST(1)	ESCAPE 0 0 1 | 1 1 1 1 0 0 1 1		250–800
F2XM1 = $2^{ST(0)}-1$	ESCAPE 0 0 1 | 1 1 1 1 0 0 0 0		310–630
FYL2X = ST(1) • Log_2 [ST(0)]	ESCAPE 0 0 1 | 1 1 1 1 0 0 0 1		900–1100
FYL2XP1 = ST(1) • Log_2 [ST(0) +1]	ESCAPE 0 0 1 | 1 1 1 1 1 0 0 1		700–1000
Processor Control			
FINIT = Initialize NPX	ESCAPE 0 1 1 | 1 1 1 0 0 0 1 1		2–8
FSETPM = Enter Protected Mode	ESCAPE 0 1 1 | 1 1 1 0 0 1 0 0		2–8
FSTSW AX = Store Control Word	ESCAPE 1 1 1 | 1 1 1 0 0 0 0 0		10–16
FLDCW = Load Control Word	ESCAPE 0 0 1 | MOD 1 0 1 R/M	DISP	7–14
FSTCW = Store Control Word	ESCAPE 0 0 1 | MOD 1 1 1 R/M	DISP	12–18
FSTSW = Store Status Word	ESCAPE 1 0 1 | MOD 1 1 1 R/M	DISP	12–18
FCLEX = Clear Exceptions	ESCAPE 0 1 1 | 1 1 1 0 0 0 1 0		2–8
FSTENV = Store Environment	ESCAPE 0 0 1 | MOD 1 1 0 R/M	DISP	40–50
FLDENV = Load Environment	ESCAPE 0 0 1 | MOD 1 0 0 R/M	DISP	35–45
FSAVE = Save State	ESCAPE 1 0 1 | MOD 1 1 0 R/M	DISP	205–215
FRSTOR = Restore State	ESCAPE 1 0 1 | MOD 1 0 0 R/M	DISP	205–215
FINCSTP = Increment Stack Pointer	ESCAPE 0 0 1 | 1 1 1 1 0 1 1 1		6–12
FDECSTP = Decrement Stack Pointer	ESCAPE 0 0 1 | 1 1 1 1 0 1 1 0		6–12

Table 6. 80287 Extensions to the 80286 Instruction Set (cont.)

		Clock Count Range
FFREE = Free ST(i)	ESCAPE 1 0 1 | 1 1 0 0 0 ST(i)	9–16
FNOP = No Operation	ESCAPE 0 0 1 | 1 1 0 1 0 0 0 0	10–16

NOTES:

1. if mod=00 then DISP=0*, disp-low and disp-high are absent
   if mod=01 then DISP=disp-low sign-extended to 16-bits, disp-high is absent
   if mod=10 then DISP=disp-high; disp-low
   if mod=11 then r/m is treated as an ST(i) field
2. if r/m=000 then EA=(BX) + (SI) +DISP
   if r/m=001 then EA=(BX) + (DI) +DISP
   if r/m=010 then EA=(BP) + (SI) +DISP
   if r/m=011 then EA=(BP) + (DI) +DISP
   if r/m=100 then EA=(SI) + DISP
   if r/m=101 then EA=(DI) + DISP
   if r/m=110 then EA=(BP) + DISP
   if r/m=111 then EA=(BX) + DISP

   *except if mod=000 and r/m=110 then EA =disp-high; disp-low.
3. MF= Memory Format
   00—32-bit Real
   01—32-bit Integer
   10—64-bit Real
   11—16-bit Integer
4. ST(0)= Current stack top
   ST(i) i^{th} register below stack top
5. d= Destination
   0—Destination is ST(0)
   1—Destination is ST(i)
6. P= Pop
   0—No pop
   1—Pop ST(0)
7. R= Reverse: When d=1 reverse the sense of R
   0—Destination (op) Source
   1—Source (op) Destination
8. For **FSQRT:** $-0 \leq ST(0) \leq +\infty$
   For **FSCALE:** $-2^{15} \leq ST(1) < +2^{15}$ and ST(1) integer
   For **F2XM1:** $0 \leq ST(0) \leq 2^{-1}$
   For **FYL2X:** $0 < ST(0) < \infty$
   $-\infty < ST(1) < +\infty$
   For **FYL2XP1:** $0 \leq |ST(0)| < (2 - \sqrt{2})/2$
   $-\infty < ST(1) < \infty$
   For **FPTAN:** $0 \leq ST(0) \leq \pi/4$
   For **FPATAN:** $0 \leq ST(0) < ST(1) < +\infty$
9. ESCAPE bit pattern is 11011.

82284
CLOCK GENERATOR AND READY INTERFACE FOR iAPX 286 PROCESSORS

(82284, 82284-6)

- **Generates System Clock for iAPX 286 Processors**
- **Uses Crystal or TTL Signal for Frequency Source**
- **Provides Local READY and Multibus* READY Synchronization**
- **18-pin Package**
- **Single +5V Power Supply**
- **Generates System Reset Output from Schmitt Trigger Input**
- **Available in EXPRESS**
  - **Standard Temperature Range**
  - **Extended Temperature Range**

The 82284 is a clock generator/driver which provides clock signals for iAPX 286 processors and support components. It also contains logic to supply $\overline{\text{READY}}$ to the CPU from either asynchronous or synchronous sources and synchronous RESET from an asynchronous input with hysteresis.

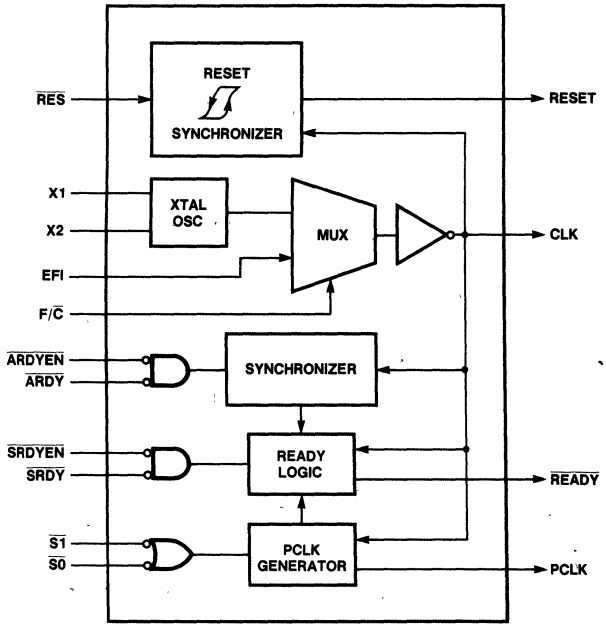


Figure 1. 82284 Block Diagram

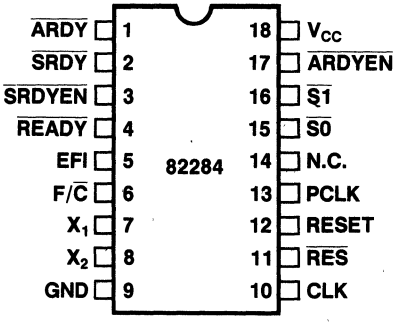


Figure 2. 82284 Pin Configuration

* Multibus is a patented bus of Intel

NOVEMBER 1983
Order Number 210453-002

Table 1. Pin Description

The following pin function descriptions are for the 82284 clock generator.

Symbol	Type	Name and Function
CLK	O	**System Clock** is the signal used by the processor and support devices which must be synchronous with the processor. The frequency of the CLK output has twice the desired internal processor clock frequency. CLK can drive both TTL and MOS level inputs.
F/$\overline{C}$	I	**Frequency/Crystal Select** is a strapping option to select the source for the CLK output. When F/$\overline{C}$ is strapped LOW, the internal crystal oscillator drives CLK. When F/$\overline{C}$ is strapped HIGH, the EFI input drives the CLK output.
X1, X2	I	**Crystal In** are the pins to which a parallel resonant fundamental mode crystal is attached for the internal oscillator. When F/$\overline{C}$ is LOW, the internal oscillator will drive the CLK output at the crystal frequency. The crystal frequency must be twice the desired internal processor clock frequency.
EFI	I	**External Frequency In** drives CLK when the F/$\overline{C}$ input is strapped HIGH. The EFI input frequency must be twice the desired internal processor clock frequency.
PCLK	O	**Peripheral Clock** is an output which provides a 50% duty cycle clock with 1/2 the frequency of CLK. PLCK will be in phase with the internal processor clock following the first bus cycle after the processor has been reset.
$\overline{ARDYEN}$	I	**Asynchronous Ready Enable** is an active LOW input which qualifies the $\overline{ARDY}$ input. $\overline{ARDYEN}$ selects $\overline{ARDY}$ as the source of ready for the current bus cycle. Inputs to $\overline{ARDYEN}$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
$\overline{ARDY}$	I	**Asynchronous Ready** is an active LOW input used to terminate the current bus cycle. The $\overline{ARDY}$ input is qualified by $\overline{ARDYEN}$. Inputs to $\overline{ARDY}$ may be applied asynchronously to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
$\overline{SRDYEN}$	I	**Synchronous Ready Enable** is an active LOW input which qualifies $\overline{SRDY}$. $\overline{SRDYEN}$ selects $\overline{SRDY}$ as the source for $\overline{READY}$ to the CPU for the current bus cycle. Setup and hold times must be satisfied for proper operation.
$\overline{SRDY}$	I	**Synchronous Ready** is an active LOW input used to terminate the current bus cycle. The $\overline{SRDY}$ input is qualified by the $\overline{SRDYEN}$ input. Setup and hold times must be satisfied for proper operation.
$\overline{READY}$	O	**Ready** is an active LOW output which signals the current bus cycle is to be completed. The $\overline{SRDY}$, $\overline{SRDYEN}$, $\overline{ARDY}$, $\overline{ARDYEN}$, $\overline{S1}$, $\overline{S0}$ and $\overline{RES}$ inputs control $\overline{READY}$ as explained later in the $\overline{READY}$ generator section. $\overline{READY}$ is an open collector output requiring an external 300 ohm pullup resistor.
$\overline{S0}$, $\overline{S1}$	I	**Status** inputs prepare the 82284 for a subsequent bus cycle. $\overline{S0}$ and $\overline{S1}$ synchronize PCLK to the internal processor clock and control $\overline{READY}$. These inputs have pullup resistors to keep them HIGH if nothing is driving them. Setup and hold times must be satisfied for proper operation.
RESET	O	**Reset** is an active HIGH output which is derived from the $\overline{RES}$ input. RESET is used to force the system into an initial state. When RESET is active, $\overline{READY}$ will be active (LOW).
$\overline{RES}$	I	**Reset In** is an active LOW input which generates the system reset signal RESET. Signals to $\overline{RES}$ may be applied asynchronously to CLK. A Schmitt trigger input is provided on $\overline{RES}$, so that an RC circuit can be used to provide a time delay. Setup and hold times are given to assure a guaranteed response to synchronous inputs.
V_{CC}		**System Power**: +5V power supply
GND		**System Ground**: 0 volts

FUNCTIONAL DESCRIPTION

Introduction

The 82284 generates the clock, ready, and reset signals required for iAPX 286 processors and support components. The 82284 is packaged in an 18-pin DIP and contains a crystal controlled oscillator, MOS clock generator, peripheral clock generator, Multibus ready synchronization logic and system reset generation logic.

Clock Generator

The CLK output provides the basic timing control for an iAPX 286 system. CLK has output characteristics sufficient to drive MOS devices. CLK is generated by either an internal crystal oscillator or an external source as selected by the F/$\overline{C}$ strapping option. When

210453-003
AFN-00786B

F/$\overline{C}$ is LOW, the crystal oscillator drives the CLK output. When F/$\overline{C}$ is HIGH, the EFI input drives the CLK output.

The 82284 provides a second clock output (PCLK) for peripheral devices. PCLK is CLK divided by two. PCLK has a duty cycle of 50% and TTL output drive characteristics. PCLK is normally synchronized to the internal processor clock.

After reset, the PCLK signal may be out of phase with the internal processor clock. The $\overline{S1}$ and $\overline{S0}$ signals of the first bus cycle are used to synchronize PCLK to the internal processor clock. The phase of the PCLK output changes by extending its HIGH time beyond one system clock (see waveforms). PCLK is forced HIGH whenever either $\overline{S0}$ or $\overline{S1}$ were active (LOW) for the two previous CLK cycles. PCLK continues to oscillate when both $\overline{S0}$ and $\overline{S1}$ are HIGH.

Since the phase of the internal processor clock will not change except during reset, the phase of PCLK will not change except during the first bus cycle after reset.

Oscillator

The oscillator circuit of the 82284 is a linear Pierce oscillator which requires an external parallel resonant, fundamental mode, crystal. The output of the oscillator is internally buffered. The crystal frequency chosen should be twice the required internal processor clock frequency. The crystal should have a typical load capacitance of 32 pF.

X1 and X2 are the oscillator crystal connections. For stable operation of the oscillator, two loading capacitors are recommended, as shown in Table 2. The sum of the board capacitance and loading capacitance should equal the values shown. It is advisable to limit stray board capacitances (not including the effect of the loading capacitors or crystal capacitance) to less than 10 pF between the X1 and X2 pines. Decouple V_{CC} and GND as close to the 82284 as possible.

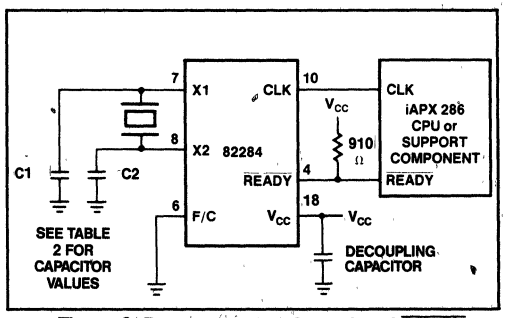


Figure 3. Recommended Crystal and $\overline{READY}$ Connections

Reset Operation

The reset logic provides the RESET output to force the system into a known, initial state. When the $\overline{RES}$ input is active (LOW), the RESET output becomes active (HIGH). $\overline{RES}$ is synchronized internally at the falling edge of CLK before generating the RESET output (see waveforms). Synchronization of the $\overline{RES}$ input introduces a one or two CLK delay before affecting the RESET output.

At power up, a system does not have have a stable V_{CC} and CLK. To prevent spurious activity, $\overline{RES}$ should be asserted until V_{CC} and CLK stabilize at their operating values. iAPX 286 processors and support components also require their RESET inputs be HIGH a minimum of 16 CLK cycles. An RC network, as shown in Figure 4, will keep $\overline{RES}$ LOW long enough to satisfy both needs.

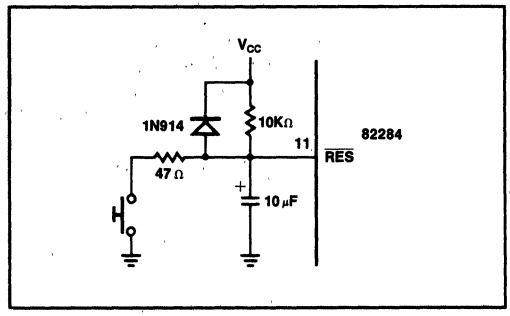


Figure 4. Typical RC $\overline{RES}$ Timing Circuit

A Schmitt trigger input with hysteresis on $\overline{RES}$ assures a single transition of RESET with an RC circuit on $\overline{RES}$. The hysteresis separates the input voltage level at which the circuit output switches between HIGH to LOW from the input voltage level at which the circuit output switches between LOW to HIGH. The $\overline{RES}$ HIGH to LOW input transition voltage is lower than the $\overline{RES}$ LOW to HIGH input transition voltage. As long as the slope of the $\overline{RES}$ input voltage remains in the same direction (increasing or decreasing) around the $\overline{RES}$ input transition voltage, the RESET output will make a single transition.

Ready Operation

The 82284 accepts two ready sources for the system ready signal which terminates the current bus cycle. Either a synchronous ($\overline{SRDY}$) or asynchronous ready ($\overline{ARDY}$) source may be used. Each ready input has an enable ($\overline{SRDYEN}$ and $\overline{ARDYEN}$) for selecting the type of ready source required to terminate the current bus cycle. An address decoder would normally select one of the enable inputs.

210453-003
AFN-00786B

$\overline{\text{READY}}$ is enabled (LOW), if either $\overline{\text{SRDY}}$ + $\overline{\text{SRDYEN}}$ = 0 or $\overline{\text{ARDY}}$ + $\overline{\text{ARDYEN}}$ = 0 when sampled by the 82284 $\overline{\text{READY}}$ generation logic. $\overline{\text{READY}}$ will remain active for at least two CLK cycles.

The $\overline{\text{READY}}$ output has an open-collector driver allowing other ready circuits to be wire or'ed with it, as shown in Figure 3. The $\overline{\text{READY}}$ signal of an iAPX 286 system requires an external 910 ohm ± 5% pull-up resistor. To force the $\overline{\text{READY}}$ signal inactive (HIGH) at the start of a bus cycle, the $\overline{\text{READY}}$ output floats when either $\overline{\text{S1}}$ or $\overline{\text{S0}}$ are sampled LOW at the falling edge of CLK. Two system clock periods are allowed for the pull-up resistor to pull the $\overline{\text{READY}}$ signal to V_{IH}. When RESET is active, $\overline{\text{READY}}$ is forced active one CLK later (see waveforms).

Figure 5 illustrates the operation of $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$. These inputs are sampled on the falling edge of CLK when $\overline{\text{S1}}$ and $\overline{\text{S0}}$ are inactive and PCLK is HIGH. $\overline{\text{READY}}$ is forced active when both $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$ are sampled as LOW.

Figure 6 shows the operation of $\overline{\text{ARDY}}$ and $\overline{\text{ARDYEN}}$. These inputs are sampled by an internal synchronizer at each falling edge of CLK. The output of the synchronizer is then sampled when PCLK is HIGH. If the synchronizer resolved both the $\overline{\text{ARDY}}$ and $\overline{\text{ARDYEN}}$ have been resolved as active, the $\overline{\text{SRDY}}$ and $\overline{\text{SRDYEN}}$ inputs are ignored. Either $\overline{\text{ARDY}}$ or $\overline{\text{ARDYEN}}$ must be HIGH at end of T_S (see figure 6).

$\overline{\text{READY}}$ remains active until either $\overline{\text{S1}}$ or $\overline{\text{S0}}$ are sampled LOW, or the ready inputs are sampled as inactive.

Table 2. 82284 Crystal Loading Capacitance Values

Crystal Frequency	C1 Capacitance (pin 7)	C2 Capacitance (pin 8)
1 to 8 MHz	60 pF	40 pF
8 to 16 MHz	25 pF	15 pF

NOTE: Capacitance values must include stray board capacitance.

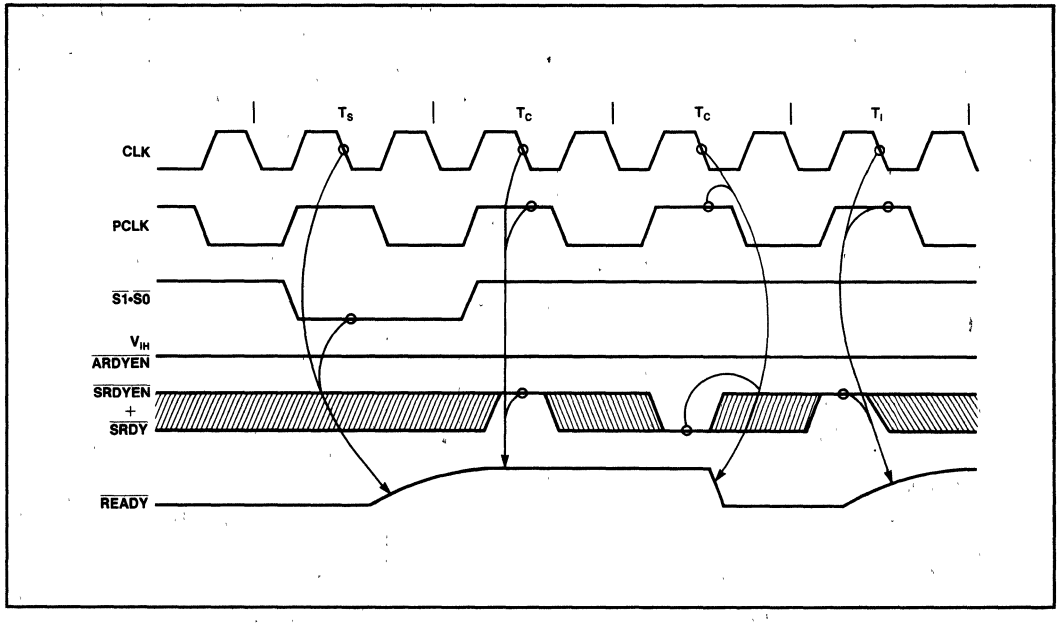


Figure 5. Synchronous Ready Operation

210453-003
AFN-00786B

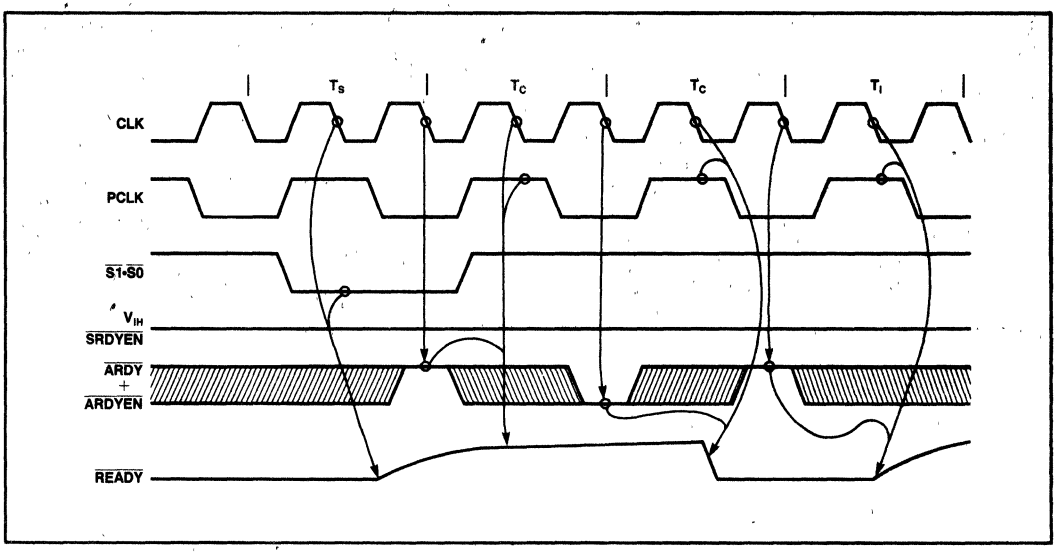


Figure 6. Asynchronous Ready Operation

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... 0°C to 70°C

Storage Temperature ..... -65°C to +150°C

All Output and Supply Voltages ..... -0.5V to +7V

All Input Voltages ..... -1.0V to +5.5V

Power Dissipation ..... 1 Watt

**Notice: Stresses above those listed under "Absolute Maxmum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V, ± 10%)

		6 MHz		8 MHz			
Sym	Parameter	-6 Min	-6 Max	Min	Max	Unit	Test Condition
V_{IL}	Input LOW Voltage		.8		.8	V	
V_{IH}	Input HIGH Voltage	2.0		2.0		V	
V_{IHR}	$\overline{RES}$ and EFI Input HIGH Voltage	2.6		2.6		V	
V_{HYS}	$\overline{RES}$ Input hysteresis	0.25		0.25		V	
V_{OL}	RESET, PCLK Output LOW Voltage		.45		.45	V	I_{OL} = 5mA
V_{OH}	RESET, PCLK Output HIGH Voltage	2.4		2.4		V	I_{OH} = -1mA
V_{OLR}	$\overline{READY}$, Output LOW Voltage		.45		.45	V	I_{OL} = 7mA
V_{OLC}	CLK Output LOW Voltage		.45		.45	V	I_{OL} = 5mA
V_{OHC}	CLK Output HIGH Voltage	4.0		4.0		V	I_{OH} = -800μA
V_C	Input Forward Clamp Voltage		-1.0		-1.0	V	I_C = -5mA
I_F	Forward Input Current		-.5		-.5	mA	V_F = .45V
I_R	Reverse Input Current		50		50	uA	V_R = V_{CC}
I_{CC}	Power Supply Current		145		145	mA	
C_I	Input Capacitance		10		10	pF	F_C = 1MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V, ± 10%)

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

		6 MHz		8 MHz			
Sym	Parameter	-6 Min	-6 Max	Min	Max	Unit	Test Condition
1	EFI to CLK Delay		35		30	ns	at 1.5V Note 1
2	EFI LOW Time	35		32		ns	at 0.8V Note 1
3	EFI HIGH Time	35		28		ns	at 2.0V Note 1
4	CLK Period	83	500	62	500	ns	
5	CLK LOW Time	20		15		ns	at 0.6V Note 1 Note 2
6	CLK HIGH Time	25		20		ns	at 3.8V Note 1 Note 2
7	CLK Rise Time		10		10	ns	1.0V to 3.5V Note 1
8	CLK Fall Time		10		10	ns	3.5V to 1.0V Note 1
9	Status Setup Time	28		22.5		ns	Note 1
10	Status Hold Time	0		0		ns	Note 1
11	$\overline{SRDY}$ or $\overline{SRDYEN}$ Setup Time	25		15		ns	Note 1
12	$\overline{SRDY}$ or $\overline{SRDYEN}$ Hold Time	0		0		ns	Note 1
13	$\overline{ARDY}$ or $\overline{ARDYEN}$ Setup Time	5		0		ns	Note 1 Note 3
14	$\overline{ARDY}$ or $\overline{ARDYEN}$ Hold Time	30		16		ns	Note 1 Note 3
15	$\overline{RES}$ Setup Time	25		16		ns	Note 1 Note 3
16	$\overline{RES}$ Hold Time	10		0		ns	Note 1 Note 3
17	$\overline{READY}$ Inactive Delay	5		5		ns	at 0.8V Note 4
18	$\overline{READY}$ Active Delay	0	33	0	24	ns	at 0.8V Note 4
19	PCLK Delay	0	45	0	40	ns	Note 5
20	RESET Delay	0	50	0	40	ns	Note 5
21	PCLK LOW Time	t4-20.		t4-13.		ns	Note 5 Note 6
22	PCLK HIGH Time	t4-20.		t4-13.		ns	Note 5 Note 6

NOTE 1: CLK loading: C1 = 150pF.

NOTE 2: With the internal crystal oscillator using recommended crystal and capacitive loading, or with the EFI input meeting specifications t2, and t3. Use a parallel-resonant, fundamental mode crystal. The recommended crystal loading for CLK frequencies of 8-16MHz are 25pF from pin X_1 to ground, and 15pF from pin X_2 to ground. These recommended values are ± 5pF and include all stray capacitance. Decouple V_{CC} and GND as close to the 82284 as possible.

NOTE 3: This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at specific CLK edge.

NOTE 4: $\overline{READY}$ loading: I_{OL} = 7mA, C_L = 150pF. In system application, use 910 ohm ±5% pullup resistor to meet 80286, 80286-6 and 80286-4 timing requirements.

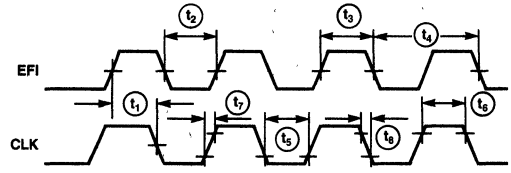
NOTE 5: PCLK and RESET loading: C_L = 75pF.

NOTE 6: t4 refers to any allowable CLK period.

210453-003
AFN-00786B

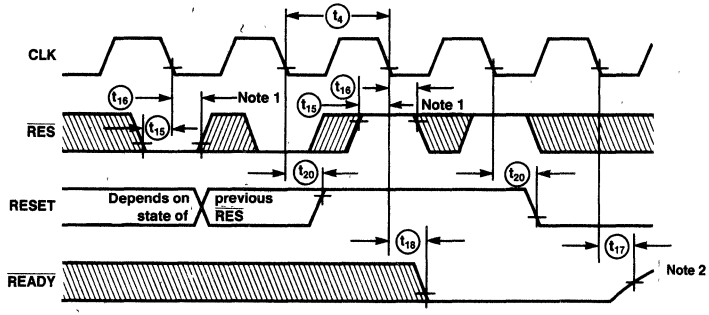
Waveforms

CLK as a Function of EFI



NOTE: The EFI input LOW and HIGH times as shown are required to guarentee the CLK LOW and HIGH times shown.

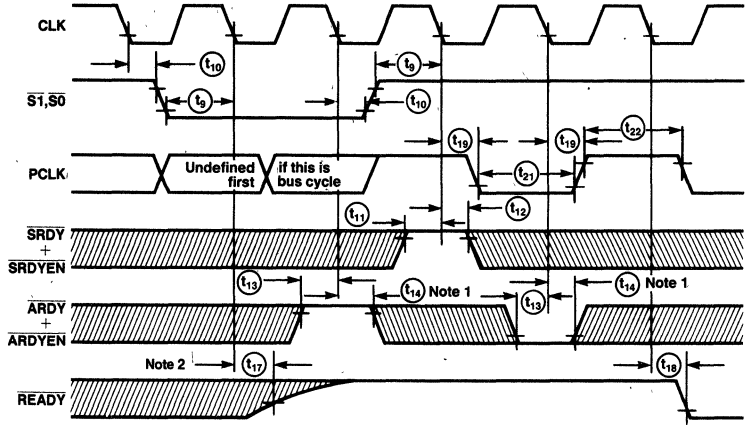
RESET and $\overline{\text{READY}}$ Timing as a Function of $\overline{\text{RES}}$ with $\overline{\text{S1}}$ and $\overline{\text{S0}}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

NOTE 2: Tie 910 ohm ±5% pullup resistor to the $\overline{\text{READY}}$ output

$\overline{\text{READY}}$ and PCLK Timing with $\overline{\text{RES}}$ HIGH



NOTE 1: This is an asynchronous input. The setup and hold times shown are required to guarantee the response shown.

NOTE 2: Tie 910 ohm ±5% pullup resistor to the $\overline{\text{READY}}$ output

82288
BUS CONTROLLER FOR iAPX 286 PROCESSORS

(82288, 82288-6)

- **Provides Commands and Control for Local and System Bus**
- **Offers Wide Flexibility in System Configurations**
- **Flexible Command Timing**
- **Optional Multibus* Compatible Timing**
- **Control Drivers with 16 ma I_{OL} and 3-State Command Drivers with 32 ma I_{OL}**
- **Single +5V Supply**

The Intel 82288 Bus Controller is a 20-pin HMOS component for use in iAPX 286 microsystems. The bus controller provides command and control outputs with flexible timing options. Separate command outputs are used for memory and I/O devices. The data bus is controlled with separate data enable and direction control signals.

Two modes of operation are possible via a strapping option: Multibus compatible bus cycles, and high speed bus cycles.

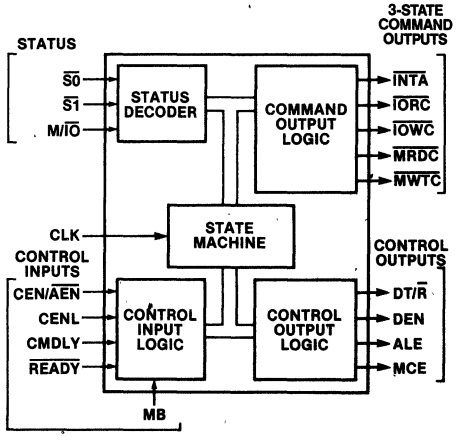


Figure 1. 82288 Block Diagram

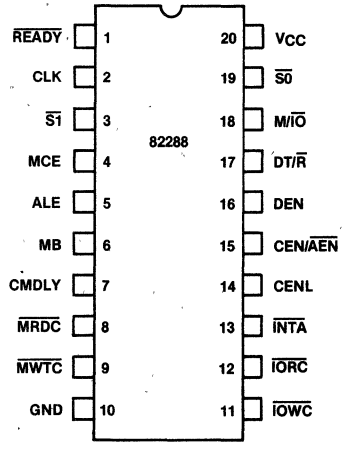


Figure 2. 88228 Pin Configuration

*Multibus is a patented bus of Intel.

NOVEMBER 1983
Order Number 210471-003

Table 1. Pin Description

The following pin function descriptions are for the 82288 bus controller.

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the basic timing control for the 82288 in an iAPX 286 microsystem. Its frequency is twice the internal processor clock frequency. The falling edge of this input signal establishes when inputs are sampled and command and control outputs change.
$\overline{S0}$, $\overline{S1}$	I	**Bus Cycle Status** starts a bus cycle and, along with M/$\overline{IO}$, defines the type of bus cycle. These inputs are active LOW. A bus cycle is started when either $\overline{S1}$ or $\overline{S0}$ is sampled LOW at the falling edge of CLK. These inputs have pullups sufficient to hold them HIGH when nothing drives them. Setup and hold times must be met for proper operation.

iAPX 286 Bus Cycle Status Definition

M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Type of Bus Cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	None; idle
1	0	0	Halt or shutdown
1	0	1	Memory read
1	1	0	Memory write
1	1	1	None; idle

M/$\overline{IO}$	I	**Memory or I/O Select** determines whether the current bus cycle is in the memory space or I/O space. When LOW, the current bus cycle is in the I/O space. Setup and hold times must be met for proper operation.
MB	I	**Multibus Mode Select** determines timing of the command and control outputs. When HIGH, the bus controller operates with Multibus-compatible timings. When LOW, the bus controller optimizes the command and control output timing for short bus cycles. The function of the CEN/$\overline{AEN}$ input pin is selected by this signal. This input is intended to be a strapping option and not dynamically changed. This input may be connected to V_{CC} or GND.
CENL	I	**Command Enable Latched** is a bus controller select signal which enables the bus controller to respond to the current bus cycle being initiated. CENL is an active HIGH input latched internally at the end of each T_S cycle. CENL is used to select the appropriate bus controller for each bus cycle in a system where the CPU has more than one bus it can use. This input may be connected to V_{CC} to select this 82288 for all transfers. No control inputs affect CENL. Setup and hold times must be met for proper operation.
CMDLY	I	**Command Delay** allows delaying the start of a command. CMDLY is an active HIGH input. If sampled HIGH, the command output is not activiated and CMDLY is again sampled at the next CLK cycle. When sampled LOW the selected command is enabled. If $\overline{READY}$ is detected LOW before the command output is activated, the 82288 will terminate the bus cycle, even if no command was issued. Setup and hold times must be satisfied for proper operation. This input may be connected to GND if no delays are required before starting a command. This input has no effect on 82288 control outputs.
$\overline{READY}$	I	**READY** indicates the end of the current bus cycle. $\overline{READY}$ is an active LOW input. Multibus mode requires at least one wait state to allow the command outputs to become active. $\overline{READY}$ must be LOW during reset, to force the 82288 into the idle state. Setup and hold times must be met for proper operation. The 82284 drives $\overline{READY}$ LOW during RESET.

210471-003
AFN-00787A

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
CEN/$\overline{AEN}$	I	**Command Enable/Address Enable** controls the command and DEN outputs of the bus controller. CEN/$\overline{AEN}$ inputs may be asynchronous to CLK. Setup and hold times are given to assure a guaranteed response to synchronous inputs. This input may be connected to V_{CC} or GND. When MB is HIGH this pin has the $\overline{AEN}$ function. $\overline{AEN}$ is an active LOW input which indicates that the CPU has been granted use of a shared bus and the bus controller command outputs may exit 3-state OFF and become inactive (HIGH). $\overline{AEN}$ HIGH indicates that the CPU does not have control of the shared bus and forces the command outputs into 3-state OFF and DEN inactive (LOW). $\overline{AEN}$ would normally be controlled by an 82289 bus arbiter which activates $\overline{AEN}$ when that arbiter owns the bus to which the bus controller is attached. When MB is LOW this pin has the CEN function. CEN is an unlatched active HIGH input which allows the bus controller to activate its command and DEN outputs. With MB LOW, CEN LOW forces the command and DEN outputs inactive but does not tristate them.
ALE	O	**Address Latch Enable** controls the address latches used to hold an address stable during a bus cycle. This control output is active HIGH. ALE will not be issued for the halt bus cycle and is not affected by any of the control inputs.
MCE	O	**Master Cascade Enable** signals that a cascade address from a master 8259A interrupt controller may be placed onto the CPU address bus for latching by the address latches under ALE control. The CPU's address bus may then be used to broadcast the cascade address to slave interrupt controllers so only one of them will respond to the interrupt acknowledge cycle. This control output is active HIGH. MCE is only active during interrupt acknowledge cycles and is not affected by any control input. Using MCE to enable cascade address drivers requires latches which save the cascade address on the falling edge of ALE.
DEN	O	**Data Enable** controls when data transceivers connected to the local data bus should be enabled. DEN is an active HIGH control output. DEN is delayed for write cycles in the Multibus mode.
DT/$\overline{R}$	O	**Data Transmit/Receive** establishes the direction of data flow to or from the local data bus. When HIGH, this control output indicates that a write bus cycle is being performed. A LOW indicates a read bus cycle. DEN is always inactive when DT/$\overline{R}$ changes states. This output is HIGH when no bus cycle is active. DT/$\overline{R}$ is not affected by any of the control inputs.
$\overline{IOWC}$	O	**I/O Write Command** instructs an I/O device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{IORC}$	O	**I/O Read Command** instructs an I/O device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{MWTC}$	O	**Memory Write Command** instructs a memory device to read the data on the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{MRDC}$	O	**Memory Read Command** instructs the memory device to place data onto the data bus. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
$\overline{INTA}$	O	**Interrupt Acknowledge** tells an interrupting device that its interrupt request is being acknowledged. This command output is active LOW. The MB and CMDLY inputs control when this output becomes active. $\overline{READY}$ controls when it becomes inactive.
V_{CC}		**System Power:** +5V power supply
GND		**System Ground:** 0 volts

210471-003
AFN-00787A

FUNCTIONAL DESCRIPTION

Introduction

The 82288 bus controller is used in iAPX 286 systems to provide address latch control, data transceiver control, and standard level-type command outputs. The command outputs are timed and have sufficient drive capabilities for large TTL buses and meet all IEEE-796 requirements for Multibus. A special Multibus mode is provided to statisfy all address/data setup and hold time requirements. Command timing may be tailored to special needs via a CMDLY input to determine the start of a command and $\overline{\text{READY}}$ to determine the end of a command.

Connection to multiple buses are supported with a latched enable input (CENL). An address decoder can determine which, if any, bus controller should be enabled for the bus cycle. This input is latched to allow an address decoder to take full advantage of the pipelined timing on the iAPX 286 local bus.

Buses shared by several bus controllers are supported. An $\overline{\text{AEN}}$ input prevents the bus controller from driving the shared bus command and data signals except when enabled by an external bus arbiter such as the 82289.

Separate DEN and DT/$\overline{\text{R}}$ outputs control the data transceivers for all buses. Bus contention is eliminated by disabling DEN before changing DT/$\overline{\text{R}}$. The DEN timing allows sufficient time for tristate bus drivers to enter 3-state OFF before enabling other drivers onto the same bus.

The term CPU refers to any iAPX 286 processor or iAPX 286 support component which may become an iAPX 286 local bus master and thereby drive the 82288 status inputs.

Processor Cycle Definition

Any CPU which drives the local bus uses an internal clock which is one half the frequency of the system clock (CLK) (see Figure 3). Knowledge of the phase of the local bus master internal clock is required for proper operation of the iAPX 286 local bus. The local bus master informs the bus controller of its internal clock phase when it asserts the status signals. Status signals are always asserted beginning in Phase 1 of the local bus master's internal clock.

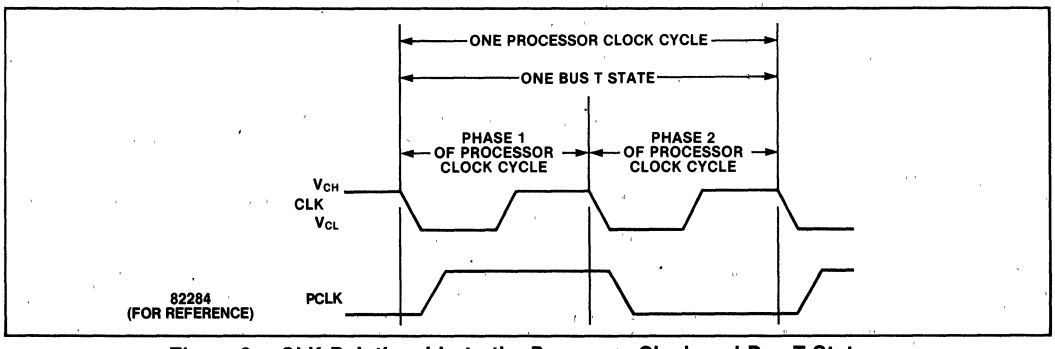


Figure 3. CLK Relationship to the Processor Clock and Bus T-States

210471-003
AFN-00787A

Bus State Definition

The 82288 bus controller has three bus states (see Figure 4): Idle (T_I) Status (T_S) and Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

The T_I bus state occurs when no bus cycle is currently active on the iAPX 286 local bus. This state may be repeated indefinitely. When control of the local bus is being passed between masters, the bus remains in the T_I state.

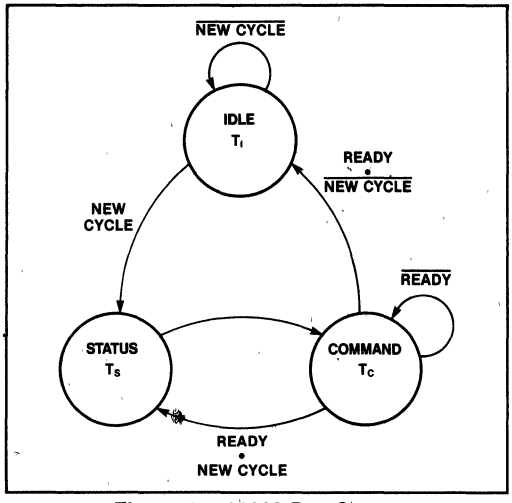


Figure 4. 82288 Bus States

Bus Cycle Definition

The $\overline{S1}$ and $\overline{S0}$ inputs signal the start of a bus cycle. When either input becomes LOW, a bus cycle is started. The T_S bus state is defined to be the two CLK cycles during which either $\overline{S1}$ or $\overline{S0}$ are active (see Figure 5). These inputs are sampled by the 82288 at every falling edge of CLK. When either $\overline{S1}$ or $\overline{S0}$ are sampled LOW, the next CLK cycle is considered the second phase of the internal CPU clock cycle.

The local bus enters the T_C bus state after the T_S state. The shortest bus cycle may have one T_S state and one T_C state. Longer bus cycles are formed by repeating T_C states. A repeated T_C bus state is called a wait state.

The $\overline{READY}$ input determines whether the current T_C bus state is to be repeated. The $\overline{READY}$ input has the same timing and effect for all bus cycles. $\overline{READY}$ is sampled at the end of each T_C bus state to see if it is active. If sampled HIGH, the T_C bus state is repeated. This is called inserting a wait state. The control and command outputs do not change during wait states.

When $\overline{READY}$ is sampled LOW, the current bus cycle is terminated. Note that the bus controller may enter the T_S bus state directly from T_C if the status lines are sampled active at the next falling edge of CLK.

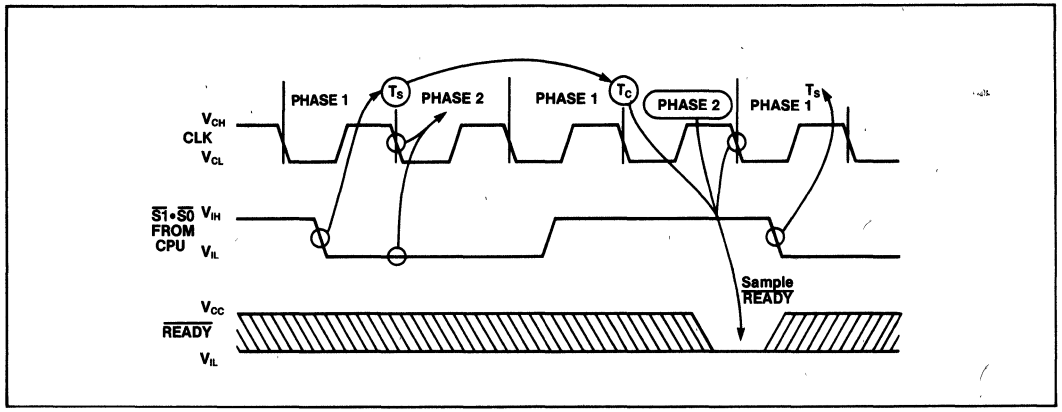


Figure 5. Bus Cycle Definition

210471-003
AFN-00787A

Table 2. Command and Control Outputs for Each Type of Bus Cycle

Type of Bus Cycle	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Command Activated	DT/$\overline{R}$ State	ALE, DEN Issued?	MCE Issued?
Interrupt Acknowledge	0	0	0	$\overline{INTA}$	LOW	YES	YES
I/O Read	0	0	1	$\overline{IORC}$	LOW	YES	NO
I/O Write	0	1	0	$\overline{IOWC}$	HIGH	YES	NO
None; idle	0	1	1	None	HIGH	NO	NO
Halt/Shutdown	1	0	0	None	HIGH	NO	NO
Memory Read	1	0	1	$\overline{MRDC}$	LOW	YES	NO
Memory Write	1	1	0	$\overline{MWTC}$	HIGH	YES	NO
None; idle	1	1	1	None	HIGH	NO	NO

Operating Modes

Two types of buses are supported by the 82288: Multibus and non-Multibus. When the MB input is strapped HIGH, Multibus timing is used. In Multibus mode, the 82288 delays command and data activation to meet IEEE-796 requirements on address to command active and write data to command active setup timing. Multibus mode requires at least one wait state in the bus cycle since the command outputs are delayed. The non-Multibus mode does not delay any outputs and does not require wait states. The MB input affects the timing of the command and DEN outputs.

Command and Control Outputs

The type of bus cycle performed by the local bus master is encoded in the M/$\overline{IO}$, $\overline{S1}$, and $\overline{S0}$ inputs. Different command and control outputs are activated depending on the type of bus cycle. Table 2 indicates the cycle decode done by the 82288 and the effect on command, DT/$\overline{R}$, ALE, DEN, and MCE outputs.

Bus cycles come in three forms: read, write, and halt. Read bus cycles include memory read, I/O read, and interrupt acknowledge. The timing of the associated read command outputs ($\overline{MRDC}$, $\overline{IORC}$, and $\overline{INTA}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical for all read bus cycles. Read cycles differ only in which command output is activated. The MCE control output is only asserted during interrupt acknowledge cycles.

Write bus cycles activate different control and command outputs with different timing than read bus cycles. Memory write and I/O write are write bus cycles whose timing for command outputs ($\overline{MWTC}$ and $\overline{IOWC}$), control outputs (ALE, DEN, DT/$\overline{R}$) and control inputs (CEN/$\overline{AEN}$, CENL, CMDLY, MB, and $\overline{READY}$) are identical. They differ only in which command output is activated.

Halt bus cycles are different because no command or control output is activated. All control inputs are ignored until the next bus cycle is started via $\overline{S1}$ and $\overline{S0}$.

210471-003
AFN-00787A

Figures 6-10 show the basic command and control output timing for read and write bus cycles. Halt bus cycles are not shown since they activate no outputs. The basic idle-read-idle and idle-write-idle bus cycles are shown. The signal label CMD represents the appropriate command output for the bus cycle. For Figures 6-10, the CMDLY input is connected to GND and CENL to V_{CC}. The effects of CENL and CMDLY are described later in the section on control inputs.

Figures 6, 7 and 8 show non-Multibus cycles. MB is connected to GND while CEN is connected to V_{CC}. Figure 6 shows a read cycle with no wait states while Figure 7 shows a write cycle with one wait state. The $\overline{READY}$ input is shown to illustrate how wait states are added.

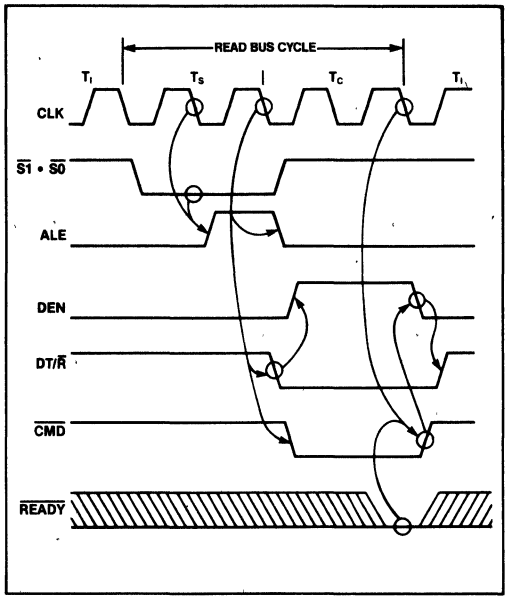


Figure 6. Idle-Read-Idle Bus Cycles with MB = 0

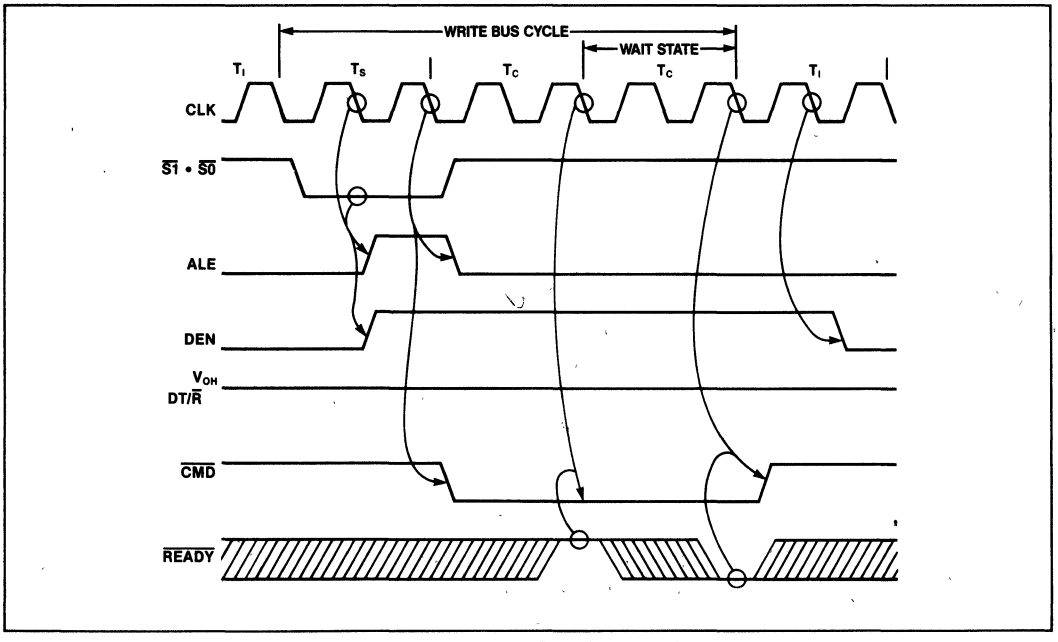


Figure 7. Idle-Write-Idle Bus Cycles with MB = 0

Bus cycles can occur back to back with no T_I bus states between T_C and T_S. Back to back cycles do not affect the timing of the command and control outputs. Command and control outputs always reach the states shown for the same clock edge (within T_S, T_C, or following bus state) of a bus cycle.

A special case in control timing occurs for back to back write cycles with MB = 0. In this case, DT/$\overline{R}$ and DEN remain HIGH between the bus cycles (see Figure 8). The command and ALE output timing does not change.

Figures 9 and 10 show a Multibus cycle with MB = 1. $\overline{AEN}$ and CMDLY are connected to GND. The effects of CMDLY and $\overline{AEN}$ are described later in the section on control inputs. Figure 9 shows a read cycle with one wait state and Figure 10 shows a write cycle with two wait states. The second wait state of the write cycle is shown only for example purposes and is not required. The $\overline{READY}$ input is shown to illustrate how wait states are added.

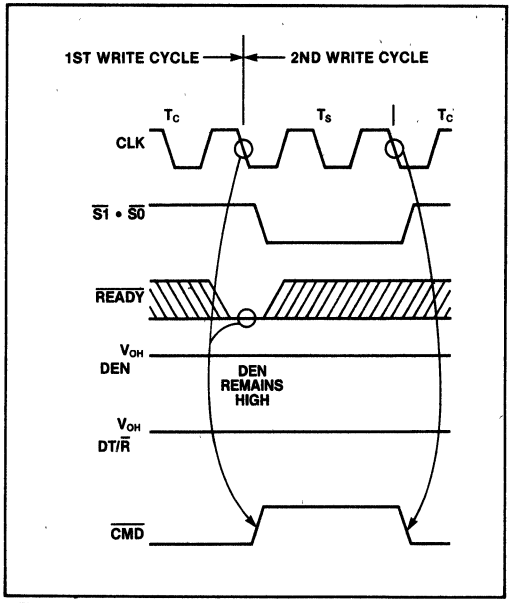


Figure 8. Write-Write Bus Cycles with MB = 0

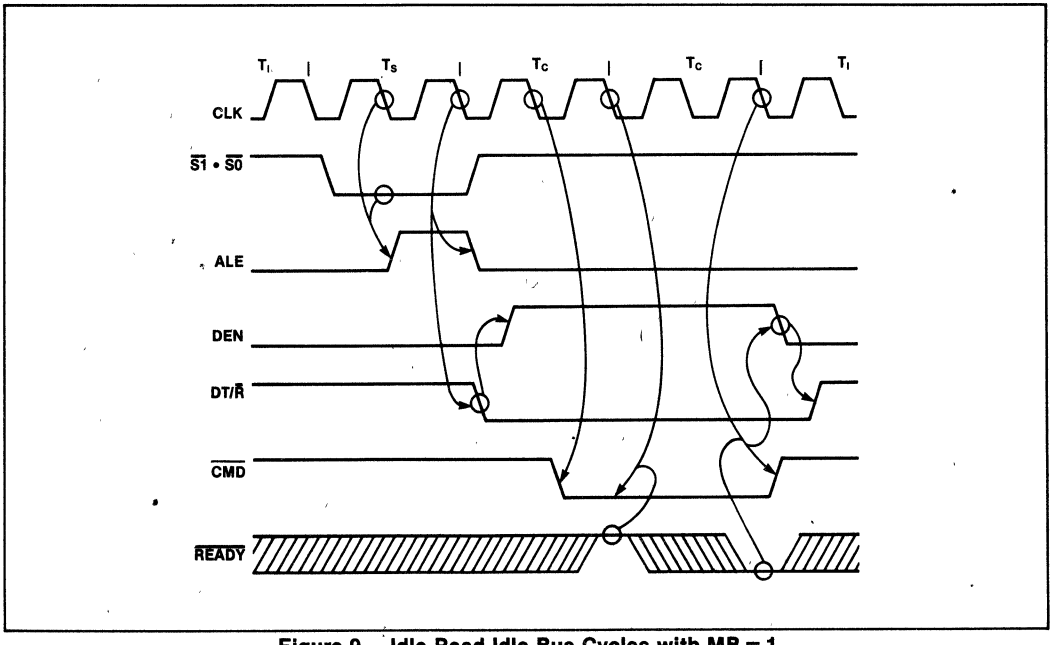


Figure 9. Idle-Read-Idle Bus Cycles with MB = 1

210471-003
AFN-00787A

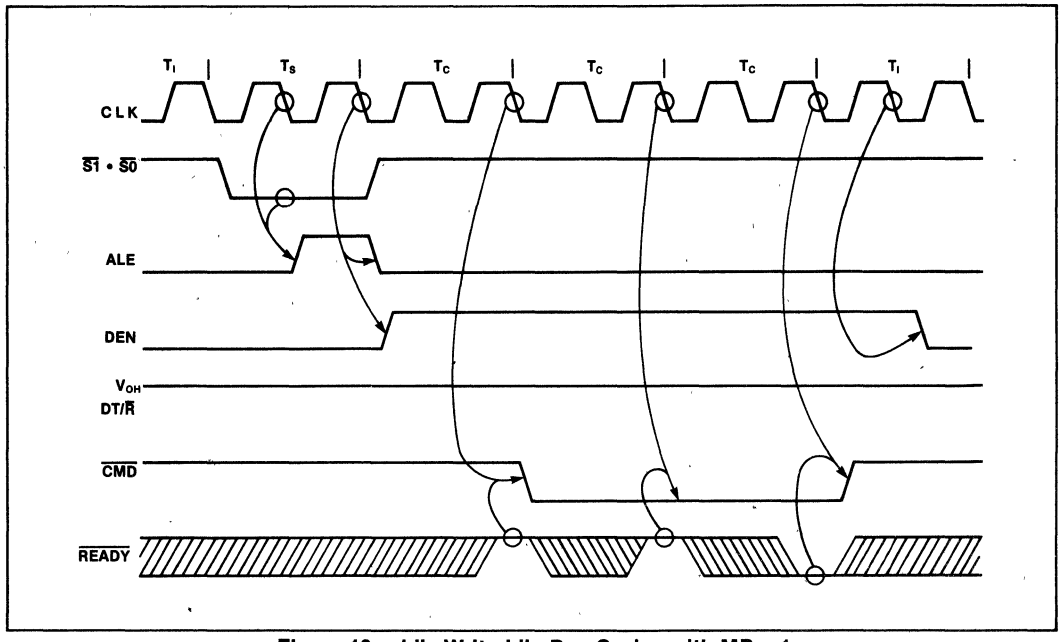


Figure 10. Idle-Write-Idle Bus Cycles with MB = 1

The MB control input affects the timing of the command and DEN outputs. These outputs are automatically delayed in Multibus mode to satisfy three requirements:

1) 50 ns minimum setup time for valid address before any command output becomes active.

2) 50 ns minimum setup time for valid write data before any write command output becomes active.

3) 65 ns maximum time from when any read command becomes inactive until the slave's read data drivers reach 3-state OFF.

Three signal transitions are delayed by MB = 1 as compared to MB = 0:

1) The HIGH to LOW transition of the read command outputs ($\overline{IORC}$, $\overline{MRDC}$, and $\overline{INTA}$) are delayed one CLK cycle.

2) The HIGH to LOW transition of the write command outputs ($\overline{IOWC}$ and $\overline{MWTC}$) are delayed two CLK cycles.

3) The LOW to HIGH transition of DEN for write cycles is delayed one CLK cycle.

Back to back bus cycles with MB = 1 do not change the timing of any of the command or control outputs. DEN always becomes inactive between bus cycles with MB = 1.

Except for a halt or shutdown bus cycle, ALE will be issued during the second half of T_S for any bus cycle. ALE becomes inactive at the end of the T_S to allow latching the address to keep it stable during the entire bus cycle. The address outputs may change during Phase 2 of any T_C bus state. ALE is not affected by any control input.

Figure 11 shows how MCE is timed during interrupt acknowledge (INTA) bus cycles. MCE is one CLK cycle longer than ALE to hold the cascade address from a master 8259A valid after the falling edge of ALE. With the exception of the MCE control output, an INTA bus cycle is identical in timing to a read bus cycle. MCE is not affected by any control input.

210471-003
AFN-00787A

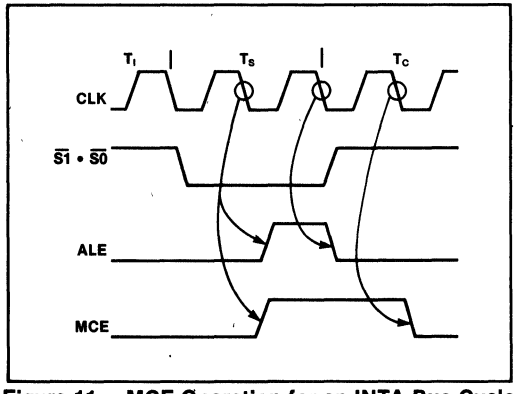


Figure 11. MCE Operation for an INTA Bus Cycle

Control Inputs

The control inputs can alter the basic timing of command outputs, allow interfacing to multiple buses, and share a bus between different masters. For many iAPX 286 systems, each CPU will have more than one bus which may be used to perform a bus cycle. Normally, a CPU will only have one bus controller active for each bus cycle. Some buses may be shared by more than one CPU (i.e. Multibus) requiring only one of them use the bus at a time.

Systems with multiple and shared buses use two control input signals of the 82288 bus controller, CENL and $\overline{AEN}$ (see Figure 12). CENL enables the bus controller to control the current bus cycle. The $\overline{AEN}$ input prevents a bus controller from driving its command outputs. $\overline{AEN}$ HIGH means that another bus controller may be driving the shared bus.

In Figure 12, two buses are shown: a local bus and a Multibus. Only one bus is used for each CPU bus cycle. The CENL inputs of the bus controllers select which bus controller is to perform the bus cycle. An address decoder determines which bus to use for each bus cycle. The 82288 connected to the shared Multibus must be selected by CENL and be given access to the Multibus by $\overline{AEN}$ before it will begin a Multibus operation.

CENL must be sampled HIGH at the end of the T_S bus state (see waveforms) to enable the bus controller to activate its command and control outputs. If sampled LOW the commands and DEN will not go active and DT/$\overline{R}$ will remain HIGH. The bus controller will ignore the CMDLY, CEN, and $\overline{READY}$ inputs until another bus cycle is started via $\overline{S1}$ and $\overline{S0}$. Since an address decoder is commonly used to identify which bus is required for each bus cycle, CENL is latched to avoid the need for latching its input.

The CENL input can affect the DEN control output. When MB = 0, DEN normally becomes active during Phase 2 of T_S in write bus cycles. This transition occurs before CENL is sampled. If CENL is sampled LOW, the DEN output will be forced LOW during T_C as shown in the timing waveforms.

When MB = 1, CEN/$\overline{AEN}$ becomes $\overline{AEN}$. $\overline{AEN}$ controls when the bus controller command outputs enter and exit 3-state OFF. $\overline{AEN}$ is intended to be driven by a bus arbiter, like the 82289, which assures only one bus controller is driving the shared bus at any time. When $\overline{AEN}$ makes a LOW to HIGH transition, the command outputs immediately enter 3-state OFF and DEN is forced inactive. An inactive DEN should force the local data transceivers connected to the shared data bus into 3-state OFF (see Figure 12). The LOW to HIGH transition of $\overline{AEN}$ should only occur during T_I or T_S bus states.

The HIGH to LOW transition of $\overline{AEN}$ signals that the bus controller may now drive the shared bus command signals. Since a bus cycle may be active or be in the process of starting, $\overline{AEN}$ can become active during any T-state. $\overline{AEN}$ LOW immediately allows DEN to go to the appropriate state. Three CLK edges later, the command outputs will go active (see timing waveforms). The Multibus requires this delay for the address and data to be valid on the bus before the commands become active.

When MB = 0, CEN/$\overline{AEN}$ becomes CEN. CEN is an asynchronous input which immediately affects the command and DEN outputs. When CEN makes a HIGH to LOW transition, the commands

and DEN are immediately forced inactive. When CEN makes a LOW to HIGH transition, the commands and DEN outputs immediately go to the appropriate state (see timing waveforms). READY must still become active to terminate a bus cycle if CEN remains LOW for a selected bus controller (CENL was latched HIGH).

Some memory or I/O systems may require more address or write data setup time to command active than provided by the basic command output timing. To provide flexible command timing, the CMDLY input can delay the activation of command outputs. The CMDLY input must be sampled LOW to activate the command outputs. CMDLY does not affect the control outputs ALE, MCE, DEN, and DT/R.

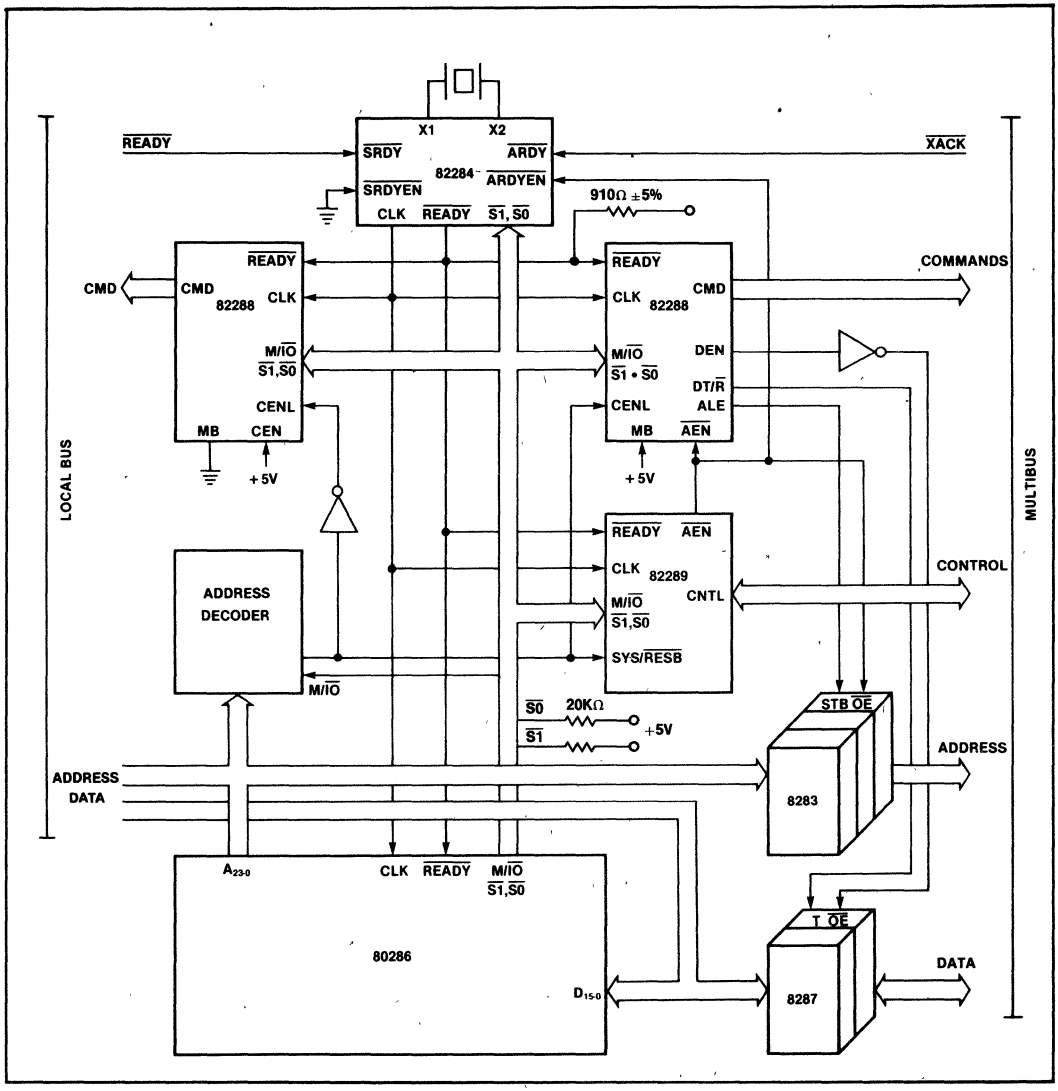


Figure 12. System Use of AEN and CENL

210471-003
AFN-00787A

CMDLY is first sampled on the falling edge of the CLK ending T_S. If sampled HIGH, the command output is not activated, and CMDLY is again sampled on the next falling edge of CLK. Once sampled LOW, the proper command output becomes active immediately if MB = 0. If MB = 1, the proper command goes active no earlier than shown in Figures 9 and 10.

$\overline{READY}$ can terminate a bus cycle before CMDLY allows a command to be issued. In this case no commands are issued and the bus controller will deactivate DEN and DT/$\overline{R}$ in the same manner as if a command had been issued.

Waveforms Discussion

The waveforms show the timing relationships of inputs and outputs and do not show all possible transitions of all signals in all modes. Instead, all signal timing relationships are shown via the general cases. Special cases are shown when needed. The waveforms provide some functional descriptions of the 82288; however, most functional descriptions are provided in Figures 5 through 11.

To find the timing specification for a signal transition in a particular mode, first look for a special case in the waveforms. If no special case applies, then use a timing specification for the same or related function in another mode.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to GND . . . . . −0.5V to +7V
Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V, ±10%)

		6 MHz		8 MHz			
Symbol	Parameter	−6 Min.	−6 Max.	Min.	Max.	Units	Test Conditions
V_{IL}	Input LOW Voltage	−.5	.8	−.5	.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC} + .5	2.0	V_{CC} + .5	V	
V_{ILC}	CLK Input LOW Voltage	−.5	.6	− 5	.6	V	
V_{IHC}	CLK Input HIGH Voltage	3.8	V_{CC} +.5	3.8	V_{CC} +.5	V	
V_{OL}	Output LOW Voltage						
	Command Outputs		.45		.45	V	I_{OL} = 32mA Note 1
	Control Outputs		.45		.45	V	I_{OL} = 16mA Note 2
V_{OH}	Output HIGH Voltage						
	Command Outputs	2.4		2.4		V	I_{OH} = −5mA Note 1
	Control Outputs	2.4		2 4		V	I_{OH} = −1mA Note 2
I_F	Input Current ($\overline{S0}$ and $\overline{S1}$ inputs)		− 5		− 5	mA	Vf = .45V
I_{IL}	Input Leakage current (all other inputs)		±10		±10	µA	OV ≤ V_{IN} ≤ V_{CC}
I_{LO}	Output Leakage Current		±10		± 10	µA	.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	Power Supply Current		100		100	mA	
C_{CLK}	CLK Input Capacitance		12		12	pF	F_C = 1 MHz
C_I	Input Capacitance		10		10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance		20		20	pF	F_C = 1 MHz

NOTE: 1. Command Outputs are $\overline{INTA}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{MRDC}$, $\overline{MWRC}$.
2. Control Outputs are DT/$\overline{R}$, DEN, ALE and MCE.

A.C. CHARACTERISTICS

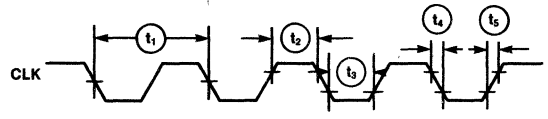
(T_A = 0°C to 70°C, V_{CC} = 5V, ±10%)
AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in data sheet waveforms, unless otherwise noted.

		6 MHz		8 MHz			
Sym	Parameter	-6 Min.	-6 Max.	Min.	Max.	Unit	Test Condition
1	CLK Period	83	250	62	250	ns	
2	CLK HIGH Time	25	235	20	235	ns	at 3.8V
3	CLK LOW Time	20	225	15	230	ns	at 0.6V
4	CLK Rise Time		10		10	ns	1.0V to 3.5V
5	CLK Fall Time		10		10	ns	3.5V to 1.0V
6	M/$\overline{IO}$ and Status Setup Time	28		22 5		ns	
7	M/$\overline{IO}$ and Status Hold Time	0		0		ns	
8	CENL Setup Time	30		20		ns	
9	CENL Hold Time	0		0		ns	
10	$\overline{READY}$ Setup Time	50		38.5		ns	
11	$\overline{READY}$ Hold Time	35		25		ns	
12	CMDLY Setup Time	25		20		ns	
13	CMDLY Hold Time	0		0		ns	
14	$\overline{AEN}$ Setup Time	30		25		ns	Note 3
15	$\overline{AEN}$ Hold Time	0		0		ns	Note 3
16	ALE, MCE Active Delay from CLK	3	25	3	15	ns	Note 4
17	ALE, MCE Inactive Delay from CLK		35		20	ns	Note 4
18	DEN (Write) Inactive from CENL		35		35	ns	Note 4
19	DT/$\overline{R}$ LOW from CLK		40		20	ns	Note 4
20	DEN (Read) Active from DT/$\overline{R}$	10	50	10	40	ns	Note 4
21	DEN (Read) Inactive Dly from CLK	3	40	3	35	ns	Note 4
22	DT/$\overline{R}$ HIGH from DEN Inactive	5	45	10	40	ns	Note 4
23	DEN (Write) Active Delay from CLK		35		30	ns	Note 4
24	DEN (Write) Inactive Dly from CLK	3	35	3	30	ns	Note 4
25	DEN Inactive from CEN		40		25	ns	Note 4
26	DEN Active from CEN		35		25	ns	Note 4
27	DT/$\overline{R}$ HIGH from CLK (when CEN = LOW)		50		50	ns	Note 4
28	DEN Active from $\overline{AEN}$		35		30	ns	Note 4
29	$\overline{CMD}$ Active Delay from CLK	3	40	3	20	ns	Note 5
30	$\overline{CMD}$ Inactive Delay from CLK	3	30	3	20	ns	Note 5
31	$\overline{CMD}$ Inactive from CEN		35		25	ns	Note 5
32	$\overline{CMD}$ Active from CEN		45		25	ns	Note 5
33	$\overline{CMD}$ Inactive Enable from $\overline{AEN}$		40		40	ns	Note 5
34	$\overline{CMD}$ Float Delay from $\overline{AEN}$		40		40	ns	Note 6

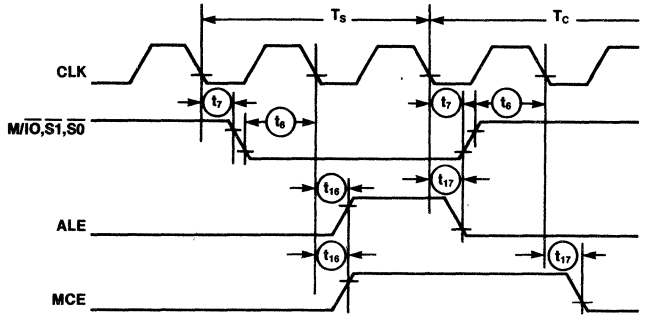
NOTE: 3. $\overline{AEN}$ and MB are asynchronous inputs. This specification is for testing purposes only, to assure recognition at a specific CLK edge.
4. Control output load: Cl = 150pF.
5. Command output load: Cl = 300pF.
6. Float condition occurs when output current is less then I_{LO} in magnitude

WAVEFORMS

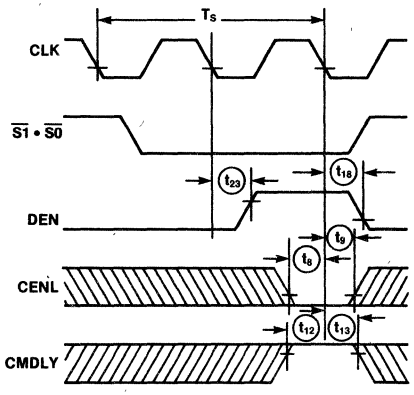
CLK CHARACTERISTICS



STATUS, ALE, MCE, CHARACTERISTICS



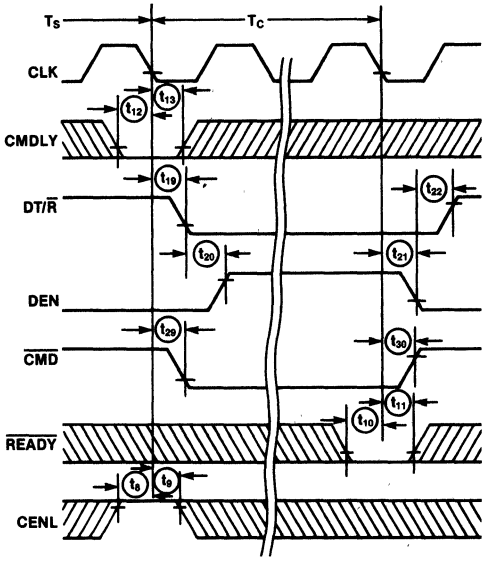
CENL, CMDLY, DEN CHARACTERISTICS WITH MB = 0 AND CEN = 1 DURING WRITE CYCLE



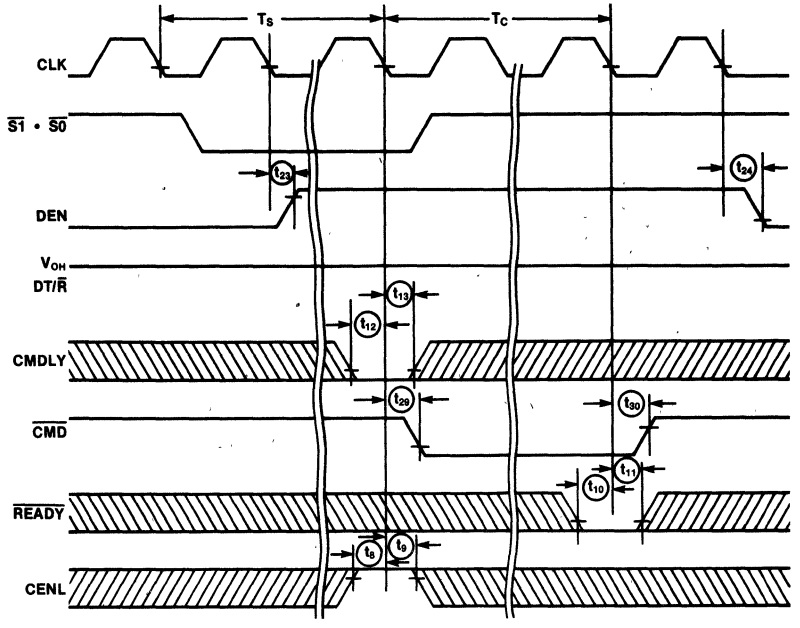
210471-003
AFN-00787A

WAVEFORMS (Continued)

READ CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1



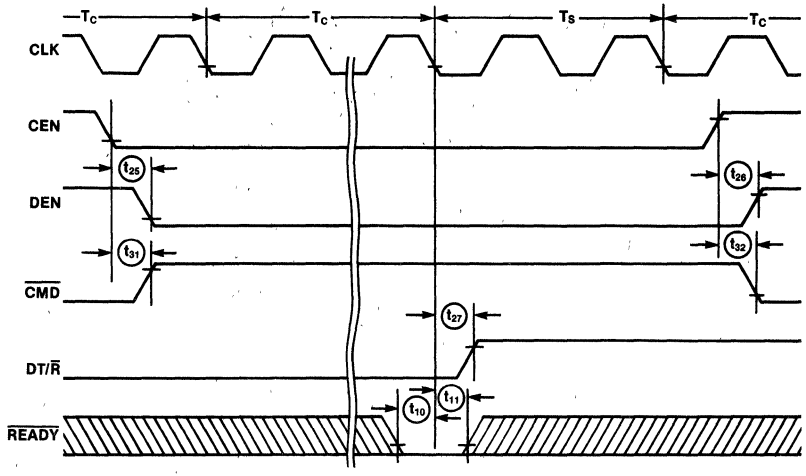
WRITE CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1



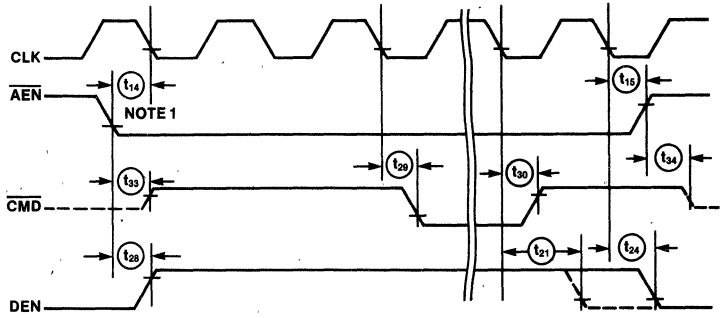
210471-003
AFN-00787A

WAVEFORMS (Continued)

CEN CHARACTERISTICS WITH MB = 0



AEN CHARACTERISTICS WITH MB = 1



NOTE 1: AEN is an asynchronous input. AEN setup and hold time is specified to guarantee the response shown in the waveforms.

iAPX 432 Micromainframe™

5

Microprocessors Section

iAPX 43201/iAPX 43202
FAULT TOLERANT GENERAL DATA PROCESSOR

- **Range of Performance**
  - **— Adding Processors Increases Performance**
  - **— No Software Changes Required**
- **Large Address Space:**
  - **— 16 Megabytes Physical Memory**
  - **— 1 Terabyte Virtual Memory**
- **Capability-Based Addressing for Maximum Dependability**
  - **— Most Software Faults Trapped Before Damage Occurs**
  - **— Debugging Time Reduced**
  - **— Leads to Highly-Reliable, Robust Systems**
- **Memory Management Support On-Chip**
- **Symmetrical Instruction Set for all 8-, 16-, and 32-bit Data Types**
- **IEEE Standard 32-, 64-, and 80-Bit Floating-Point Operations**
- **Master/Checker Pairs Detect Hardware Errors Automatically**
- **Quad Modular Redundancy Ensures Immediate Recovery From Hardware Faults**

The iAPX 432 Micromainframe is a 32-bit multiprocessor specifically designed for those critical applications which demand absolute software reliability or hardware fault tolerance. At the heart of the system is the iAPX 432 General Data Processor (GDP) consisting of two VLSI components, the iAPX 43201 and iAPX 43202. Together with the other members of the iAPX 432 component family (i.e., the 43203 Interface Processor, the 43204 Bus Interface Unit, and the 43205 Memory Control Unit) the GDP can be used to build a fault-tolerant computer system that sustain any single-point failure and yet continue to operate correctly, without interruption.

Intel 432 systems offer a range of performance: by adding or removing GDPs, the throughput of a 432-based product can be increased to support more users or reduced to save hardware cost with no change to software. Thus, a family of end products with differing levels of performance can be developed from identical hardware modules using the same software.

The iAPX 43201 and iAPX 43202 are fabricated with Intel's highly reliable +5-Volt, depletion load, N-channel, silicon gate HMOS technology and each is housed in a 68-pin JEDEC Type A package. See Figures 1 and 2.

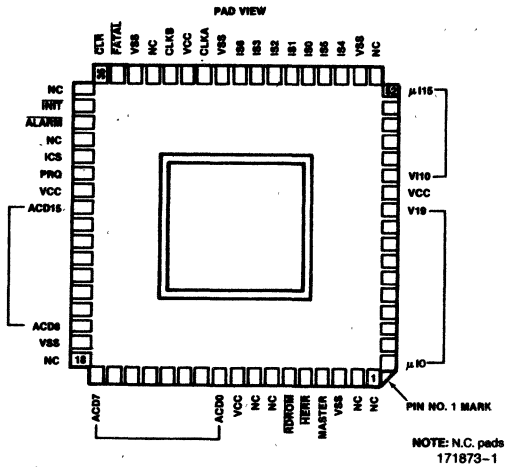


Figure 1. 43201 Pin Assignment, Instruction Decoder/Microinstruction Sequencer

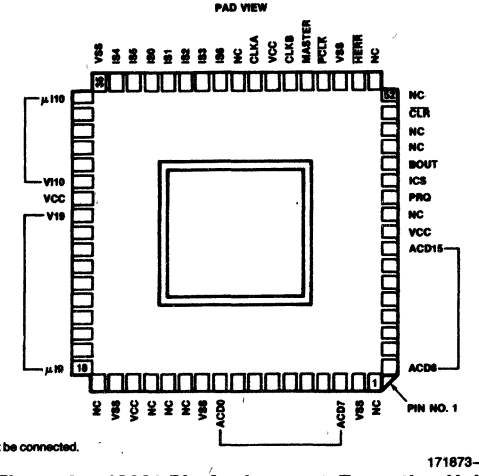


Figure 2. 43202 Pin Assignment, Execution Unit

November 1983
Order Number: 171873-002

Table 1 summarizes all signal groups, signal names and their active states, and whether or not they are monitored by the Hardware Error Detection circuitry.

Table 1. Pin Description

Symbol	Type	Name and Function
Processor Packet Bus Group		
ACD_{15} ACD_0	I/O	**Address/Control/Data Lines:** These 16 bidirectional signals carry physical memory addresses, control information (access length and type), and data to and from the memory bus. When the GDP is in checker mode, the ACD pins are monitored by the hardware error detection logic and are in the high impedance state.
PRQ	O	**Processor Packet Bus Request:** is issued to gain access to the bus. Normally low, the PRQ pin is brought high during the same cycle as when the first double-byte of address information appears on the ACD pins. PRQ remains high for only one cycle during the access, unless an address development fault occurs. In that case, the GDP leaves PRQ high for a second cycle to indicate it has detected an addressing or segments rights fault in completing the address generation. PRQ is checked by the hardware detection logic and remains in a high impedance mode when the GDP is in checker mode.
ICS	I	**Interconnect Status:** carries information on errors, data synchronization, and interprocessor communication. The interpretation of this signal depends on the current cycle of the bus transaction. See page 39 for a complete description.
B_{OUT}	O	**Enable Buffers for Output:** controls the direction of external buffers, if any are used. When B_{OUT} is asserted, it indicates that the buffers must be directed to carry information outbound from the GDP.
System Group		
V_{CC}	I	**Power:** These four pins supply 5-volt power to the GDP, and all must be connected; the pins are not connected together within the GDP.
V_{SS}	I	**Ground:** These five pins provide the ground reference for the GDP, and all must be connected; the pins are not connected together within the GDP.
$\overline{ALARM}$	I	**$\overline{Alarm}$:** monitors the condition of an unusual, system-wide condition such as power failure. Alarm is sampled on the rising edge of CLK_A.
$\overline{FATAL}$	O	**$\overline{Fatal}$:** is asserted by the GDP under microcode control when the GDP is unable to continue due to various error or fault conditions. Once $\overline{FATAL}$ is asserted, it can only be reset by the assertion of $\overline{INIT}$.
$\overline{PCLK}$	I	**Processor Clock:** The assertion of $\overline{PCLK}$ for one cycle causes the system timer within the GDP to decrement. Assertion of $\overline{PCLK}$ for two or more cycles causes the system timer to be reset. $\overline{PCLK}$ must remain unasserted for at least 10 clock cycles before being asserted again. The GDP samples $\overline{PCLK}$ on the rising edge of CLK_A.

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function
System Group (Continued)		
$\overline{CLR}$	I	**Clear:** Assertion of $\overline{CLR}$ results in a microprogram trap causing the GDP to immediately terminate any bus transactions or internal operations in progress. The GDP resets to a known state, asserts $\overline{FATAL}$, and awaits an IPC for initialization. The IPC is not serviced for at least four clock cycles following the assertion of $\overline{CLR}$. Response to $\overline{CLR}$ is disabled by the first $\overline{CLR}$ assertion and is reenabled when the GDP receives the first IPC (or $\overline{INIT}$ assertion). The GDP samples $\overline{CLR}$ on the rising edge of CLK_A.
$\overline{INIT}$	I	**Initialize:** Assertion of $\overline{INIT}$ resets the internal state of the GDP and starts execution of the initialization microcode. $\overline{INIT}$ must be asserted for a minimum of 10 clock cycles. After the $\overline{INIT}$ pin returns to its nonasserted state, the GDP initializes all of its internal registers and windows, and waits for a local IPC. $\overline{INIT}$ is sampled on the rising edge of CLK_A.
MASTER	I	**Master:** This signal determines whether the 43202 is to function as a master or a checker. In master mode, the 43202 functions normally and drives all of its outputs. In checker mode, ACD_{15}-ACD_0 and PRQ enter a high impedance state and B_{OUT} is unconditionally low. A 43202, whether master or checker, monitors the ACD_{15}-ACD_0 and PRQ lines and compares the data on them to its internally generated result, signaling disagreement on the $\overline{HERR}$ line. For normal operation, MASTER should be left unconnected or tied high.
$\overline{HERR}$	O	**Hardware Error:** This signal is asserted by the GDP to indicate disagreement between data appearing on the ACD_{15}-ACD_0 and PRQ pins and the internally generated result of the GDP. $\overline{HERR}$ is valid during CLK_A and can normally be asserted by a GDP and every clock cycle, except immediately following $\overline{CLR}$. $\overline{HERR}$ requires an external 2.2 kΩ nominal pullup resistor.
CLK_A	I	**Clock A:** is a square-wave clock which must operate continuously to preserve the operating state of the GDP.
CLK_B	I	**Clock B:** is a square-wave clock which operates at the same frequency as CLK_A, but lags it by 90 degrees. CLK_B must operate continuously to preserve the operating state of the GDP.
Intra-GDP Group		
IS_6–IS_0	N/A	**Interchip Status Lines:** carry microprogram information from the 43202 back to the 43201. The lines are not checked by the hardware detection logic.
μI_0–μI_{15}	N/A	**Microinstruction Bus Lines:** carry microinstructions from the 43201 to the 43202. They are not checked by hardware detection logic.
$\overline{RDROM}$	I	**Read ROM:** This signal is used to read the microprogram from the 43201 ROM. If $\overline{RDROM}$ is held low when $\overline{INIT}$ goes high, the 43201 enters a diagnostic mode, and the microinstruction sequencer steps through the microprogram ROM, sequentially displaying (but not executing) the microinstructions on the μI_{15}-μI_0 lines. While $\overline{RDROM}$ is useful for testing, it should be tied to V_{CC} for normal operation.

FUNCTIONAL DESCRIPTION

Introduction

The iAPX 432 Micromainframe is a 32-bit multiprocessor especially designed for those critical applications which demand absolute software reliability or hardware fault tolerance. By developing the 432, Intel has broken with three decades of tradition that have defined how computers operate and redrawn the line separating functions of hardware and software. Many operations that 432 processors perform automatically would be done by the operating system in conventional machines. The development of the 432 was driven by two major objectives: to reduce the cost of software over the life cycle of the product, and to develop a computer with unprecedented reliability. From any perspective, the 432 is an uncommon machine.

Similar to many mainframe computers, processing in the 432 is divided between a central system, which handles data processing, and one or more peripheral subsystems, which transfer data to and from I/O devices. There are two types of processors, General Data Processors (GDPs) and Interface Processors (IPs), and two types of support components, Bus Interface Units (BIUs) and Memory Control Units (MCUs). Together, these VLSI components can be used to build a fault-tolerant computer system that is able to sustain any single-point failure of a component or bus, and yet continue to operate correctly, without program interruption and without software intervention.

This concern for reliability in the 432's design is not limited to automatic recovery from hardware faults, but extends to software as well. The 432 processors can detect hundreds of different software fault conditions from an attempt to divide by zero or execute data, to complex faults involving several independent processes. While most computers do not detect these faults at all, the 432 is able to trap and identify most faults before serious damage can occur. As a consequence, 432-based systems are easier to debug, and systems shipped to end-users will prove substantially more reliable.

Another important advantage of the 432 is the ability to tailor the throughput of the system to meet the price/performance requirements of each application or end-user. A family of products, for example, can be developed using the same hardware modules, simply adding or removing boards as the application requires. The end-user, for example, could buy an entry-level system with only two processors. The system would run more slowly than the high-end system, but it would also cost much less. Later, when the user's needs grew, additional General Data Processors could be installed to handle the heavier load on the system. No need to change software; all the programs that the user had developed would still be compatible. In fact, if at any time, one of the processors failed, the user could remove it, and the remaining processors in the system would continue to execute programs correctly while a replacement was sought.

This unprecedented flexibility is possible only because the 432 takes a unique approach to architecture, one closely tied to the structure of programs. The processors are no longer passive entities, responding only to software, but they can execute many functions automatically to keep the system working efficiently and reliably.

ARCHITECTURE

This section describes the architecture of the iAPX 432; that is, the machine-level programmer's view of the computer. As a rule, only compiler writers actually deal with the 432 at this level; however, many application programmers—who typically code in Ada—will find this discussion valuable for getting a "feel" for the operation of the underlying machine. Bear in mind that this discussion does not cover all programming facilities and some of the concepts have been simplified for the sake of clarity and space; a complete description of the architecture can be found in the **iAPX 432 General Data Processor Architecture Reference Manual.**

Since the 432 is a multiprocessor system, with specialized types of processors optimized for different kinds of work, the architecture of the GDP is different from the Interface Processor. At the same time, these diverse processors (and their associated software) must cooperate with each other to accomplish the overall task of the system. Therefore, the designs of the GDP and IP share an architectural foundation, the 432 common base architecture. The overall arrangement is illustrated in Figure 3.

Common Base Architecture

The common base architecture of the 432 is the glue that binds multiple processors, of the same and different types, together in a coherent system. Therefore, the common base architecture defines "global" facilities used by all processors, software, and support components (i.e., iAPX 43204 Bus Interface Unit and iAPX 43205 Memory Control Unit). These facilities include addressing, protection, object management, communication, timekeeping, and exception processing.

Objects

Memory protection systems can be considered in terms of boxes into which information can be locked and keys which can open the boxes. The 432's boxes and keys correspond to the logical units found in many high-level programming languages (e.g., Ada, Pascal); in other words, programming units and protection units match. Boxes are variable in size, each box containing one **object**; the object may be an array, a record, a list element, the text of a procedure—in fact, any logical entity which should be uniquely identified and protected.

Keys in the 432 are called access descriptors and are manifested in high level languages as pointers (instances of access variables in Ada). Whenever storage for a new object is allocated, whether by the linker or program statements, the processor automatically boxes the object (in its own segment), manufactures a unique key (access descriptor), and returns the key to the creating procedure. A brief introduction to objects can be found in Table 2.

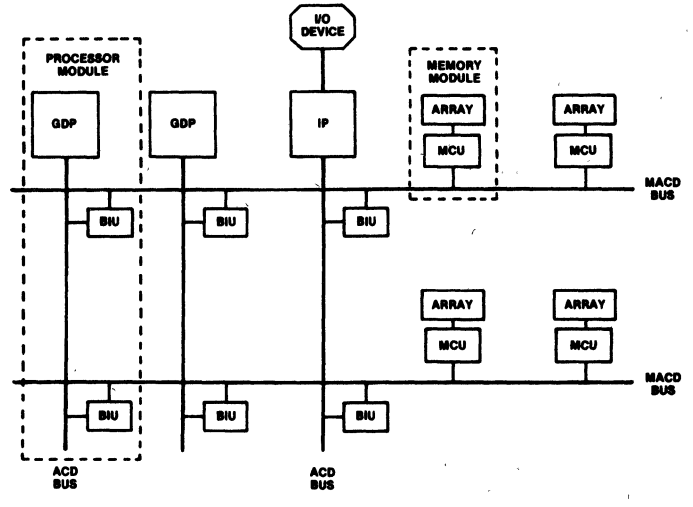


Figure 3. A typical 432 system with two memory buses, two General Data Processors, and a single peripheral subsystem for I/O

Table 2. Introducing Objects

1. **All information in an iAPX 432 system is contained in objects.**
   Even the instruction pointer, status flags, and other information used by the GDP are contained in objects.

2. **Each object can have two parts, a data part and an access part.**
   The data part can contain any information except accesses. Data in the data part can be added, assigned, manipulated as bit fields, or used for any purpose other than accessing an object. The access part can contain only **access descriptors** (ADs). ADs are used for referencing objects and can only be modified in carefully controlled ways.

3. **Objects can be created with different lengths.**
   An object can have from 0 to 65,536 bytes in its data part, and from 0 to 16,384 ADs in its access part. Any reference to an object is automatically checked to ensure that it falls within the bounds of the object.

Table 2. Introducing Objects (Continued)

4. **Each object has a fixed type.**
   The type of an object is determined when the object is created. An object's type can be used to define the operations allowed on the object. Software can define new object types at run-time.

5. **Objects can be local to a program or subprogram call.**
   Each object is created at a particular **level** that specifies whether the object is global or limited in scope to a particular program or subprogram activation.

6. **Objects can only be read or written via access descriptors.**
   To access data in an object, you must specify an AD that references the object and also specify the offset within the object's data part to the field accessed.

7. **A procedure call can only access those objects it has ADs for.**
   Each activation of a program or procedure is itself represented by a **context object**. The instructions executed by the context can only access those objects for which the context has ADs or can obtain ADs.

8. **Access descriptors can provide restricted access to objects.**
   Each AD specifies several **rights** bits, including read rights and write rights. To read from an object requires read rights set on the AD used; to write to an object requires write rights on the AD used. Different module activations can have ADs for the same object, but with different rights.

Storage

Address Spaces There are several distinct address spaces in a iAPX 432 system. Each peripheral subsystem, for example, has a local memory space, and typically, a local I/O space. A portion of each peripheral subsystem memory space is mapped by an Interface Processor into the central system. Processors (and DMA controllers) in a peripheral subsystem can gain access to central system data by reading and writing these local mapped address spaces.

The central system is divided into two 16-Megabyte physical address spaces, the **memory space** and the **interconnect space**. For the most part, the interconnect space consists of hardware registers used by the Bus Interface and Memory Control Units to maintain fault-tolerant systems. The MCU, for example, logs the number of memory errors it has detected and corrected in a register that processors are able to read by addressing the interconnect space.

GDPs use the instruction MOVE FROM INTERCONNECT SPACE to read these registers and MOVE TO INTERCONNECT SPACE to write to them. Peripheral subsystem software can gain access to the interconnect space through Window 1. All other 432 instructions and commands reference the memory space.

Logical addressing To operate on a data item, a GDP instruction (or IP command) presents a logical address as shown in Figure 4. For a GDP, a data item is an integer, character, or real number. In the case of an IP data transfer, such data items are simply bytes or double bytes. The logical address of a data item consists of an access descriptor and a displacement (offset) into that object. The several ways of specifying these components, particularly the displacement, give rise to the GDP's addressing modes, which are described in a later section.

Physical Translation GDPs translate logical addresses into 24-bit physical address; programs have no way to generate physical addresses directly. As shown in Figure 5, the essence of the translation is finding the physical base address of the target object and adding the displacement component to it.

As you'll recall, associated with each 432 procedure (or function) is an object reference list (set of keys) called access descriptors; this array of access descriptors defines the objects that are currently addressable by the procedure. The access descriptor in turn selects an entry in an object table. This entry, called an object descriptor, contains the base address of the target object.

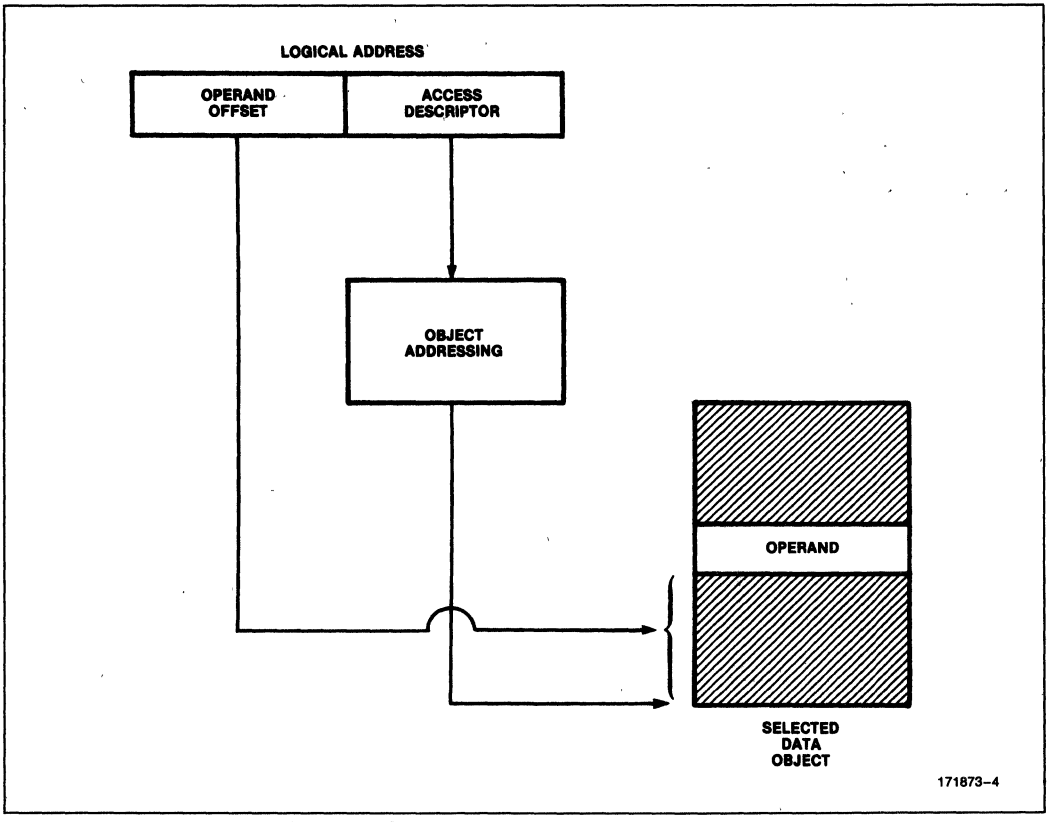


Figure 4. Simplified View of Logical Addressing

The focal point of the translation procedure is the object table. (Naturally, only the operating system's memory manager has a reference to the object table itself.) Every object in a system is represented by a single object descriptor, which contains its base address and other descriptive information. Conversely, there may be many access descriptors for a single object scattered among the access descriptor lists of all programs that have access to that object. Moving an object (to compact physical memory, for example), only requires updating its object descriptor, regardless of how many programs hold references to it.

Note that to improve performance, 432 processors automatically maintain two groups of physical addresses on chip. The addresses of frequently used system objects, such as the object table, are always immediately available. In addition, the processors cache exploits the tendency of most programs to cluster their references to a few objects at a time.

Virtual Memory Support Three fields in object descriptors aid a 432 executive's virtual memory manager. The **allocated** bit indicates whether real memory is associated with the object. When the virtual memory manager swaps an object out to external storage, it clears this bit first. The hardware checks the allocated bit during address translation; if it is clear, the hardware faults, and control passes to the memory manager. The memory manager can then swap the object back into physical memory, set the allocated bit, and return control to the instruction that faulted.

The **accessed** bit is set by the hardware when an object is referenced by an instruction. By periodically checking and clearing this bit in all object descriptors, the virtual memory manager can gain insight into the frequency with which each object is being used. This information can then be used to select objects to be swapped out, for example, on a least-recently-used basis.

The **altered** bit is set by hardware when an object is updated. If a virtual memory manager decides to swap an object that has not been altered, and it knows that a copy already exists in external storage, it can save time by discarding the memory-resident copy without swapping it. Both the accessed and altered bits are also cached on-chip to keep object table references to a minimum.

Protection and Access Control The services provided by modern operating systems typically take the form of procedures that can be called by user processes. Since most computers do not provide procedure-level protection as a matter of course, a special arrangement is necessary to protect operating system procedures from their callers (an OS procedure runs in the same process as its caller).

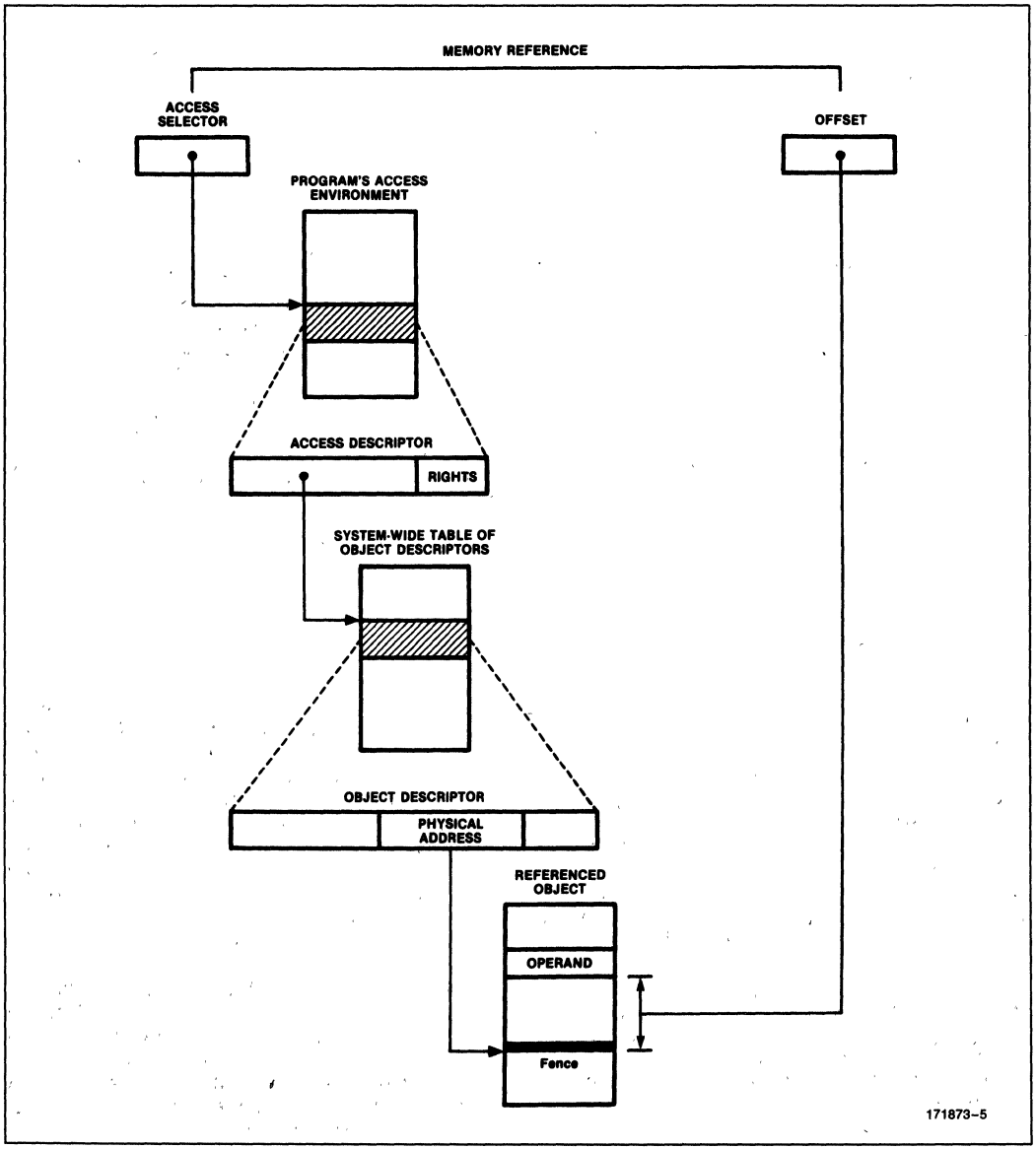


Figure 5. Two-Level Address Mapping

The usual approach is to confer **privilege** upon OS procedures while denying it to user processes. In effect, privilege is a second lock that operating system procedures can apply to boxes containing their information. If a computer has one privilege level (e.g., privileged supervisor, unprivileged user) then one key opens any privilege lock. A 4-level privilege scheme (e.g. kernel, executive, supervisor, user) provides more locks, one for each level. Each privileged key, however, not only opens all locks of the same level, but is effectively a master key for all lower privilege levels.

Privilege thus provides an asymmetric form of protection: information stored at a given level is protected from lower-level procedures only, not from procedures at the same or higher levels; in fact a procedure at the lowest level runs on an unprotected machine.

The 432's protection system, in contrast, is symmetric: every procedure—part of the operating system or not—is equally protected from every other because every procedure has access only to those objects it absolutely needs. The 432's run-time protection system is based on limiting the distribution of information and controlling the manner in which accessible information is referenced. Each object is protected as a unit, and each executing procedure has a unique access environment that specifies exactly which objects it may use, and in what manner it may use them. At the same time, the system is flexible enough to permit straightforward sharing of information between cooperating processes.

Limited Access Environments At any given point in time, a procedure's access list constitutes its current access environment. Since addresses can only be constructed from the access descriptors on this list, it is impossible for a procedure to gain access to any object for which it does not hold an access descriptor.

A procedure creates an initial access environment (called a context) for a procedure as part of its execution of the CALL instruction that invokes the procedure. (Actually, the entire context is not created each time a procedure is called, rather, to increase performance, each process is given a set of inactive contexts after compilation. When a procedure is called the context is "filled in" with the parameters for that activation.) This initial environment includes references for all objects that could be ascertained at compile-time: objects containing constants and statically allocated variables, and any procedures called by this procedure. In addition, the processor creates a reference to any parameters passed by the caller, to free storage from which new objects may be allocated, and to an operand stack for use in evaluating expressions in this activation.

As the procedure executes, it may modify its own access environment. For example, it may create a new object, it may receive an object reference from another process, or it may delete an object reference that is no longer needed. Notice that while the procedure may create new data at will, the only way for it to obtain access to data outside its initial environment is through the explicit cooperation of another process (i.e., through the interprocess communication facility called ports).

By entering a new context for each procedure activation, the 432 provides inherent support for reentrant and recursive programming. The complete change in access environments caused by the procedure invocation ensures that the called procedure inherits no part of its caller's environment, except for the parameters that have been passed to it. When the called procedure returns, the processor switches back to the caller's environment.

Integral Sharing Sharing is the other side of protection. To be able to share pieces of memory among processes in a controlled way is an important capability for a computer. Popular procedures and programs (especially editors and compilers) are often executed by many processes simultaneously; sharing a single copy of the code among all processes not only saves memory, but reduces virtual memory I/O traffic and disk space requirements.

The ability to share data is as important as sharing instructions; some applications, especially control-oriented ones, are most easily implemented as multiple processes sharing key data structures. The alternatives to memory sharing (maintaining separate copies for each process or sharing a file instead of memory) are often too wasteful, too complicated, or too slow to be practical in applications where the processes must have fast access to up-to-date information.

Unfortunately, most computer addressing schemes favor either protection or sharing. To protect processes from each other, many computers map them into separate address spaces; sharing an item then requires construction of a bridge between the two address spaces, which—when done at all—provides an awkward and incomplete facility. Other computers run processes in the same address space, enabling sharing but sacrificing protection (i.e., processes share everything).

The 432 has a single address space; every object in this space is potentially sharable by every process in the system. Protection is obtained by restricting the object each procedure can address to the subset it "needs to know" to perform its function; at any time the subset is defined by the procedures list of access descriptors.

An object is shared if procedures in more than one process has references for it (conversely, an object is private if there is only a single reference for it). The references can be provided by the compiler or can be obtained at run-time. (Note that interprocess sharing is a requirement of Ada and other languages that support concurrent programming.)

Typing Given that a procedure has access to an object (has a reference for it), the protection system insures the basic integrity of that object. That is, instruction references to the object are checked to make certain that they make sense for that object. By **typing** each object, the 432 hardware can positively determine what an object is; this, in turn, implies which operations (i.e., instructions) are valid for the object and which are not. In order to maintain good performance, however, the individual data items (e.g., characters, integers, etc.) within an object are typed and typed-checked at compile-time, not run-time.

Types of Objects There are five classes of objects that are used in a 432 system:

- System objects
- Generic objects
- Dynamic-type objects
- Refinements of any of these objects
- Interconnect objects

System objects have specific uses and formats recognized by 432 processors, and serve as the backbone of the architecture and the operating system. The 432 provides a very high-level of support for operating system functions because the processors are able to recognize and manipulate system objects. Processors operate on system objects in two ways: by executing high-level instructions on behalf of the operating system, and by executing certain functions on their own initiative, without software intervention.

The structure of a high-level program is reflected in the type and function of system objects. An Ada package, for example, is represented within the 432 as a **domain object.** The domain object contains a list of procedures, functions, and data associated with the single package; and like a package, is divided into **private** and **public** parts. Procedures in the public part of a domain correspond to procedures which are declared in the specification part of a package, and which are accessible to outside users. The private part of the domain, in contrast, corresponds to those data structures, functions, and procedures within the implementation section (body). No outside user can operate on, or directly call, these structures.

Instruction and data objects are good examples of the protection that an object-based architecture provides. Instructions contained in an instruction object cannot be manipulated as data, and likewise, the entries in a data object cannot be fetched by a processor for execution. Further, both the instruction and data objects are hidden from users in the sense that access to them is provided only through a procedure call.

The type managers for system objects are combinations of hardware and operating system software, so application programmers have no need to understand the internal organizations (representations) of these objects. A few of them, however, are cornerstones of the 432 architecture and it is worthwhile to examine them from a functional point of view. **Instruction objects,** from which processors fetch a procedure's machine instructions, have already been mentioned. Later in this section **storage resource** and **port** objects will be discussed.

Processor and **process objects** are used, naturally, to manage processors and processes. There is one processor object for each GDP and IP in the system. At initialization time, each processor obtains a reference for its own processor object, which it holds on-chip indefinitely. There is also one process object for each process (task) in the system. A processor object contains an access descriptor for a process object. Dispatching a new process to run on a processor amounts to changing this access descriptor. The entire set of system objects is described briefly in Table 3 and illustrated in Figure 6.

Generic objects An object whose type is **generic** is one that has no special significance to the hardware; its system type is effectively "none." Creating generic objects is faster than creating other types of objects and requires no special privilege. In languages such as Ada or Pascal, executing a NEW operation creates a record referenced by a pointer. In the 432, the operation creates a generic object and an access descriptor to reference it.

Dynamic-Typed Objects OEM-developed software added to a computer needs protection for exactly the same reasons that the operating system does. Coexisting with unknown programs that have been written by fallible (and sometimes malicious) end-users, the OEM software must keep running correctly and must prevent the disclosure or alteration of sensitive information belonging to either the OEM or the end-user. The principal weakness of conventional protection systems is that the hardware vendor monopolizes the protection facilities of the machine, leaving only interprocess protection for the OEM or end-user. In the 432, full protection can be extended to every new facility, whether added by Intel, the OEM, a software vendor, or the end-user.

Table 3. iAPX 432 System Objects

Instruction Object
contains GDP instructions; the GDP will fetch instructions only from instruction objects.

Domain
represents a program module (package) and references subprograms (instruction objects) and data objects in the module.

Context
represents a program or subprogram activation (call) and defines the access environment of the call, i.e., the set of objects that the activation can reference.

Type Definition Object (TDO)
represents a software-defined object type, and can contain attributes of the type (e.g., the type name).

Type Control Object (TCO)
represents type-specific privileges, such as the right to create objects of a particular type or to gain access to objects of a particular type.

Object Table
contains the object descriptors used in object addressing and memory management.

Storage Resource Object (SRO)
represents a free storage pool used to create new objects; references an object table that will contain the new object's descriptor, a physical storage object from which the new segment will be allocated, and a storage claim object that limits allocation from this SRO.

Physical Storage Object (PSO)
specifies free storage blocks in memory.

Storage Claim Object (SCO)
limits the number of bytes that can be allocated from a set of SROs that reference this SCO.

Process
represents a program or subprogram activation that can execute concurrently (in parallel) with other processes.

Port
provides communication between concurrent activities. A port includes a queue of messages sent to the port but not yet received, and a queue of blocked activities waiting to receive messages (at an empty port) or to send messages (at a port with a full message queue).

Carrier
represents an activity in communication with other concurrent activities via ports. Carriers carry messages to and from ports.

Processor Object
contains attributes and state information for an iAPX 432 processor (e.g., a GDP). Because programs in an iAPX 432 system can only manipulate information in objects, all information about a processor that must be visible to software must be contained in an object.

Processor Communication Object
used by the iAPX 432 interprocessor communication mechanism to transfer messages between processors.

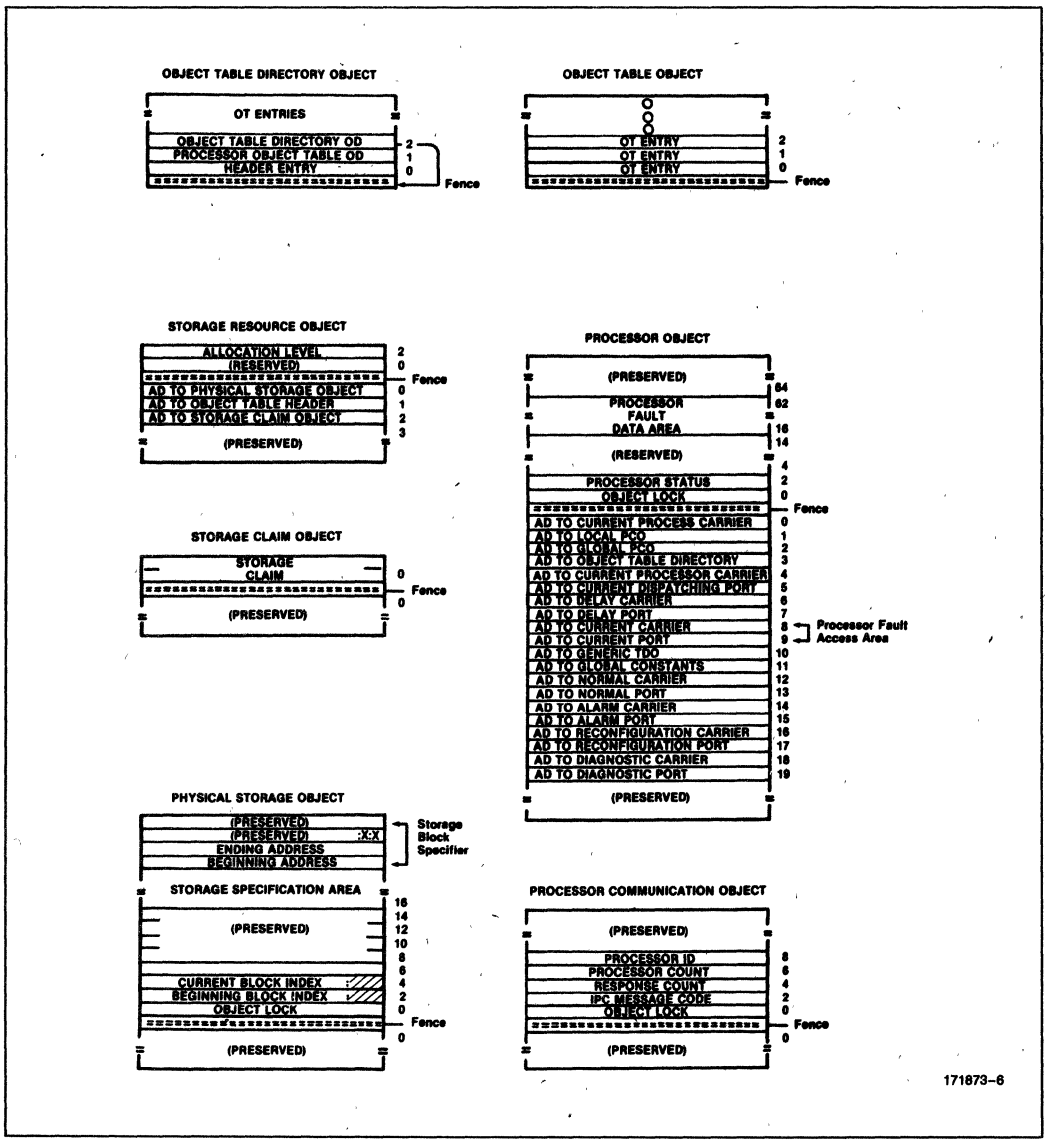


Figure 6. 432 System Objects (Reserved areas are used by processors; preserved areas are available to system software)

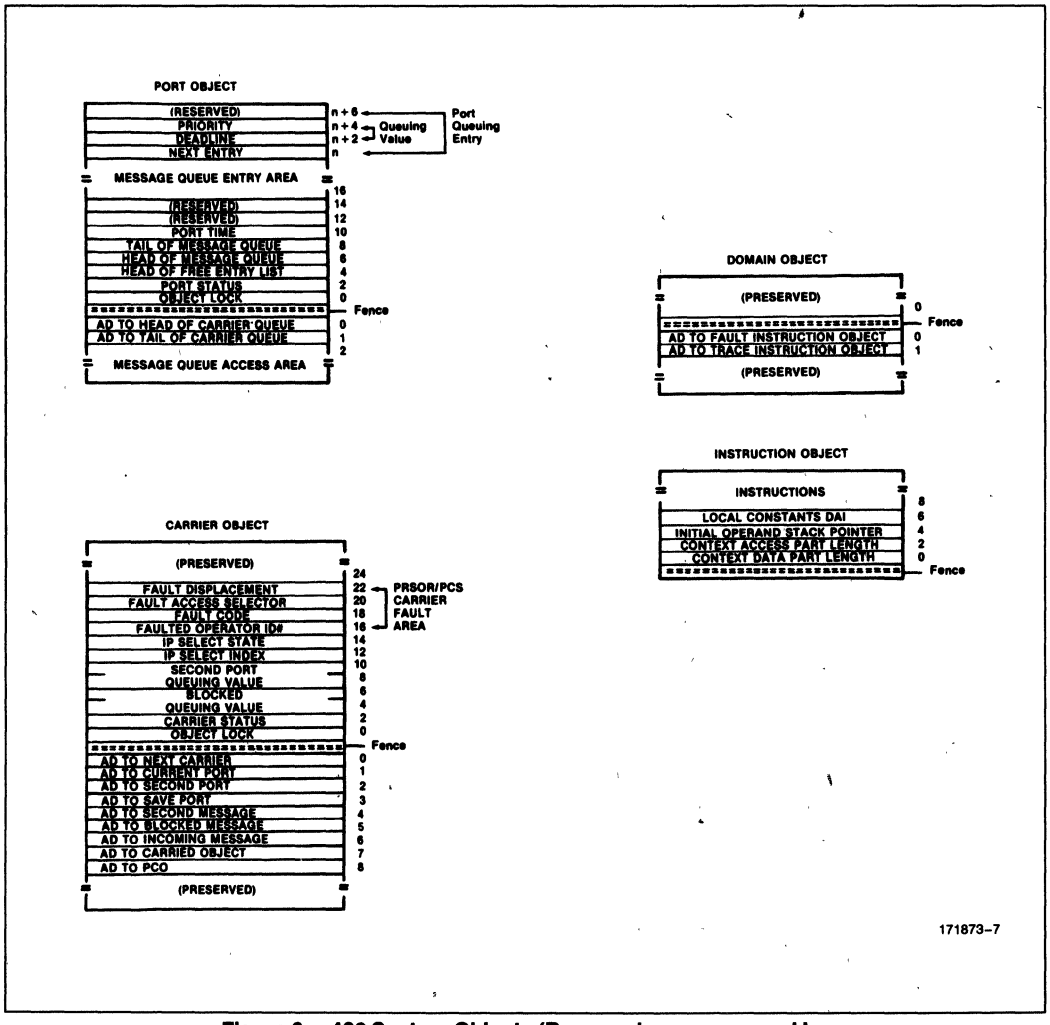


Figure 6. 432 System Objects (Reserved areas are used by processors; preserved areas are available to system software) (Continued)

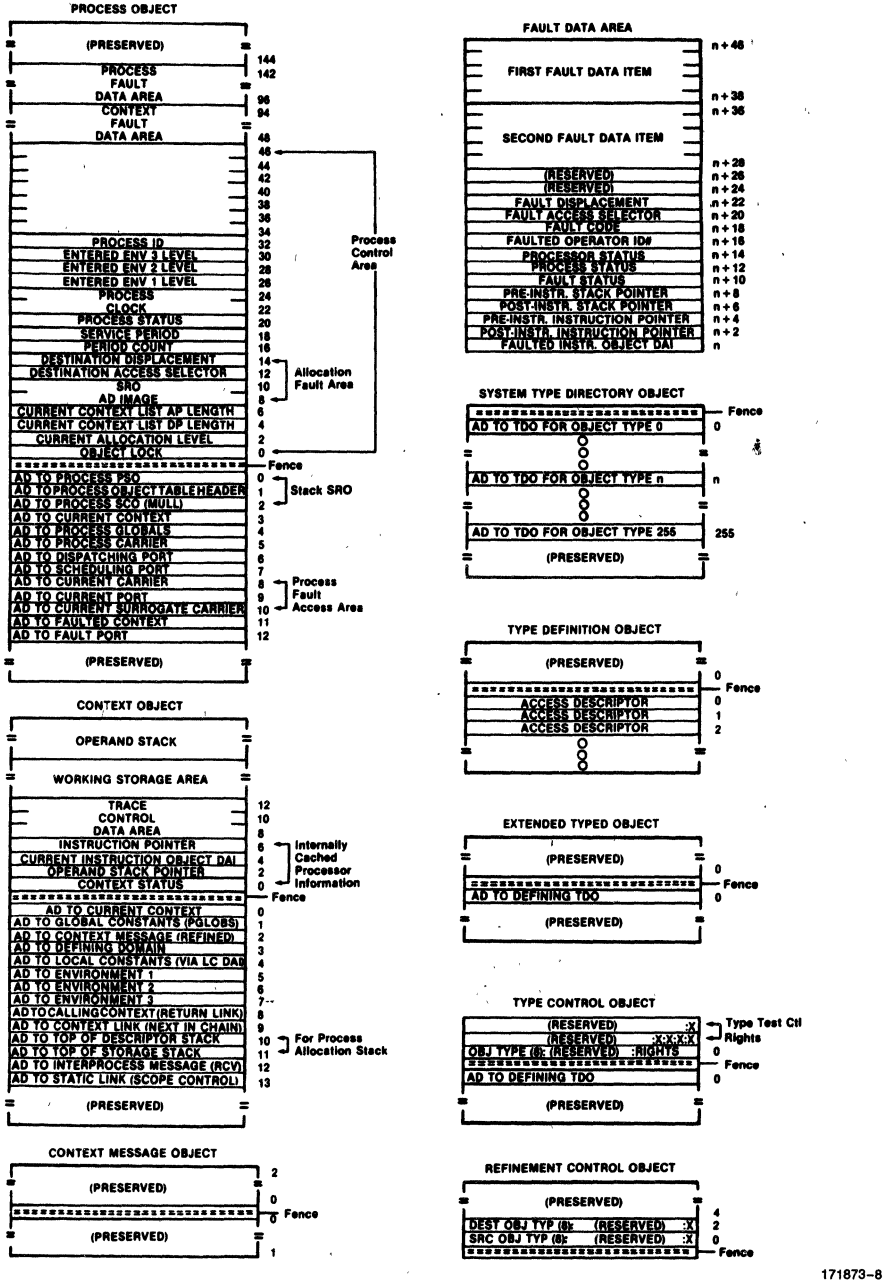


Figure 6. 432 System Objects (Reserved areas are used by processors; preserved areas are available to system software) (Continued)

If base typing protects the 432's addressing mechanism, and system typing protects the 432's system objects, dynamic-typing can be viewed as protecting users from one another. In essence, dynamic-typing enables a programmer to define a specific type of object including the operations that are valid for it, and then have the hardware enforce the definition, just as it would for any of the system objects.

When a type manager creates a dynamic-type object for a user, it returns an access descriptor for that object. When the user subsequently wishes to manipulate the object, it passes the access descriptor as a parameter in a call to one of the type manager's operations. Dynamic-typing prevents the user from manipulating the object's representation directly; the user holds a reference for the object, but the object is effectively "sealed" from access. Only the type manager can "unseal" the object; any other attempt to operate on the object's contents is aborted by the hardware.

Refinement Occasionally, it is convenient to define a new object that is a subset of an existing object. A "personnel record," for example, might contain "public" information such as name and department, and "private" information, such as salary. A process with access to the complete record may want to send only the public part to another process. It may do so by creating what is called a **refinement** of the object (see Figure 7).

Interconnect Objects Groups of hardware registers within the interconnect space are represented as Interconnect Objects, and can be accessed by the GDP only through the MOVE TO INTERCONNECT and MOVE FROM INTERCONNECT operators (Note: an Interface Processor gains access by opening Window 1 onto an Interconnect Object). Interconnect objects have no access descriptors.

Rights and Bounds Each access descriptor contains a set of rights which further defines how the holder of the reference may use the referenced object (see Figure 8). These **base** and **system** rights are set when the holder is given the reference. Base rights define whether the object may be read-only, write-only, either or neither. A user with a reference to a system object, for example, will be unable to read or write the object. System rights are system-type-specific; for example, the system rights for a port object indicate whether the holder may send a message to the port, or receive a message from the port.

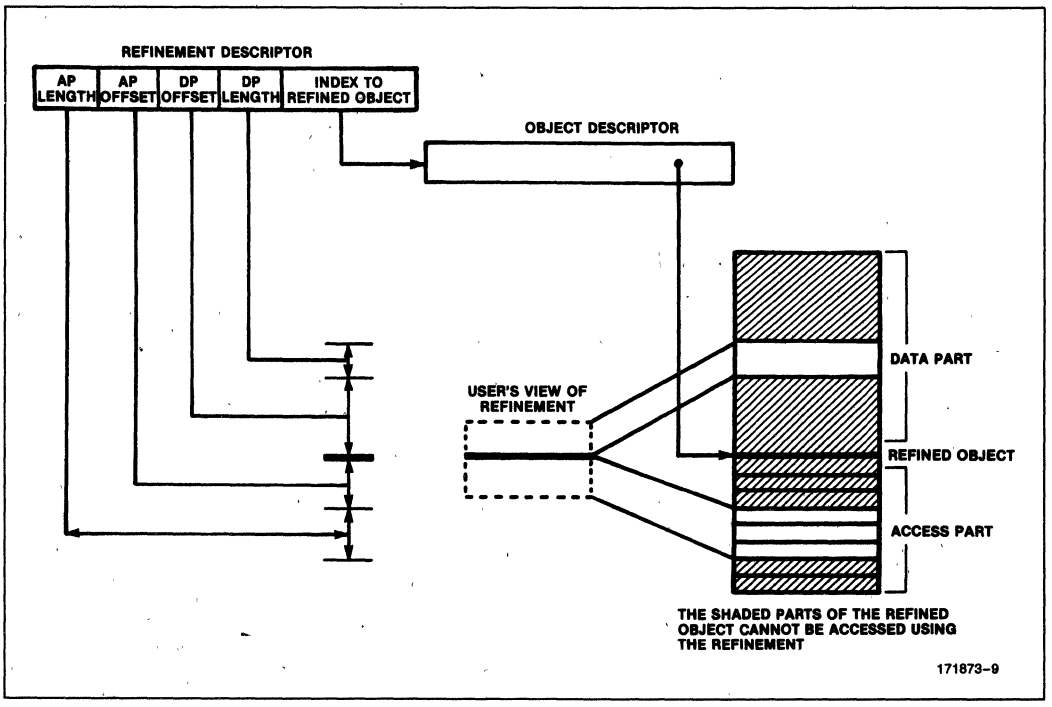


Figure 7. Refinement Object

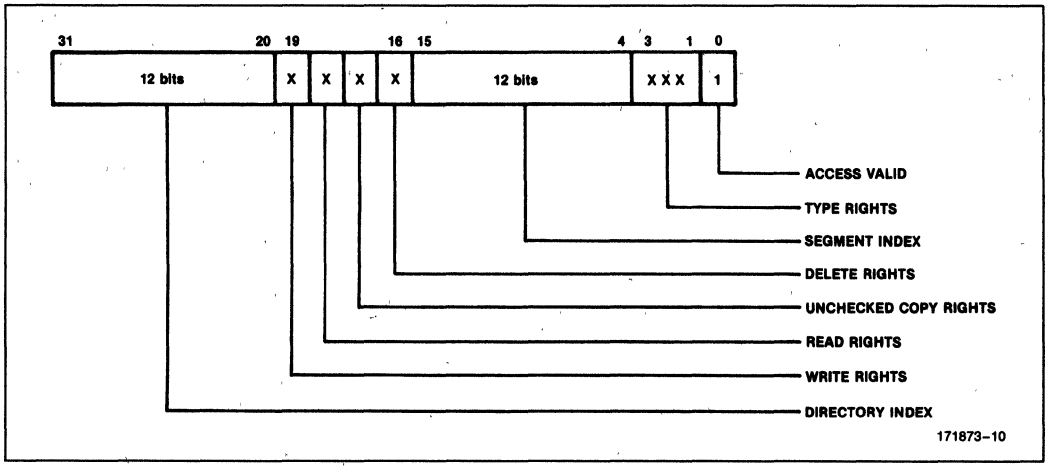


Figure 8. Access Descriptor Format

A type manager may define up to five rights for the dynamic-typed object it manages. It may selectively grant rights to a user when it creates an object and returns the reference for it. When a type manager receives a reference for an object, it may compare the rights in the reference to the operation requested by the user. The manager of a "bank account" object, for example, may permit all users to credit and debit bank accounts, but only a few may be granted the right to close an account.

Finally, the hardware performs a bounds check to insure that the displacement component of a logical address in fact falls within the target object. This is done by simply comparing the displacement to the object's length (contained in the object descriptor).

For improved performance, the hardware holds length and rights information for frequently-used objects, as well as their physical base addresses, and virtual memory control information, on chip.

Object Management

Reference Manipulation A procedure sometimes needs to manipulate the entries in its access descriptor list (as opposed to the objects the list references). The common base architecture defines operations that permits copying an access descriptor from one "slot" to another, and for nulling an access descriptor. This last operation is used to delete the reference to an object that is no longer needed. Another operation permits a procedure to inspect an access descriptor, for example, to examine the rights bits. A procedure may similarly inspect the object table entry indexed by an access descriptor to see, for example, if the object is dynamic-typed (i.e., the entry is for a type control object, rather than an object descriptor).

Object Creation and Deletion Free storage in central system memory is accounted for in system objects called **storage resource objects,** or SROs. SROs are lists of unallocated blocks of memory. Given a reference to an SRO, a procedure can create a new object dynamically; the instructions that create new objects automatically update the SRO from which storage is obtained.

Objects have "lifetimes": they come into being and occupy storage, and they also disappear, giving up their storage. The 432 common base architecture distinguished between short-term and long-term objects. A short-term object exists for the lifetime of the procedure that creates it; that is, it is allocated when the procedure is called and it is deallocated when the procedure returns. An operand stack, for example, is automatically created when a procedure is called, and a procedure may create an object to pass as a parameter to another procedure. A long-term object exists after its creating procedure returns; in fact, it lives indefinitely, until there are no object references left for it. A type manager "create" operation will create a long-term object.

As mentioned, short-term objects are deallocated automatically by the hardware when the creating procedure returns to its caller. Long-term objects are deallocated by a software routine called a **garbage collector.** This operating system routine sweeps through memory looking for objects that no longer have references left to them. When such an object is found, the garbage collector reclaims the storage occupied by the object by removing the object's descriptor from the object table and returning the storage block(s) occupied by the object to the SRO from which it was allocated.

Garbage collection is a complex operation; most conventional systems let garbage accumulate until a request for memory allocation cannot be satisfied. Then they halt normal execution, collect garbage, and resume operation. To avoid this suspension of service, the 432 architecture defines a **reclamation bit** in each object descriptor. By setting the reclamation bit whenever it copies an object reference, the hardware permits garbage to be collected on-the-fly, in parallel with normal execution. The operating system performs garbage collection automatically, relieving programmers of much of the burden of storage management.

Mutual Exclusion It is perfectly possible for two processes to hold references for the same object (see Figure 9, for example). If both processes only read the object they need not coordinate their operations. Consider, however, an object that accumulates the total number of transactions handled by several processes. Every so often each of these processes adds the number of transactions it has handled in the preceding time period to a field in this object. Although the addition is performed in a single machine instruction, at least three memory accesses are required to complete the operation: 1) read the old total; 2) read the increment; 3) write the new total. The integrity of the total is in jeopardy when two processes update it at nearly the same time.

The 432 has three classes of mutual exclusion mechanisms: I/O locks, object locks, and indivisible operations. Each object's storage descriptor contains a bit called the I/O lock. When an Interface Processor opens a window on the object, it checks to see that the object is not already I/O-locked, and then it locks it for the duration of the I/O transfer. Software running on a GDP may check for an I/O lock by inspecting an object's storage descriptor.

An **object lock** field may be defined for any data object. Processes that update such an object agree by convention not to update the object without first locking it, and further agree to unlock the object as soon as exclusion is no longer required. The LOCK OBJECT is conditional: it returns a value that indicates if the object was successfully locked (i.e., was not locked by another process). A process should refrain from accessing the object until it successfully locks it. In general, an object's type manager will take care of locking and unlocking the objects it manages, eliminating the need for object users to know anything about locks.

Note that a processor locks a 432 system object when it needs exclusive access to it during the execution of a high-level operation. This prevents another processor, or an executive routine, from interfering with a critical operation on the object.

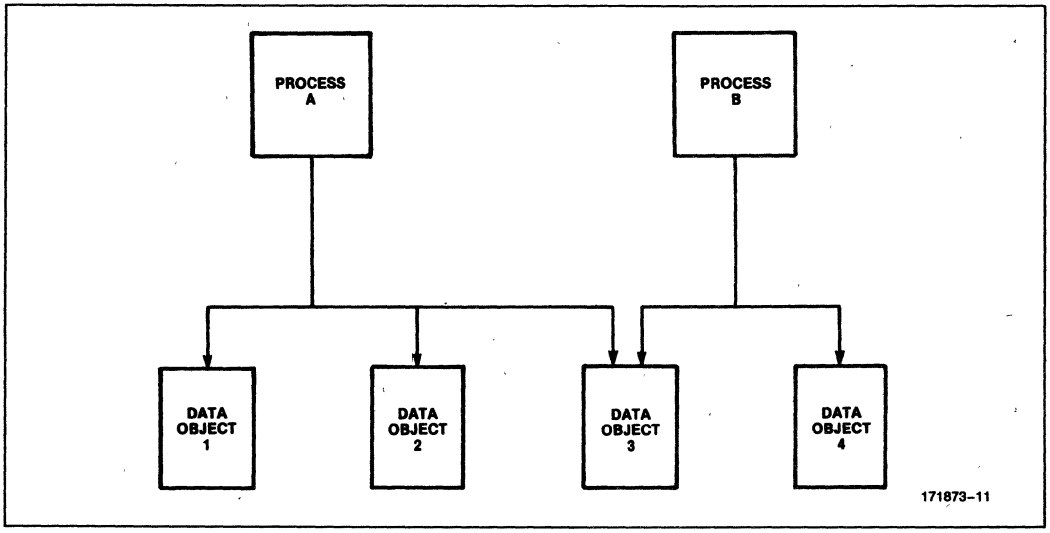


Figure 9. Two processes can share access to the same object. In the figure, both Processes A and B can access Data Object 3. In these cases, some mutual exclusion mechanism must be used to prevent inconsistent updating.

Often an object only needs to be locked for the duration of one instruction. The common base architecture defines operators that permit addition and bit field insertion (1 to 32 bits) to be performed **indivisibly.** When an indivisible operator is executed, the processor signals a read-modify-write bus cycle; the system memory controller must not permit a second RMW-write access to the target memory until the updated value has been written back into memory. Thus, the integrity of a shared value is guaranteed so long as programs that update it do so with indivisible operations.

While a refinement is effectively a new object that is a contiguous subset of an existing one, it actually requires only a new access descriptor, thereby saving storage and execution time. A process with a reference to a refinement has no knowledge of the "underlying" refined object. The process that created the refinement, however, can "retrieve" the refined object from a reference to the refinement.

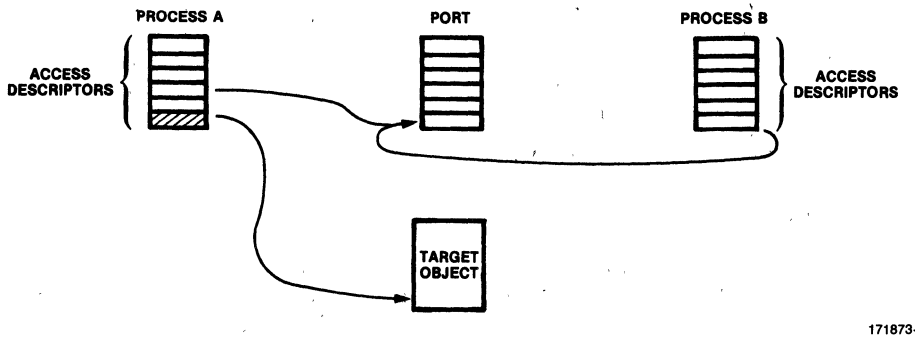
Process Communication

Except as they hold references to the same objects, 432 processes are completely independent of one another. Two processes may execute alternately on the same processor, or they may execute simultaneously on different processors. The 432 interprocess communication facility enables processes to communicate with each other by transmitting access descriptors (as messages) through memory during execution.

Since any object reference can be transmitted, process communication is an extremely efficient and versatile facility; it provides the basis for I/O operations and process dispatching in addition to more traditional message passing. Process communication can also be used to implement another form of mutual exclusion.

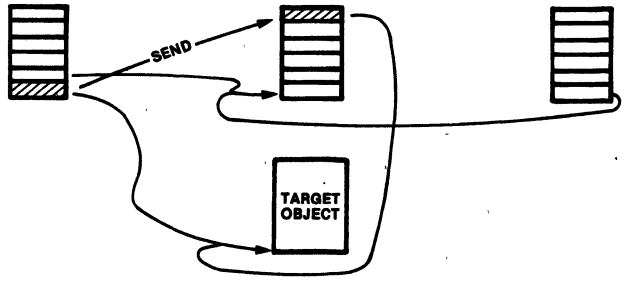
A complete transmission consists of a send and a corresponding receive. Since processes execute **asynchronously** with respect to each other, the time at which a process desires to send a message is unrelated to the time at which another process is ready to receive a message. Further, the rates at which processes send and receive are, for the most part, unpredictable. This is in contrast to the **synchronous** communication of procedures within the same process, which may pass object references as parameters in ordinary call and return operations. A call effectively suspends execution of the caller and starts execution of the called procedure; a return terminates the called procedure and resumes the caller. 432 interprocess communication, on the other hand, allows the communicating processes to run concurrently.

The 432 **port** object provides the synchronization and buffering needed for asynchronous process communication. Conceptually, a port is a queue; two processes with references to the same port have a channel over which they can communicate. A process wishing to transmit a message executes a SEND operator, which **copies** the access descriptor to the port (see Figure 10). A processes ready to obtain the message executes a RECEIVE operator, which moves the access descriptor at the head of the queue to the receiver's object reference list, thereby making the object accessible and deleting it from the queue.



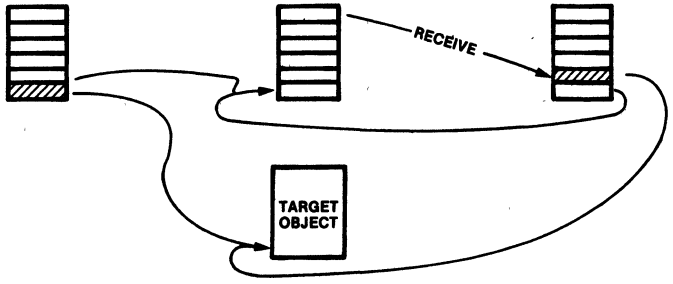
1) Both Process A and Process B have an access descriptor for the port, while A has an access descriptor for the target object.

Figure 10. Simple Message Transmission



171873-13

2) A sends message to port; an access descriptor for the target object is queued.



171873-14

3) B receives message from port; Processes A and B now have access to the target object.

Figure 10. Simple Message Transmission (Continued)

When a port is created, it is given a queuing discipline, which may be first-in-first-out (FIFO), or priority/deadline. A send to a FIFO port inserts the message at the back of the queue. A send to a priority/deadline port inserts the message according to priority and deadline parameters associated with it (and the messages already enqueued, if any). Insertion is done so that the highest priority messages are at the front of the queue; within a priority level, messages with the shortest (least) deadline are placed in front. A receive operation always takes the message at the front of the queue.

The simple SEND and RECEIVE operators are unconditional: they imply that the executing processes is willing to **block** if the port is full (or empty). A SEND to a full port or a RECEIVE at an empty port enqueues a reference to the sender's process object at the port; further execution of the process is blocked while it effectively waits at the port. When SENDs and RECEIVEs executed by other processes make it possible to complete the original operation, the process is automatically unblocked and execution may resume. A process waiting at a port due to a blocked RECEIVE, for example, is unblocked when another process SENDs a message to that port.

The CONDITIONAL SEND and CONDITIONAL RECEIVE operators, in contrast, never block: these operators return a value that indicates whether the operation was "successful." This signal gives the process the option of doing some other work and attempting the send or receive later. Other operators, called SURROGATE SEND and SURROGATE RECEIVE enable even more sophisticated forms of interprocess communication.

A port is also useful as a mutual exclusion mechanism. Processes that periodically require exclusive access to a shared object can designate a port to hold a single "key" to the object; the key can simply be a null access descriptor. When a process needs to obtain exclusive access, it does a receive operation on the port to pick up the key. (If the key is in use, the process blocks at the port.) When it actually receives the key, the process has exclusive access to the shared object. As soon as it has finished with the object, the process sends the key back to the port, making it available to other processes.

Interprocessor Communication

To coordinate the activities of multiple processors, all 432 processors can receive and respond to a set of predefined messages (see Table 4). For the most part, these messages are sent by processors on their own initiative. System software may also send a message to a particular processor, or may broadcast a message to all processors in the system.

When a system's memory manager moves an object or swaps it out of memory, for example, it will broadcast a "flush cache" message to all processors. This eliminates the possibility that a GDP will reference the object with an on-chip cache address which has been made invalid by the action of the memory manager. Sending a message to a processor requires a reference (with proper system rights) to that processor's process object; by controlling distribution of process object references, system software can likewise limit the ability to send processor messages.

Table 4 contains the Interprocessor Communication message codes in decimal along with a short description of the message:

Table 4. IPC Message Codes

0	Wakeup
1	Start
2	Stop
3	Accept Global IPCs
4	Ignore Global IPCs
5	Requalify Object Table Cache
6	Reset Processor
7	Requalify Processor
8	Requalify Process
9	Requalify Context
10	Requalify Data Object Cache
11	Enter Normal Mode
12	Enter Alarm Mode
13	Enter Reconfiguration Mode
14	Enter Diagnostic Mode

Timekeeping

The central system runs on a single time standard which is reflected in the on-chip clock of every 432 processor. The on-chip clock is a 16-bit accumulator which is driven by the PCLK signal. All 432 processors are tied to PCLK, which is independent of the component (CLK_A, CLK_B) clocks. When PCLK is asserted by an external timing source, all processor clocks increment ("tick") in unison.

PCLK's frequency, at thus the duration of one system unit, is set according to the resolution required by the application; 200 microseconds is a typical value. As discussed later, the system unit provides the basis for GDP processing and dispatching.

Each GDP process object contains a clock field that is automatically maintained by the hardware. A process clock indicates the number of system time units that the process has been bound to a processor (i.e, how long it has been executing). Using a 200 microsecond clock, a process clock can accumulate over 236 hours before turning over. Process clocks can be read by software and provide the basis for execution-time charging algorithms and for adaptive process scheduling.

Exceptions

Software Faults Most modern computers provide a facility for detecting errors during execution; for example, many CPUs detect arithmetic overflow or an application program's attempt to execute a "privileged" instruction. The 432 extends this concept into a comprehensive, structured software **fault** system (see Table 5).

A software fault is an exceptional condition uncovered by a processor during execution. It may be a simple computational error (e.g., square root of a negative number), an attempted protection violation, or a condition that requires off-line handling though not an error or violation. Whatever the source, the architecture recognizes that normal computation cannot continue until the exceptional condition is resolved.

The architecture defines software fault **detection** and fault **reporting;** that is, the notification that a fault has occurred and provision of information describing it. Software fault **handling,** which may include fault **recovery** in many cases, is the province of application or—more frequently—system software. A software fault may be detected at any time: during the execution of an instruction or command, while a processor is performing an operation on its own initiative, during an IP data transfer, and so forth. A processor reports a software fault by first recording descriptive information in the predefined **fault information area** of a system object. This information describes the nature of the fault and provide additional information that may assist software in recovering from it. The fault handler examines the fault information and takes "appropriate action," as defined by the application. This may vary considerably according to the application and the nature of the fault.

Table 5. Detectable Software Faults

General Fault Groups

Memory Reference Faults:
- Segment Overflow Fault
- Memory Overflow Fault
- Read Rights Fault
- Write Rights Fault
- Bus Error

Instruction Fetch Fault

Data Part Cache Qualification Faults
Data Part Access Faults:
- Access Descriptor Validity Fault
- Object Descriptor Type Fault

Object Table Qualification Faults:
- Object Descriptor Type Fault
- Object Type Fault

Access Environment Altered Faults:
- Access Descriptor Validity Fault
- Object Descriptor Fault

Data Operator Fault Groups

Domain Error Fault
Overflow Fault
Underflow Fault
Inexact Fault

Non-Instruction Interface Faults

Initialization:
- Object Qualification Faults (Processor)
- Object Qualification Faults (Object Table Directory)

IPC Faults
- Object Qualification Faults (PCO)
- PCO Response Count Fault
- PCO Lock Fault

Idle:
- Delay Port Service Faults

Process Binding:
- Object Qualification Faults (Carrier)
- Process Lock Faults
- Process Qualification Faults
- Port Operation Faults

Process Selection:
- Delay Port Service Faults
- Object Qualification Faults
- Port Operation Faults

Object Operator Faults

Branch
Branch True
Branch False:
- Instruction Pointer Overflow Fault
- Instruction Object Displacement Fault

Branch Indirect:
- Instruction Object Displacement Fault

Branch Intersegment
Branch Intersegment without Trace
Branch Intersegment and Link:
- Object Qualification Faults (Instructions)
- Instruction Object Displacement Fault

Copy Access Descriptor:
- Store Access Descriptor Faults

Null Access Descriptor:
- Destination Delete Rights Fault

Amplify Rights:
- TCO Type Rights Fault
- Object Qualification Faults (TCO)
- Type Fault
- Race Condition Fault

Retrieve Type Definition:
- Source AD Validity Faults
- Store Access Descriptor Faults

Create Refinement:
- Source AD Validity Fault
- Object Descriptor Type Fault
- Offset and Length Compatibility Fault
- Refinement Overflow Fault
- Level Fault

Create Typed Refinement:
- TCO Type Rights Fault
- Source AD Validity Fault
- Object Descriptor Type Fault
- Type Fault
- Offset and Length Compatibility Fault
- Refinement Overflow Fault
- Level Fault

Create Typed Object:
- Descriptor Allocation Faults
- Object Qualification Faults (TCO)
- TCO Type Rights Fault
- Level Fault
- Segment Allocation Faults
- Store Access Descriptor Faults

Table 5. Detectable Software Faults (Continued)

Inspect Object:
- Access Path Object Descriptor Fault

Lock Object:
- Source Representation Rights Fault

Unlock Object:
- Source Representation Rights Fault
- Object Lock ID/Type Fault

Call
Call through Domain:
- Object Qualification Faults (Domain)
- Domain Access Index Overflow Fault
- Instruction Object Type Rights Fault
- Object Qualification Faults (Instructions)
- Context Parameters Size Fault
- Context Type Rights Fault
- Object Qualification Faults (Context)
- Instruction Object Displacement Fault

Return:
- Context Type Rights Fault
- Context Qualification Faults
- Object Qualification Faults (PSO)
- Object Qualification Faults (Object Table)
- PSO Lock Fault
- Instruction Object Displacement Fault

Block Move:
- Offset Overflow

Return and Fault:
- Return Fault

Send
Receive
Conditional Send
Conditional Receive
Delay Process
Send Process:
- Port Type Rights Fault
- Level Fault

Surrogate Send
Surrogate Receive:
- Surrogate Carrier Validity Fault
- Surrogate Carrier Type Rights Fault
- Destination Port Type Rights Fault
- Port Type Rights Fault
- Level Fault

Set Process Mode:
- Process Object Type Rights Fault
- Process Object Access Mismatch Fault

Send to Processor:
- PCO Type Rights Fault
- Object Qualification Faults (PCO)

Move to Interconnect
Move from Interconnect:
- Odd Displacement Fault
- Odd Interconnect Descriptor Base Address Fault
- Object Qualification Faults (Interconnect)

The common base architecture recognizes that some faults are more serious than others; indeed certain faults, such as "not allocated" (i.e., an object needs to be swapped in from external storage) will be routine in many systems. Accordingly, software faults are divided into levels based on their impact on the system and the amount of information required to resolve them. This permits software to provide a response that is appropriate to the severity of the problem and to minimize the disruption that handling the fault may have on the rest of the system.

In general, the philosophy is to keep the unaffected parts of the system running while the fault is handled outside the normal flow of execution. Processors record fault information in the system object that corresponds to the fault-level; for example, information describing a process-level fault is recorded in the process object.

A context is a single instance of a procedure in execution. A **context-level** fault is one that can normally be handled within the process (i.e., by application code). Ada, for example, permits programmers to write exception handlers that will respond to GDP context-level faults.

A **process-level** fault prevents the current process from continuing until the fault is handled, but does not affect other processes. GDPs and IPs respond to this situation similarly. Generally, the procedure is to remove the offending process from the set of active processes by sending its process object to a **fault** port and then dispatching the next ready process.

Each process is associated with a fault port; a fault port is an ordinary port that queues messages that happen to be references to "broken" processes. An operating system **fault process** can receive these process objects and attempt to "repair" them, that is, recover from the fault. For example, if the fault process determines that the fault is "object not allocated," it can notify the system's virtual memory manager to swap in the needed object. When this has been done, the fault process can send the process off to its dispatching port (see "Scheduling and Dispatching").

A **processor-level** fault threatens (but might not absolutely prevent) continued execution by the processor. The GDP and IP respond to this situation differently, but the basic procedure is to run a processor diagnostic program.

At the final level, the processor cannot do anything, not even record fault information. It therefore halts and asserts its FATAL pin. Software on some other processors might monitor this pin; for example, it could be routed to an interconnect register and periodically sampled. A halted processor can be restarted by hardware (asserting its INIT pin) or software (sending a START IPC).

Interrupts. Each 432 processor has an ALARM pin which can be asserted to signal the occurence of an extremely high priority external event. A typical example is imminent power failure. In general, when ALARM has been asserted, the GDP will complete its current instruction and then invoke a designated software process (waiting at the ALARM port).

Data Types

The memory formats of the GDP's eight basic data types are illustrated in Figure 11. Any data type may be stored on any byte boundary (performance is improved, however, when data is aligned on physical memory boundaries). The types are divided into four classes: character, ordinal (unsigned integer), integer, and real. These data types correspond directly to the "primitive" types defined in most high level languages. Implementing the essential types in hardware, with a choice of storage requirements for each class, helps ensure that compiler-generated code sequences are both compact and fast.

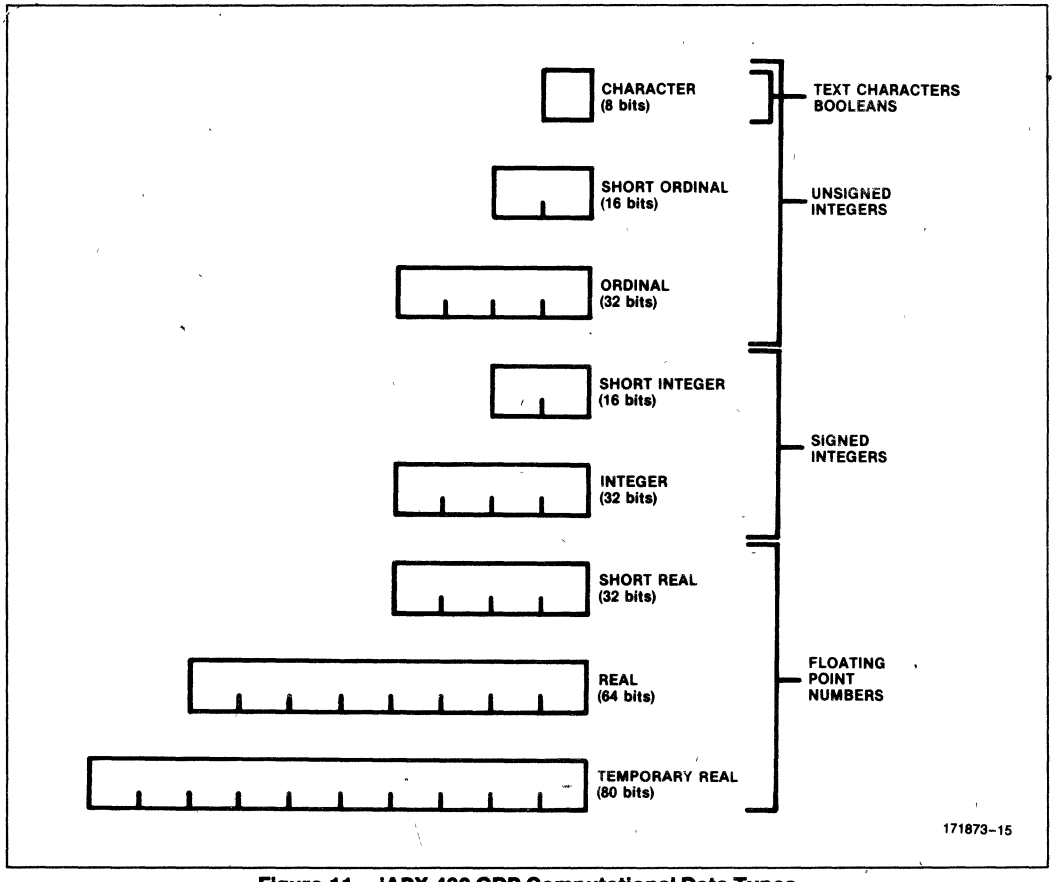


Figure 11. iAPX 432 GDP Computational Data Types

Table 6 gives the attributes of the numeric data types. Of particular note is the temporary-real data type. The extra range and precision of this type contributes to the production of consistently safe, reliable floating-point algorithms. As its name implies, the temporary-real type is intended for holding intermediate computational results. The inputs and outputs of a calculation should be defined as short-real or real, according to the range and accuracy of the available data. All intermediate results should be held in temporary-real, with the final conversion to the output at the end of the computation.

Extremely large and small values are most likely to occur in intermediate computations; using temporary-real for these makes overflow and underflow exceedingly unlikely in most applications. Temporary-real's extended precision also prevents round-off errors from accumulating during long computations; the only significant round-off occurs at the conversion from temporary-real to the real or short-real format of the final result.

Instruction Set

Table 7 shows the instructions the GDP provides for its data types. The symmetry of the instruction set with respect to data types simplifies compiler code generation.

In the data transfer group, the ZERO and ONE instructions write a constant into an operand. MOVE copies a variable, popping the stack if it is the source, pushing it if it is the destination. SAVE copies the stack top without popping the stack; it can be used to duplicate the stack top.

The logical instructions perform the customary operations. XNOR is the complement of XOR (exclusive OR): where XOR returns 1-bits when corresponding bits are unequal, XNOR returns 1-bits when corresponding bits are equal. It is thus the Boolean equivalent of "equals."

In the arithmetic group, the REMAINDER instruction performs **exact** modulo division; it is very useful for reducing an argument to a periodic transcendental function (e.g. tangent) to the range accepted by the function without introducing round-off error. SQUARE ROOT executes in about the same time as ordinary division; programmers need not contort algorithms to eliminate time-consuming square roots.

Table 6. Numeric Data Types

Data Type	Significant Digits*	Approximate Range
Character	2	$0 \le x \le 255$
Short-Ordinal	4	$0 \le x \le 65{,}535$
Ordinal	9	$0 \le x \le 4{,}294{,}967{,}295$
Short-Integer	4	$-32{,}768 \le x \le 32{,}767$
Integer	9	$-2{,}147{,}483{,}648 \le x \le 2{,}147{,}483{,}647$
Short-Real	6–7	$8.43 \times 10^{-37} \le \lvert x \rvert \le 3.37 \times 10^{38}$
Real	15–17	$4.19 \times 10^{-307} \le \lvert x \rvert \le 1.67 \times 10^{308}$
Temporary-Real	19	$3.4 \times 10^{-4932} \le \lvert x \rvert \le 1.2 \times 10^{4932}$

* Decimal equivalent

Table 7. iAPX 432 Operators and Computational Data Types

	Operators	Data Types							
		Char.	Short Ordinal	Ordinal	Short Integer	Integer	Short Real	Real	Temp. Real
MOVE OPERATORS	MOVE	X	X	X	X	X	X	X	X
	SAVE	X	X	X	X	X	X	X	X
	ZERO	X	X	X	X	X	X	X	X
	ONE	X	X	X	X	X	–	–	–
LOGICAL OPERATORS	AND	X	X	X	–	–	–	–	–
	INCLUSIVE OR	X	X	X	–	–	–	–	–
	EXCLUSIVE OR	X	X	X	–	–	–	–	–
	EQUIVALENCE	X	X	X	–	–	–	–	–
	NOT	X	X	X	–	–	–	–	–
ARITHMETIC OPERATORS	ADD	X	X	X	X	X	*	*	X
	SUBTRACT	X	X	X	X	X	*	*	X
	MULTIPLY		X	X	X	X	*	*	X
	DIVIDE		X	X	X	X	*	*	X
	REMAINDER		X	X	X	X	–	–	X
	INCREMENT	X	X	X	X	X	–	–	–
	DECREMENT	X	X	X	X	X	–	–	–
	NEGATE	–	–	–	X	X	X	X	X
	ABSOLUTE VALUE	–	–	–			X	X	X
	SQUARE ROOT								X
	INDEX	–	–	X	–		–	–	–
BIT-FIELD INSERT	EXTRACT		X	X	–	–	–	–	–
	INSERT		X	X	–	–	–	–	–
	SIGNIFICANT BIT		X	X	–	–	–	–	–
RELATIONAL OPERATORS	EQUAL	X	X	X	X	X	X	X	X
	NOT EQUAL	X	X	X	X	X			
	EQUAL ZERO	X	X	X	X	X	X	X	X
	NOT EQUAL ZERO	X	X	X	X	X			
	LESS THAN	X	X	X	X	X	X	X	X
	LESS THAN OR EQUAL	X	X	X	X	X	X	X	X
	POSITIVE	–	–	–	X	X	X	X	X
	NEGATIVE	–	–	–	X	X	X	X	X
	MOVE IN RANGE				X	X			
CONVERSION OPERATORS	TO CHARACTER	–				X			
	TO SHORT ORDINAL	X	–			X			
	TO ORDINAL			–		X			X
	TO SHORT INTEGER				–	X			
	TO INTEGER	X	X	X	X	–			X
	TO SHORT REAL						–		X
	TO REAL							–	X
	TO TEMPORARY REAL		X	X	X	X	X	X	–

WHERE:
- X — Means the operator is available for the given data type.
- * — Means the operator is available for the given data type and for instructions in which one of the operands is a temporary real.
- – — Means the operator is not available and would be of little or no use if it were.
- (BLANK) — Means the operator is not available.

The bit field instructions, EXTRACT and INSERT BIT FIELD, make the manipulation of packed bit field records simple and rapid. The SIGNIFICANT BIT instruction returns the position of the "leftmost" 1-bit in an ordinal or short-ordinal. Note that the INSERT BIT FIELD is an indivisible operation; once the instruction starts to run, no other processor can perform an indivisible operation on the field until its new value has been written into memory.

The instructions in the comparison group assert a condition existing in a single variable (e.g., EQUAL ZERO) or between two variables (e.g., GREATER THAN). These instructions return a Boolean value TRUE or FALSE according to the truth of the assertion. Conditional branching is effected by following a comparison with BRANCH TRUE or BRANCH FALSE instruction.

The GDP has the full complement of 432 common base instructions plus addition data processing operations. Some of these permit changing the flow of control in a program by conditional and unconditional, and by calling a procedure. Others facilitate the manipulation of composite objects (objects made up of other objects), access to data declared global to all procedures in a process, and setting precision and rounding modes for real number computations. Finally, two of the instructions give a GDP program access to the interconnect space.

Instruction Formats

The 432's instruction codes have been designed to minimize the space the instructions occupy in memory and still allow for efficient encoding. In order to achieve the best efficiency in storage, the instructions are encoded without regard for byte, word, or other artificial boundaries. The instructions may be viewed as a linear sequence of bits in memory, with each instruction occupying exactly the number of bits required for its complete specification.

Processors view these instructions as composed of fields of varying numbers of bits that are organized to present information to the Instruction Decoder in the sequence required for decoding. A unified form for all instructions allows instruction decoding of all instructions to proceed in the same manner.

In general, GDP instructions consist of four main fields. These fields are called the class field, the format field, the reference field, and the opcode field. The reference field, in turn, may contain several other fields, depending upon the number and the complexity of the operand references in the instruction. The fields of a GDP instruction are stored in memory in the following format.

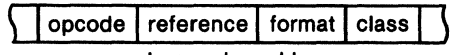
The class field is either 4-or 6-bits long, depending on its encoding. The class field specifies the number of operands required by the instruction and the primitive types of the operands. The class field may indicate 0, 1, 2, or 3 operands. If the class field indicates one or more references, a format field is required to specify whether the references are implicit or explicit and their uses.

In the case of explicit references, the format field can indicate whether or not the reference is direct or indirect. Further, the format field may indicate that a single operand plays more than one role in the execution of the instruction. As an example, consider an instruction to increment the value of an integer in memory. The instruction begins with a class field specifying that the operator is of order two and that the two operands occupy a word of storage; next, the format field indicates that a single reference specifies a logical address to be used both for fetching the source operand and for storing the result; it is followed by an explicit data reference to the integer to be incremented; and finally the instruction ends with an opcode field for the order-two operator INCREMENT INTEGER.

It is possible for a format field to indicate that an instruction contains fewer explicit data references than are indicated by the instruction's class field. In this case, the other data refefences are implicit, and the corresponding source or result operands are obtained from (or returned to) the top of the operand stack. Consider the following statement:

$$A = A + B*C$$

The instruction fragment for this statement consists of two instructions and has the following form:



171873-16

Assume that A, B, and C are integer operands. The first class field (the rightmost field shown above) specifies that the operator requires three references and that all three references are to word operands.

The first format field contains a code specifying two explicit data references supplying only two source operands. The destination is referenced implicitly so that the result of the multiplication is pushed on the operand stack. The second class field is identical to the first and specifies three required references by the operator, all to word operands. The second format field specifies one explicit data reference to be used for both the first source operand and the destination. The second source operand is referenced implicitly and popped from the operand stack when the instruction is executed.

The reference fields themselves can be of various lengths and can appear in varying numbers (consistent, of course, with the specifications in the class and format fields. If implicit references are specified, reference fields for them will not appear. Direct references will require more bits to specify than indirect references.

Following the class, format, and reference fields, the opcode field appears. The opcode field specifies the operator to be applied to the operands specified in the preceding fields.

Addressing Modes

The operands (data items) that a GDP instruction is to operate on are encoded in the instruction as data references consisting of two parts: a **base** part and an **index** part. The entire data reference can therefore be viewed as having three components: an access selection component, which selects an object; a base part of the operand offset, which provides a byte displacement to the base of the area of memory within the selected object; and an index part of the operand offset, which specifies a particular operand within that area.

The addressing is very flexible since each part of the operand offset can be specified directly or indirectly. A direct base or direct index has its value specified directly in the data reference encoding. When indirection is used, however, the value of the base or index is given by a short-ordinal value located within the currently accessible object.

There are four possible combinations of direct and indirect base and index parts, and each combination results in a different mode of reference (see Figure 12). Each of the four combinations has been used to name a data reference mode indicating the kind of data structure for which the reference would usually be used. The scalar, record, static array, and dynamic array modes correspond roughly to the direct, base, indexed, and base-plus-index modes found in many computers; all four modes are independently available for any operand specified in an instruction.

As shown in Figure 13, the displacement component may be encoded directly in the instruction or may come from base and index variables in memory (including the stack), or may consist of one direct and one indirect value. Choosing between direct and indirect specifications primarily depends on which information is fixed at compile-time and what may be computed during execution.

Note that an indirect index value (used to select an array element) is expressed naturally as the element number to be accessed. The hardware automatically **scales** the index according to the data type being manipulated by the instruction to calculate the actual byte displacement. For example, to address the third element of a vector, the indirect index variable would contain the value 3 for any type of vector—character, integer, real, etc.

A fifth addressing mode is implicitly specified when a data reference is expected (according to the "number of references" field), but none is encoded in the instruction. The data reference in this case is the operand on top of the stack. If the operand is the source, it is automatically popped from the stack; if the operand is the destination, the result of the operation is pushed onto the stack.

Large Array Indexing

The maximum size of the data part of an object is 65,636 (64K) bytes, but of course, some applications require arrays that are larger. The INDEX ORDINAL operator is used to access these large arrays.

The large array is mapped (at compile-time) into a series of objects, each with data parts that are 2,048 bytes (2K) long. All these objects are directly accessible in the current logical access environment. The INDEX ORDINAL operator works as follows:

Given:

- The size of each element in the array (i.e., a scale factor)
- The access selector for the base segment of the array
- The ordinal index for the desired array element

The operator computes:

- The access selector for the appropriate 2K data object that contains the indexed array element
- The displacement into the data part of that object in the array element

The resulting short-ordinal values can then be used with the indirect access selection mode and the record, static array, or dynamic array data reference modes to access the array element. Of course, this whole process is invisible to the typical 432 programmer who uses a high-level language and leaves the choice of machine instructions to the compiler.

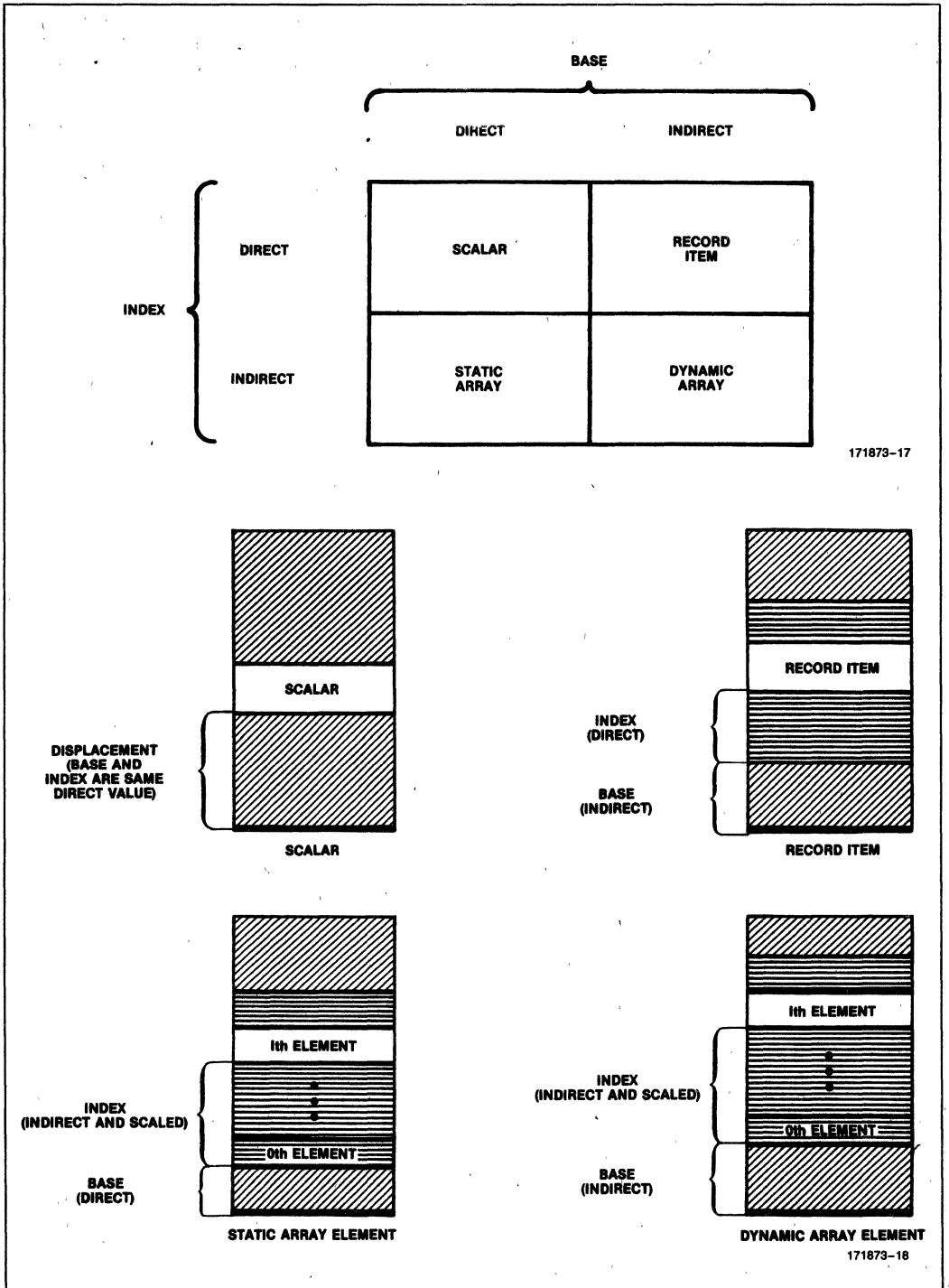
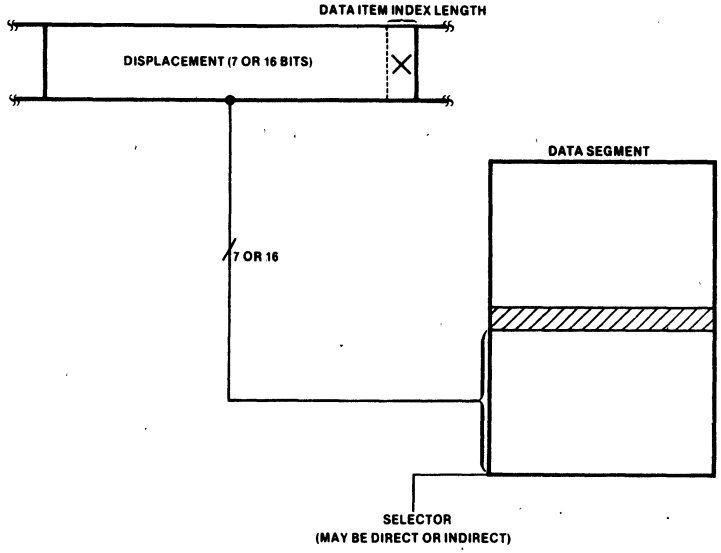
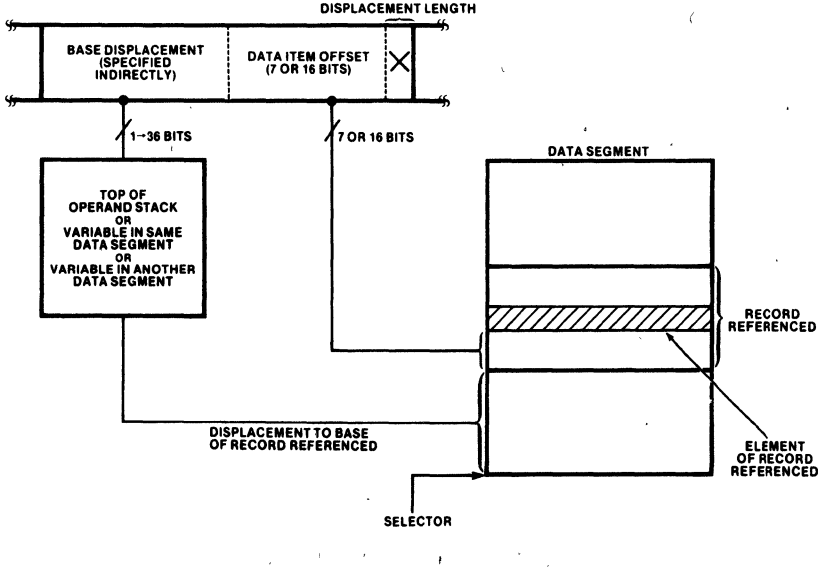


Figure 12. Addressing Modes

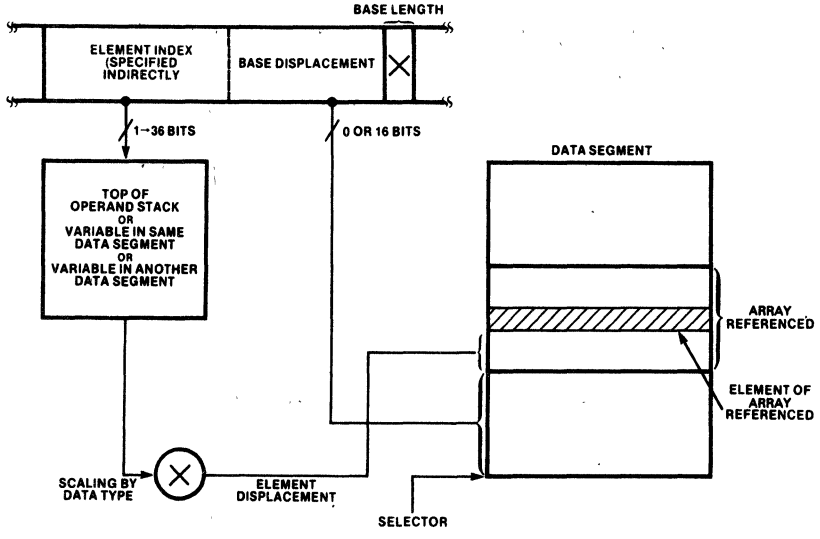


A. SCALAR DATA REFERENCE MODE

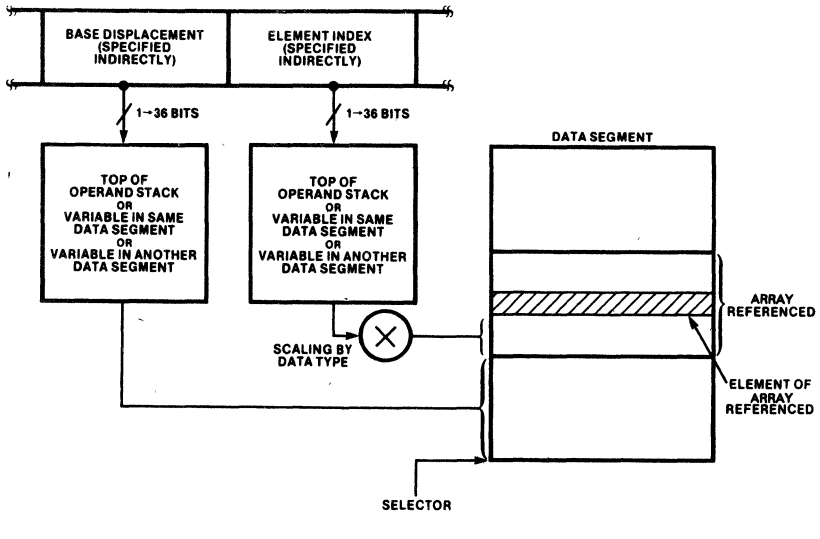


B. RECORD ITEM REFERENCE MODE

Figure 13. Modes of Displacement Generation



C. STATIC ARRAY ELEMENT REFERENCE MODE



D. DYNAMIC ARRAY ELEMENT REFERENCE MODE

Figure 13. Modes of Displacement Generation (Continued)

Scheduling and Dispatching

In most systems there will be more processes to run than there are processors. The procedure by which processes "take turns" running on GDPs is called dispatching and scheduling. Each processor is assigned to a **dispatching port,** from which it obtains its work, that is, the processes it executes. A dispatching port is an ordinary port object; it so happens that the access descriptors queued there are for process objects and processor objects. The assignment of processors to dispatching ports is defined by the application; usually all processors share one port, but each may have its own, or a processor's dispatching port may be changed by operating system software during execution.

Scheduling and dispatching are performed in two loops as shown in Figure 14. To maximize processor utilization, low-level scheduling and dispatching are performed automatically by the processor with no software intervention. Every process has four scheduling parameters; these are initially set by the operating system when a process is created. The parameters are:

1) **priority,** the relative urgency of the process;
2) **deadline,** the amount of time that may pass before the process must have a turn on the processor;
3) **service period,** the duration of one turn;
4) **period count,** the number of turns the process should be given before examining its scheduling parameters.

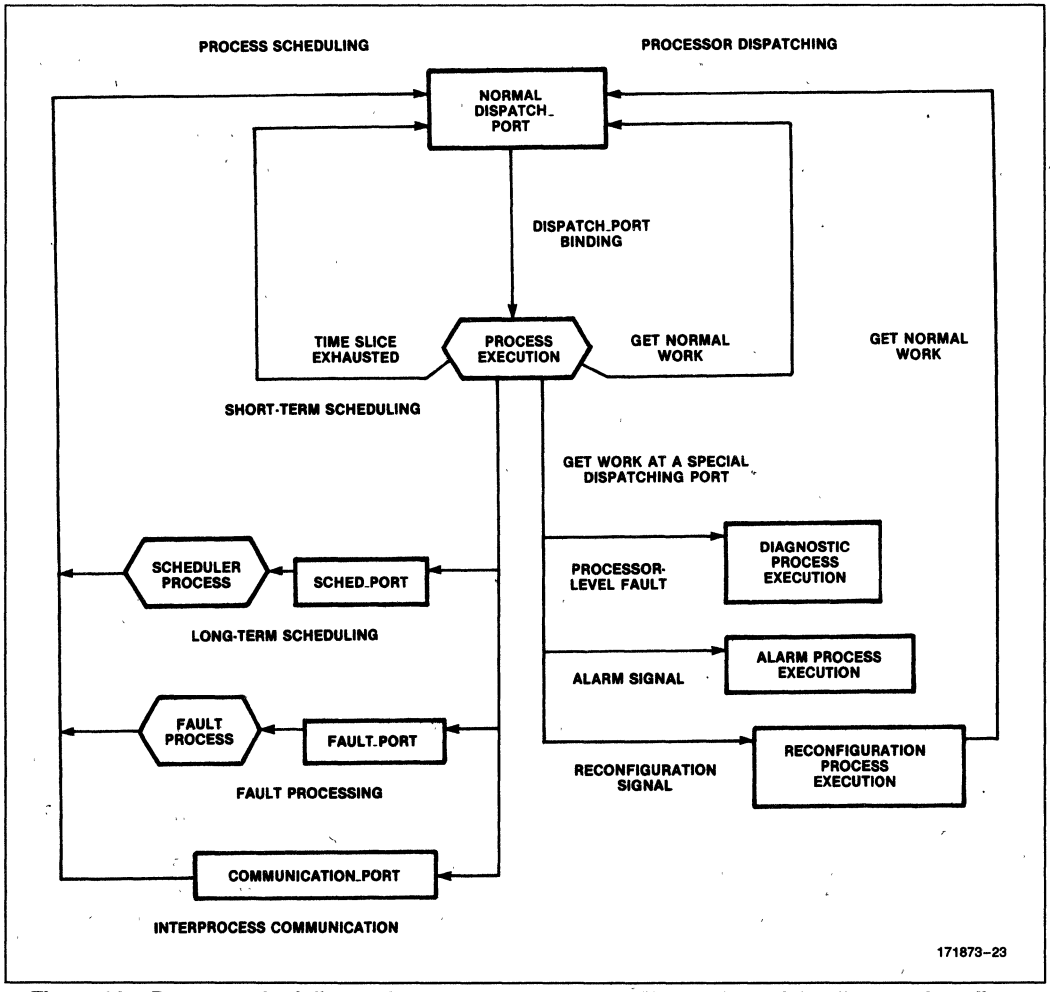


Figure 14. Process scheduling and processor dispatching. The left half of the diagram describes possible states of a process, while the right half describes possible states of a processor.

All time values are based on the system time unit. These parameters give operating system software great flexibility in setting system scheduling policy (or policies), and even altering a policy during execution.

Like the messages at any priority/deadline port, processes waiting for service at a dispatching port are ordered by deadline within priority; the highest-priority-least-deadline process is at the front of its queue. A GDP dispatching operation consists simply of "receiving" this process. The processor loads its on-chip service timer with the process's service period value and runs the processor for one service period. (Assertion of the PCLK pin increments the processor clock and decrements the service period as well.) At the end of the service period, the GDP decrements the process's period count, updates the process's process clock with the number of time units given to it, and schedules the process for another turn. If the period count has not yet expired, this is done by sending the process to its dispatching port. The send operation inserts the process into the queue according to its scheduling parameters.

If the process blocks before its period expires (before the service timer goes to zero), the period count and the process clock are also updated (with the number of actual units received), but the process is sent to a communication port instead of a dispatching port.

High-level scheduling is performed by the operating system; it gives the executive the opportunity to examine the system's performance and perhaps adjust its scheduling algorithms. When a process has exhausted all its service periods, the processor sends the process to a **scheduling port** instead of a dispatching port. The operating system scheduler receives the process, sets its scheduling and service parameters again, and sends the process back to the dispatching port, where the low-level cycle begins again.

When a GDP attempts to dispatch a process and none is available, the processor queues itself (that is, its processor object) at the dispatching port and "sleeps" until a process arrives. (A sleeping processor is almost completely idle; in a multiprocessor configuration this helps to reduce contention for use of the memory bus.) The processor that sends a process to the dispatching port also "wakes up" the sleeping processor (by means of an IPC); the awakened processor then dispatches the newly-arrived process. Operating system software may periodically check dispatching ports for idle processors and reassign them to dispatching ports that are more heavily loaded.

Designing Fault-Tolerant Systems

When used together, the five components in the iAPX 432 family provide all the logic necessary to build a system that will tolerate the failure of any single component or bus, yet continue to execute programs without error and without interruption. No software intervention is required: fault detection, isolation, and reconfiguration of the system is performed entirely by the hardware.

Each GDP is able to detect hardware errors automatically because of a capability known as Functional Redundancy Checking (FRC), so called because a second or redundant GDP checks the operations of the first or master GDP. Functional Redundancy Checking provides the low-level hardware support upon which hardware fault-tolerant modules are constructed.

During initialization, each GDP is assigned to operate as either a master or a checker (see Figure 15). While a master operates in a conventional manner, a checker places all output pins that are being checked into a high-impedance state. Those pins which are to be checked on a master and checker are parallel-connected, pin for pin, such that the checker is able to compare its master's output pin values with its own. If on any cycle, the values differ, the checker asserts HERR and the faulty components can be immediately disabled. Thus, any hardware errors can be detected as they occur and before they have had the opportunity to corrupt the operation of other components in the system.

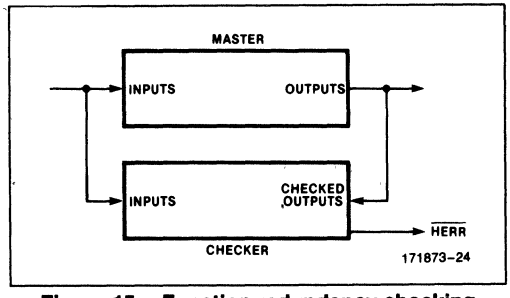


Figure 15. Function redundancy checking detects hardware errors automatically.

While FRC can be used alone to provide automatic error detection, a completely fault-tolerant system must also be able to reconfigure itself, replacing the set of failed components with another pair that is still working. In order to do so, the 432's architecture enables two pairs of master/checker components to be combined to form primary and shadow processors in a configuration known as Quad Modular Redundancy (QMR). See Figure 16.

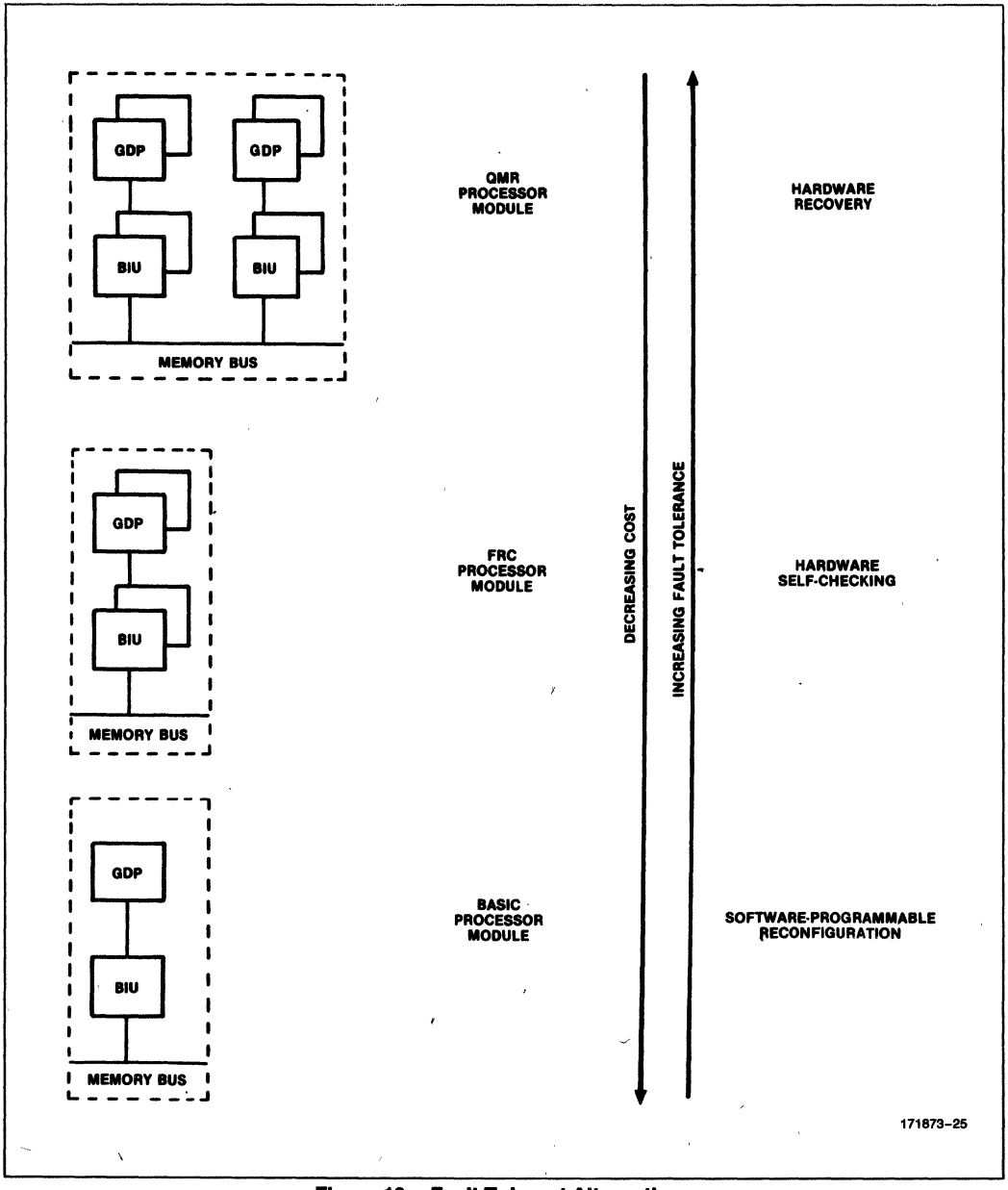


Figure 16. Fault Tolerant Alternatives

171873-002

Every module in a QMR system is paired with another self-checking module of the same type. The pair of self-checking modules operates in lock step and provides a complete and current backup for all state information in the module. The mechanism is known as module shadowing because a shadow is ready to fill in if the primary fails (or vice versa). Fault detection and recovery occurs transparently to both application and system software. When a fault is detected, the faulty pair is automatically disabled, and the remaining pair takes over. Only then is system software notified that a failure has occured.

A more complete discussion of the fault-tolerant capabilities of the iAPX 432 can be found in the **iAPX 43204-iAPX 43205 Fault Tolerant Bus Interface and Memory Control Units** data sheet (Order Number 210963).

HARDWARE IMPLEMENTATION

The iAPX 432 General Data Processor is organized as a three-stage microprogram-controlled pipeline. The first stage is the Instruction Decoder, the second the Microinstruction Sequencer, and the third the Execution Unit. The first two stages of the pipeline are physically located on the iAPX 43201 with the third stage on the iAPX 43202. Each stage, however, can be considered an independent subprocessor that operates until the pipeline is full, and then halts and waits for more work to do.

Instruction Decoder

The general task facing the Instruction Decoder (see Figure 17) is to interpret the macroinstruction stream both to extract logical addresses and to determine the next microinstruction sequence to be initiated. In doing so, it performs the following functions:

- Receives macroinstructions
- Processes variable length fields
- Extracts logical addresses
- Generates starting addresses for the micro-instruction procedures
- Generates microinstructions for simple operations

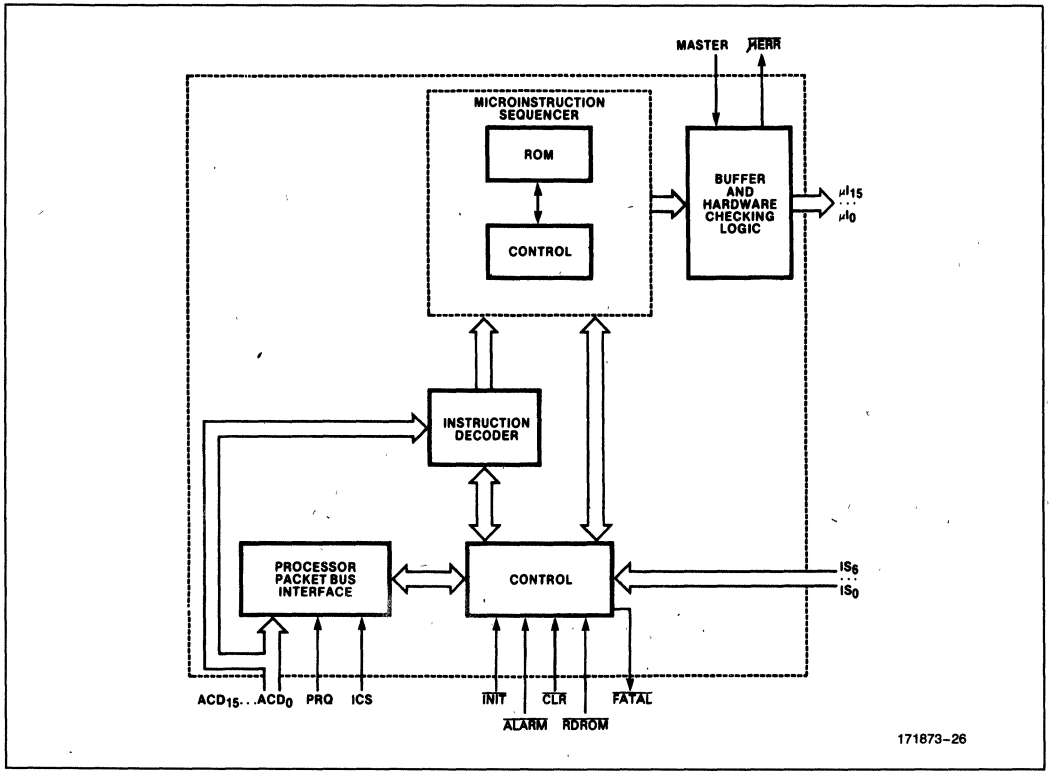


Figure 17. 43201 Block Diagram

The Instruction Decoder requests words from memory as they are needed, from one to ten bytes in a single access. Depending upon the complexity of the instruction, a 432 instruction may range from a few bits long to several hundred bits long, extending over many words.

A GDP instruction is composed of a variable number of fields and each field may contain a variable number of bits. In most cases, the encoding of a field specifies its length. The ID determines when an instruction boudary has been reached so it can properly begin decoding the next instruction.

In some cases, the interpretation of one field may depend upon the value of some previous filed. The intrepretation of the opcode (the last field in an instruction), for instance, depends on the value of the class field (the first field) in the instruction. The ID therefore saves enough information about each instruction to properly interpret each filed.

Since a GDP instruction may contain an explicit reference to some location in memory, the logical address information must be transfered to the Reference Generation Unit in order to generate the correct physical address of the operand. As with all fields in of a GDP instruction, the length of logical address fields is variable. Consequently, the ID formats the logical address and stores it until needed by the Reference Generation Unit.

Since branch instructions occur frequently, it is important to minimize the startup time for the GDP after a branch has occurred. Since an instruction may begin on any bit, the GDP is able to begin decoding at any point in a segment.

Microinstruction Sequencer

The Microinstruction Sequencer (MS) decides which microinstruction should be sent to the Execution Unit (EU) for each cycle. In doing so, it performs the following functions:

- Executes microcode sequences out of an on-chip, 4k by 16-bit ROM
- Responds to bus control signals
- Invokes macroinstruction fetches
- Issues microinstructions to the EU
- Initiates interprocessor communication and fault handling sequences

The MS chooses from two sources of microinstructions: they may come from either the ID or from the ROM in the MS. After issuing one microinstruction, the MS then computes the address in ROM (if any) for the next microinstruction. Since the EU may require differing lengths of time to complete some microinstructions, the MS waits for the requested operation to be completed before issuing the next one.

Execution Unit

The iAPX 43202 contains the third stage of the GDP pipeline—the Execution Unit (see Figure 18). The EU receives microinstructions from the 43201 and routes them to one of the two independent subpro-

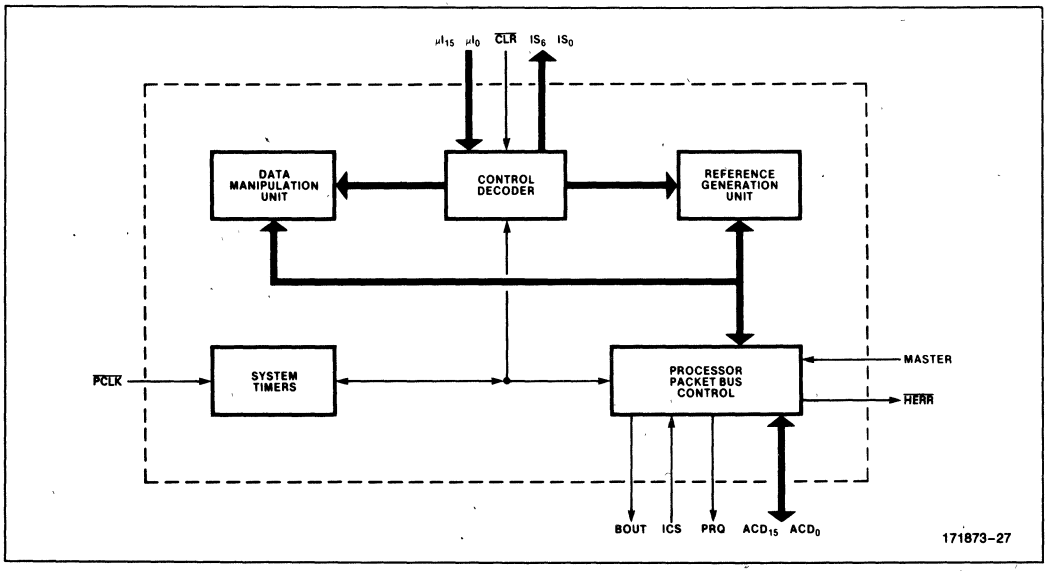


Figure 18. 43202 Block Diagram

cessors that comprise it: the Data Manipulation Unit (DMU) and the Reference Generation Unit (RGU).

While the EU executes most microinstructions in one clock cycle, each of the subprocessors has an associated sequencer that may run for many cycles in response to certain microinstructions. These sequencers are invoked, for example, for floating operations in the DMU and Processor Packet bus transactions in the RGU.

The DMU contains the registers and arithmetic logic to perform the following functions:

- Hardware recognition of nine data types
- 16- and 32-bit multiply, divide, and remainder through a built-in state machine
- Control functions for 32-, 64-, and 80-bit floating point arithmetic.

The RGU performs the following functions:

- Translates 40-bit virtual addresses into 24-bit physical addresses
- Enforces the capability-based protection system
- Sequences 8-, 16-, 32-, 64-, and 80-bit memory accesses
- Controls on-chip top-of-stack register

When a reference to a given memory segment has been translated from its logical representation to a physical address, a cache in the RGU maintains the physical base address as well as the length of the segment. Further references to the same segment reuse this information for additional address translations. A least-recently-used algorithm is implemented in hardware to determine which segment base-length pair to replace when a new segment is referenced. To further increase performance, the top 16-bit element in the operand stack is cached in the DMU.

In enforcing capability-based addressing, every memory reference is checked by the RGU to see if it is within the length of its segment, and the type of access (read, write, etc.) is verified to make certain that the object has the proper rights to perform the operation.

The iAPX 43201 and iAPX 43202 components together form a GDP. Figure 19 shows a logical representation with both units interfacing to the Processor Packet bus as a single processor. Figure 20, in turn, shows the physical layout.

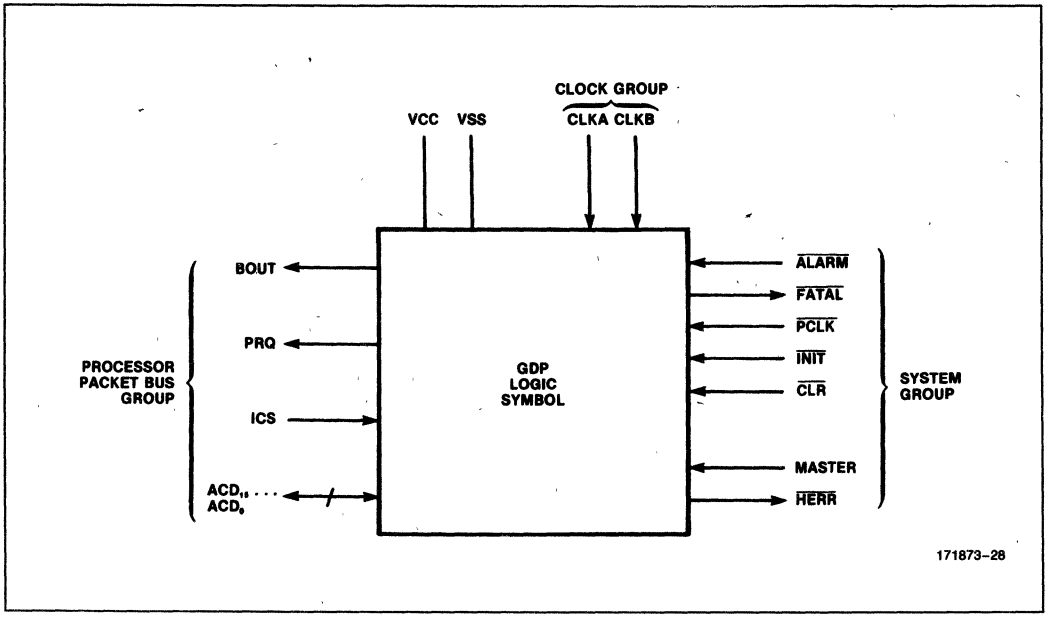


Figure 19. GDP Block Diagram

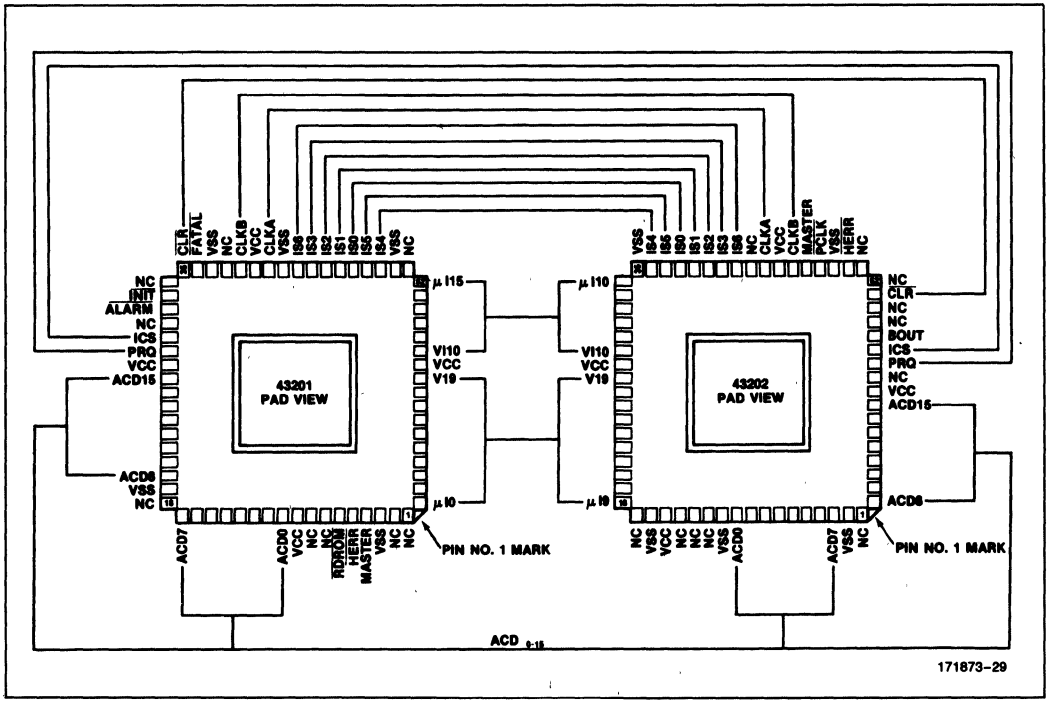


Figure 20. GDP Interconnect

PROCESSOR PACKET BUS DEFINITION

Processors sharing the same memory must contend for access to that memory over one or more system buses. Therefore, efficient bus utilization is essential in a multiprocessing system. A simple and efficient approach to building a 432 interconnect system is to use the iAPX 43204 Bus Interface Unit; the VLSI component provides the necessary circuitry to interconnect 432 processors with from one to eight memory buses. Some system designers, however, may prefer to take other approaches to the interconnect design to optimize the cost/performance ratio of the hardware for their specific application. With that requirement in mind, Intel formulated an iAPX 432 packet bus protocol which supports a wide range of system bus architectures.

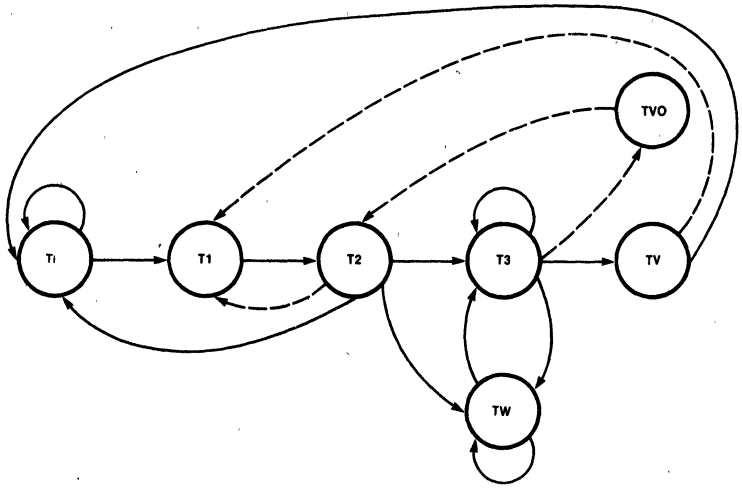
To reduce bus occupancy and increase the performance range of 432 systems, the packet bus protocol separates processor requests and replies into separate packets. A processor can issue a request packet and leave the system bus free until the reply packet is returned from memory.

As a second method of maximizing the efficiency of bus utilization, the packet bus protocol allows varaible length packets of data. If a processor wishes to read a 64-bit operand, it can be done with a single request and reply packet. Thus, fewer individual storage requests are required to process long operands. This aspect of the protocol enables processors to interface easily to 16-bit, 32-bit, or even 64-bit system buses.

This section describes the 19 signal lines that compose the Processor Packet bus and their timing relationships. While this section defines all valid bus activities, the processors do not necessarily perform all allowed activities; nevertheless, slaves to the Processor Packet bus must support all state transitions to ensure compatibility (see Figure 21).

The Processor Packet bus consists of three control lines:

- PRQ (Processor Packet bus request)
- B_{OUT} (Enable Buffers for Output)
- ICS (Interconnect Status)



171873-30

* Note that the broken transitions in the GDP state diagram are not generated by the GDP component pair.

Initial State	Next State	Trigger
Ti	T1 Ti	Bus cycle desired No bus cycle desired
T1	T2	Unconditional
T2	T3 Tw T1 Ti	ICS high ICS low Canceled, Access Pending Canceled, No Access Pending
T3	T3 Tw Tv Tvo	Additional transfer required, ICS high Additional transfer required, ICS low All transfers completed, no overlapped access Current write with overlapped access
Tv	Ti T1	No access pending Access pending
Tvo	T2	Unconditional
Tw	Tw T3	ICS low ICS high

Figure 21. Processor Packet Bus State Diagram

PRQ has two functions whose use depends upon the application; for example, PRQ either indicates the first cycle of a transaction on the bus or the cancellation of a transaction initiated during the previous cycle. Of the three control lines, B_{OUT} has the simplest function, serving as a direction control for buffers in larger systems which require more electrical drive than the processor components can provide. The ICS signal has three different interpretations depending on the state of the Processor Packet bus transaction. It may indicate whether or not:

- An interprocessor communication (IPC) is waiting,
- A slave requires more time to service the processor's request, or
- A bus error has occurred.

The bus also includes 16 three-state Address/Control/Data lines (ACD_{15}–ACD_0). These lines emit information to specify the type of cycle being initiated; transmit addresses, data to be written, and control information; and during a read operation, receive data returned to the processor. Details of the ACD operation are summarized below.

Address/Control/Data Lines

In the first cycle (T1 or Tvo) of a Processor Packet bus transaction (indicated by the rising edge of PRQ), the eight high-order ACD bits (ACD_{15}–ACD_8)specify the type of the current transaction. In this first cycle, the low-order ACD bits (ACD_7–ACD_0) contain the least significant eight bits of the 24-bit address.

During the next cycle (T2), the remainder of the address is presented on the ACD pins, aligned so that the most significant byte of the address is on ACD_{15}–ACD_8 while the mid-significant byte is on ACD_7–ACD_0. If PRQ is asserted during T2, the access is cancelled and the ACD lines are not defined.

During the third bus cycle (T3 or Tw) of a Processor Packet bus transaction, the processor presents a high impedance to the ACD lines for read transactions and asserts data for write transactions.

Once the bus has entered T3 or Tv, the sequence of state transactions depends on the type of cycle requested during the preceding T1 or Tvo. Accesses ranging in length from 1 to 10 bytes may be requested (see Table 8). If a transfer of more than one double byte has been requested, T3 must be entered for every double byte that is transferred. ICS dictates whether the processor simply enters T3 or first enters Tw to wait.

After all data is transferred, the processor enters either Tv or Tvo. Tvo can be entered only when the processor is prepared to accomplish an immediate write transfer (overlapped access). During Tvo, the ACD lines contain address and specification information aligned in the same fashion as T1. If the processor does not require an overlapped access, the bus state move to Tv (the ACD lines will be high impedance). After Tv, a new bus cycle can be initiated with T1, or the processor may enter the idle state (Ti).

Table 8. ACD Specification Encoding

ACD 15	ACD 14	ACD 13	ACD 12	ACD 11	ACD 10	ACD 9	ACD 8
Access	**Op**	**RMW**	**Length**			**Modifiers**	
0– Memory	0– Read	0– Normal	000– 1 Byte 001– 2 Bytes 010– 4 Bytes 011– 6 Bytes 100– 8 Bytes			ACD15 = 0: 00–Inst Seg Access 01–Stack Seg Access	
1– Interconnect Space	1– Write	1– RMW	101–10 Bytes 110–16 Bytes* 111–32 Bytes* *Not implemented			10–Context Ctl Seg Access 11–Other ACD15 = 1: 00–Reserved 01–Reserved 10–Reserved 11–Interconn Register	

Interconnect Status (ICS)

As discussed earlier, ICS has three possible interpretations depending on the current state of the bus transaction (see Table 9). Even so, under most conditions ICS indicates whether or not an IPC is pending; a valid low during any of these cycles with IPC significance signal the processor that an IPC has been received. While an iAPX 432 processor is only required to record and service one IPC or reconfiguration request at a time, logic in the interconnect system must record and sequence multiple (and possibly simultaneous) IPC occurrences and reconfiguration requests. Thus, the logic that implements ICS must accommodate global and local IPC arrivals and requests for reconfiguration as individual events:

Table 9. ICS Interpretation

State	Significance	Level	
		High	Low
Ti, T1, T2	IPC	No IPC Waiting	IPC Waiting
T3, Tw	Stretch	Don't Stretch	Stretch
Tv, Tvo	Err	Bus Error	No Error

1. Assert IPC significance on ICS for the arrival of an IPC or reconfiguration request.

2. When the iAPX 432 processor reads interconnect address register 2, it will respond to one of the status bits for the IPC or reconfiguration request signalled on ICS in the following order:

   BIT 2 (1 = reconfigure, 0 = do not reconfigure)

   BIT 1 (1 = global IPC pending, 0 = no global IPC)

   BIT 0 (1 = local IPC pending, 0 = no local IPC)

3. The logic in the interconnect system must clear the highest order status bit that was serviced by the iAPX 432 processor, and if an additional IPC message has arrived, the interconnect logic must signal an additional IPC to the processor by setting ICS high for at least one cycle and then setting ICS low for at least one cycle, while ICS has IPC significance.

Processor Packet Bus Request (PRQ)

PRQ is normally low and goes high only during T1, T2, and Tvo. High levels during Tvo and T1 indicate the first cycle of an access. A high level during T2 indicates that the current cycle is to be cancelled. See Table 10.

Table 10. PRQ Interpretation

State	PRQ	Condition
Ti	0	Always
T1	1	Initiate access
T2	0	Continue access
	1	Cancel access
T3	0	Always
Tw	0	Always
Tv	0	Always
Tvo	1	Initiate overlapped access

Enable Buffers for Output (B_{OUT})

B_{OUT} is provided to control external buffers when they are present. Table 11 and Figures 22 through 27 show its state under various conditions.

Processor Packet Bus Timing

Each timing diagram shown on the following pages illustrates the timing relationships on the Processor Packet bus during various types of transactions. This approach to transfer timing allows maximum time for the transfer to occur and yet guarantees hold time.

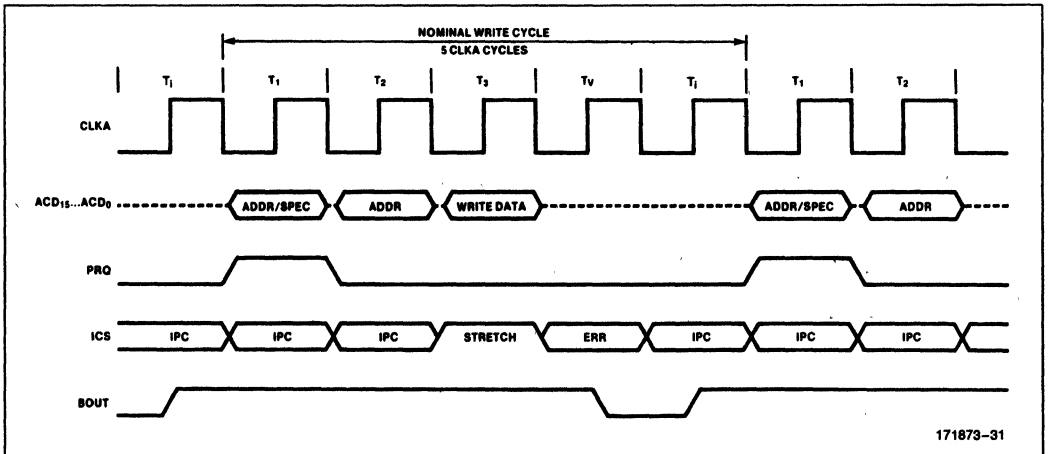
Any agent connected to the Processor Packet bus is recognized as either a processor (a GDP or IP) or a slave (e.g., the memory subsystem).

In all transfers between a processor and a slave, the data to be driven is clocked for three-quarters of a cycle before it is sampled. This allows adequate time for the transfer and ensures sufficient hold time after sampling. The B_{OUT} timing is unique because B_{OUT} functions as a direction control for external buffers.

Detailed set-up and hold times can be found in the AC Characteristics section.

Table 11. B_{OUT} Interpretation

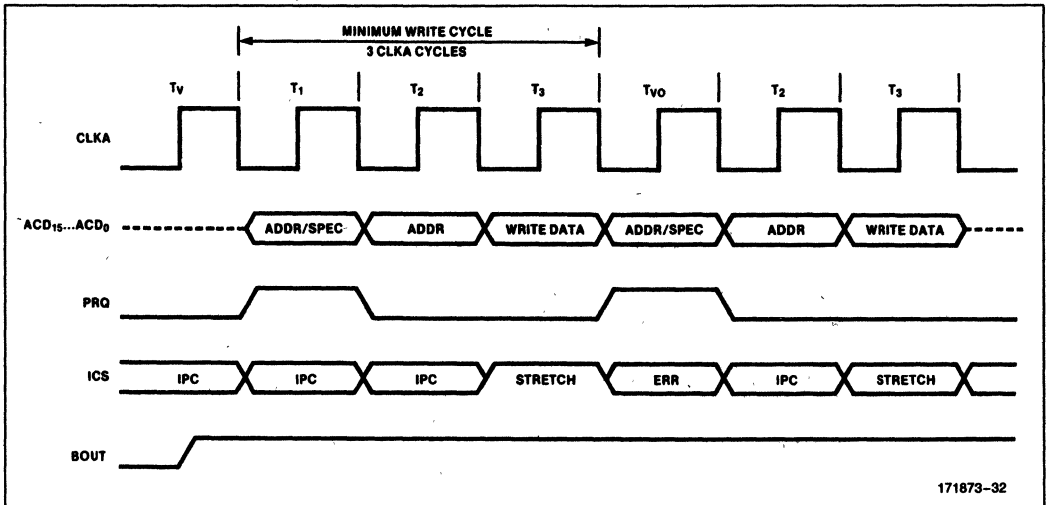
B_{OUT}	Always High	Low-to-High Transition or Low	High-to-Low Transition or Low	High-to-Low Transition or High
Write	T1, T2, T3, Tw, Tvo	Ti	None	Tv
Read	T1, T2	Ti, Tv	T3, Tw	None



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data 1*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2

*Undefined if single byte write

Figure 22. Nominal Write Cycle Timing

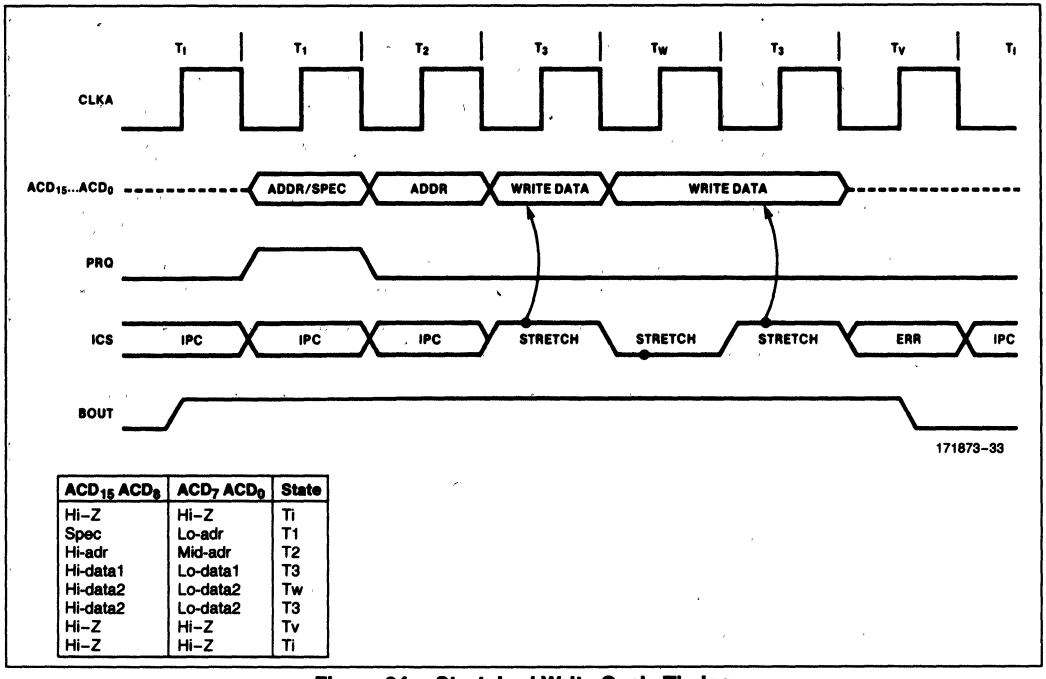


ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Spec	Lo-adr	Tvo
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3

*Undefined if single byte write
**(Preceded by read cycle)

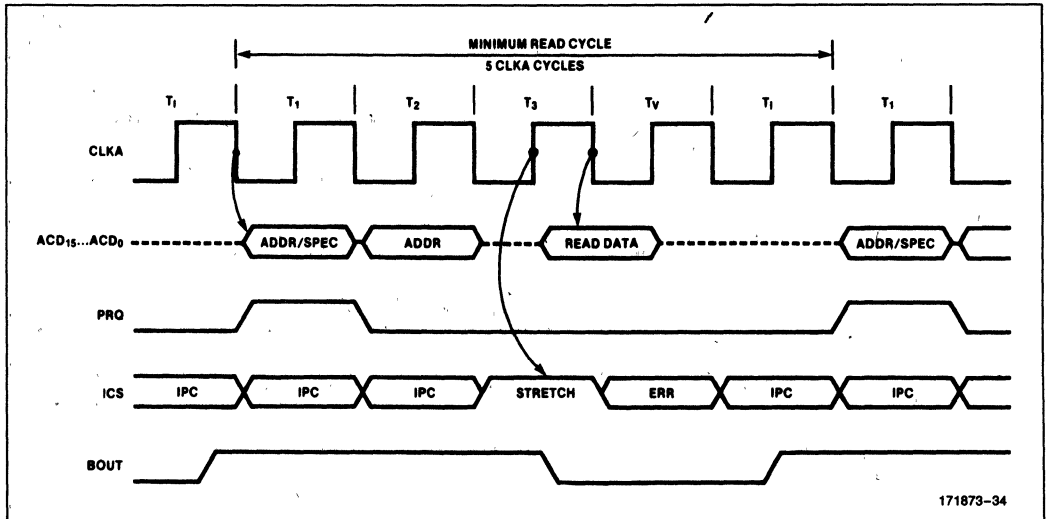
Figure 23. Minimum Write Cycle Timing

171873-002



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3
Hi-data2	Lo-data2	Tw
Hi-data2	Lo-data2	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

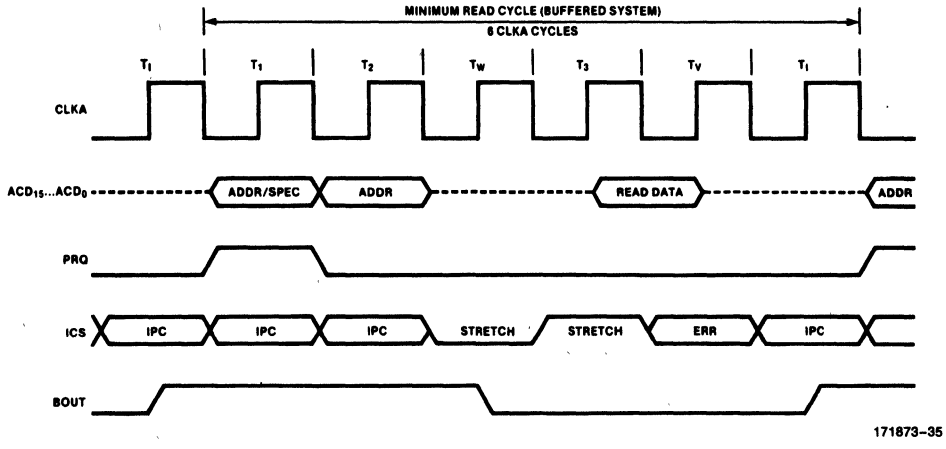
Figure 24. Stretched Write Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1

*Undefined if single byte read

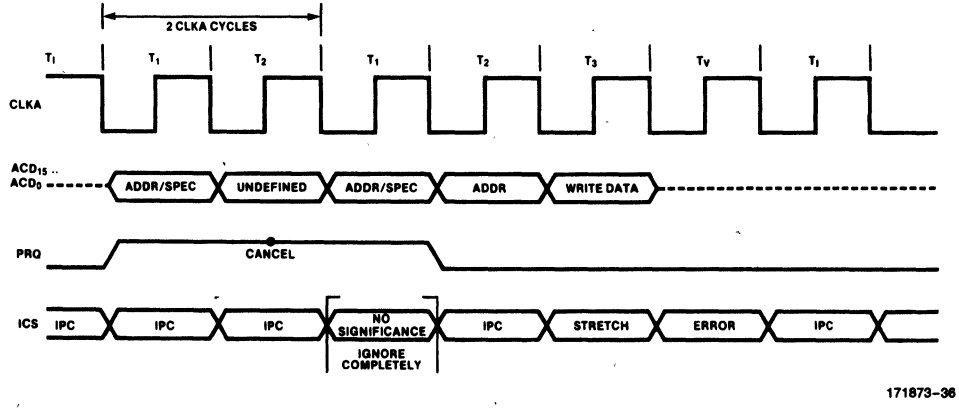
Figure 25. Minimum Read Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-Z	Hi-Z	Tw
Hi-data1*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

*Undefined if single byte read

Figure 26. Stretched Read Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Undefined	Undefined	T2**
Spec	Lo-adr	T†
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

*Undefined if single byte write
**Access Cancelled
†New Access Started (Slave must support this subsequent access even though all processors may not implement it.)

Figure 27. Minimum Faulted Access Cycle

171873-002

Package

The iAPX 43201 and iAPX 43202 are both packaged in a 68-pin, leadless JEDEC type A hermetic chip carrier. Figure 28 illustrates the package, and Figures 1 and 2 show the pinouts.

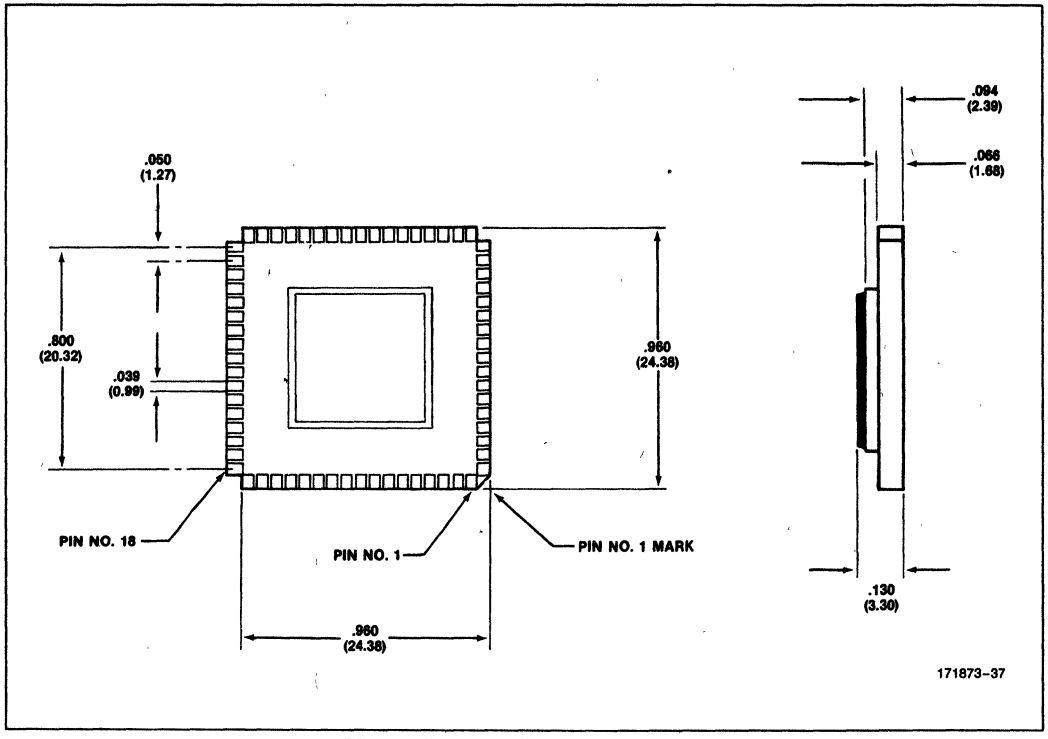


Figure 28. JEDEC Type A Package

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground .....−1 to + 7V
Power Dissipation.....2.5W

**Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC ELECTRICAL CHARACTERISTICS (V_{SS} = 0V, V_{CC} = 5V ± 10%)

Symbol	Parameter	Min	Max	Units
V_{ILC}	Input Low Voltage Clocks	−0.3	+0.5	V
V_{IHC}*	Input High Voltage Clocks	3.5	V_{CC}+0.5	V
V_{ILI}	Input Low Voltage Intra-GDP Bus	−0.3	0.7	V
V_{IHI}	Input High Voltage Intra-GDP Bus	3.0	V_{CC}+0.5	V
V_{IL}	Input Low Voltage	−0.3	0.8	V
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V
V_{OLI}	Output Low Voltage Intra-GDP Bus (I_{OLI} = 0.1 mA)	—	0.35	V
V_{OHI}	Output High Voltage Intra-GDP Bus (I_{OHI} = 0.1 mA)	3.25	V_{CC}	V
V_{OL}	Output Low Voltage (I_{OL}** = 2.0 mA)	—	0.45	V
V_{OH}	Output High Voltage (I_{OH} = −400 μA: 43201 −800 μA: 43202)	2.4	V_{CC}	V
I_{CC}	Power Supply Current (sum of all V_{CC} pins)	—	500	mA
I_{LI}	Input Leakage Current	—	±10	μA
O_{LI}	Output Leakage Current	—	±10	μA

*For operation at 5 MHz or slower, the GDP may be operated with V_{IHC} minimum of 2.7V.
**I_{OL} for $\overline{HERR}$ = 0.4 mA; for $\overline{FATAL}$ = 4 mA

171873-002

iAPX 43201 AC CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70°C)

Symbol	Description	5 MHz		7 MHz		8 MHz		Units
		Min	Max	Min	Max	Min	Max	
t_{CY}	Clock Cycle Time	200	500	143	500	125	500	ns
t_r, t_f	Clock Rise and Fall Times	0	10	0	10	0	10	ns
t1, t2, t3, t4	Clock Edge Delay Times	45	250	32	250	26	250	ns
t_{DC}	Input Signal to Clock Setup	5	—	5	—	5	—	ns
t_{DH}	Clock to Input Signal Hold Time	35	—	30	—	25	—	ns
t_{CD}	Clock to Output Signal Delay Time	—	85	—	65	—	55	ns
t_{OH}	Clock to Output Signal Hold Time	20	—	17	—	15	—	ns
T_{IE}	Input Enable Time	10	—	10	—	10	—	t_{CY}
t_{SI}	Input Signal to Init Setup Time	10	—	10	—	10	—	ns
t_{IS}	Init to Input Signal Hold Time	20	—	17	—	15	—	ns

The above specifications are subject to the following definitions and test conditions:

1. Note that tcy = t1 + t2 + t3 + t4 + 2*t_r + 2*t_f.

2. Pins under consideration were subjected to the following purely capacitive loading:
   C1 = 25pF on $\overline{HERR}$
   C1 = 50pF on ul15. . .ul10, IS6. . .IS0
   C1 = 70pF on all remaining pins.

3. All timings are measured with respect to the switching level of 1.5 Volts. The switching point of CLK_A and CLK_B is referenced to the 1.8 Volt Level.

4. CLK_A and CLK_B must be continuously applied for the 43201 to retain its state.

iAPX 43202 AC CHARACTERISTICS (V_{CC} = 5V ± 10%, T_A = 0°C to 70° C)

Symbol	Description	5 MHz		7 MHz		8 MHz		Units
		Min	Max	Min	Max	Min	Max	
t_{CY}	Clock Cycle Time ($t_{CY}=t_1+t_2+t_3+t_4+2t_r+2t_f$)	200	500	143	500	125	500	ns
t_r, t_f	Clock Rise and Fall Times	0	10	0	10	0	10	ns
t1, t2, t3, t4	Clock Pulse Widths	45	250	32	250	26	250	ns
t_{DC}	Signal to Clock Setup Time	5	—	5	—	5	—	ns
t_{DH}	Clock to Signal Hold Time	35	—	30	—	25	—	ns
t_{CD}	Clock to Signal Delay Time	—	85	—	65	—	55	ns
t_{OH}	Clock to Signal Output Time	20	—	17	—	15	—	ns
t_{DF}	Clock to Signal Data Float Time	—	75	—	75	—	55	ns

The timing characteristics given below assume the following loading on ouput pins. Loading is given in terms of a fixed capacitance plus a DC current load.

Pins	Loading
$\overline{HERR}$	90 pF Iol = 8 mA., Open Drain
B_{OUT}	70 pF Iol = 8 mA., Ioh = 800 μA
PRQ	70 pF Iol = 4 mA., Ioh = 800 μA
$IS_6 . . . IS_0$	50 pF MOS only
$ACD_{15} . . . ACD_0$	70 pF Iol = 4 mA., Ioh = 800 μA

All output delays are measured with respect to the falling edge of CLK_A except for B_{OUT}. B_{OUT} output delays are measured with respect to the rising edge of CLK_A.

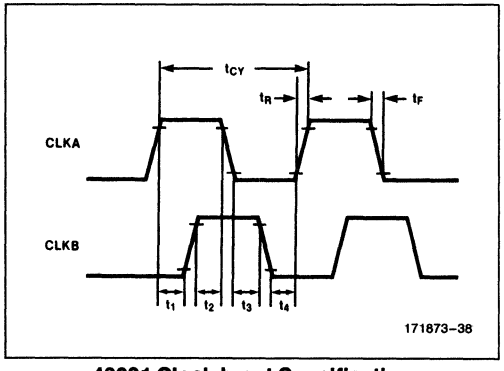
All timings are measured with respect to the switching level of 1.5 Volts. The switching point of CLK_A and CLK_B is referenced to the 1.8V level.

The 43202 is not capable of DC operation. For continuous data and logic state retention the CLK_A and CLK_B signals must be present.

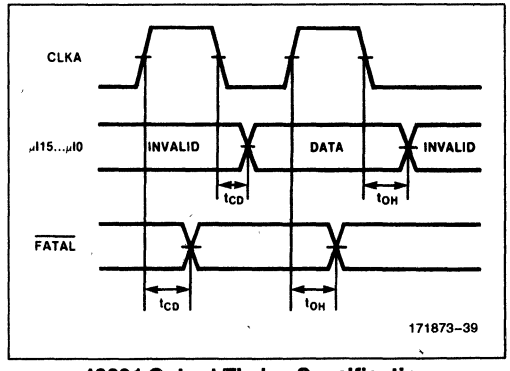
iAPX 43201/43202 Capacitance

Symbol	Parameter	Typical	Unit
C_{IN}	Input Capacitance	6	pF
C_{OUT}	Output Capacitance	12	pF
Conditions: fc = 1 MHz, V_{IN} = 0V, V_{CC} = 5V, T_A = 25°C Outputs in High Impedance State			

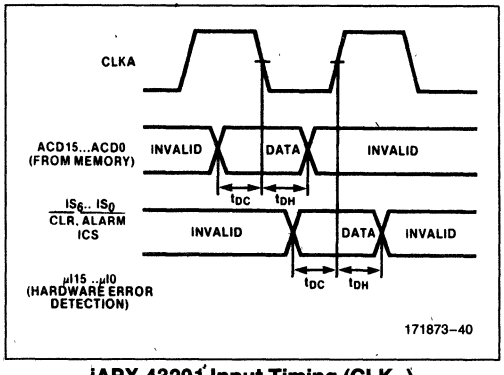
WAVEFORMS:



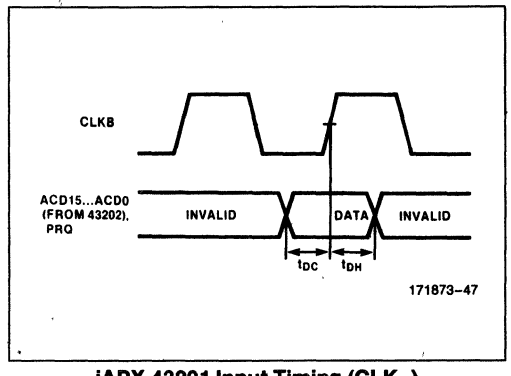
43201 Clock Input Specification



43201 Output Timing Specification

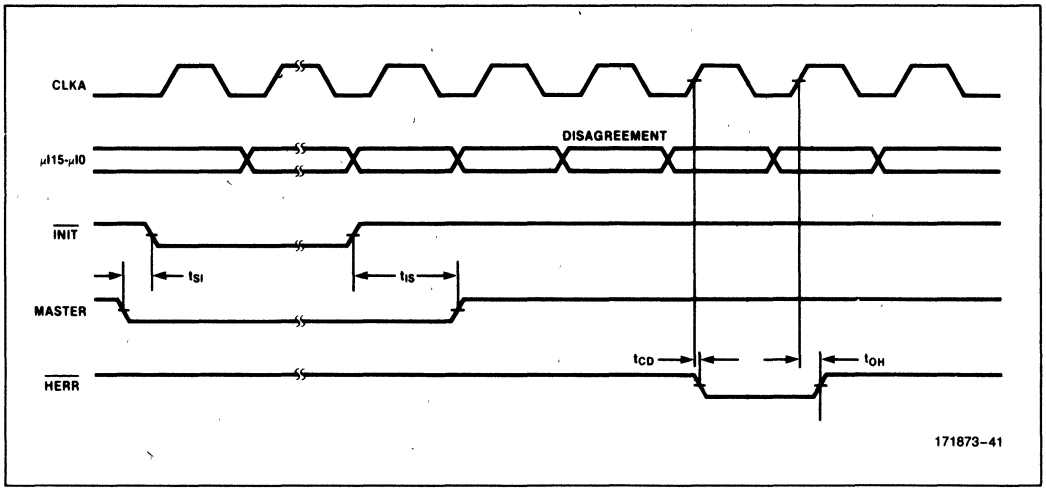


iAPX 43201 Input Timing (CLK_A)

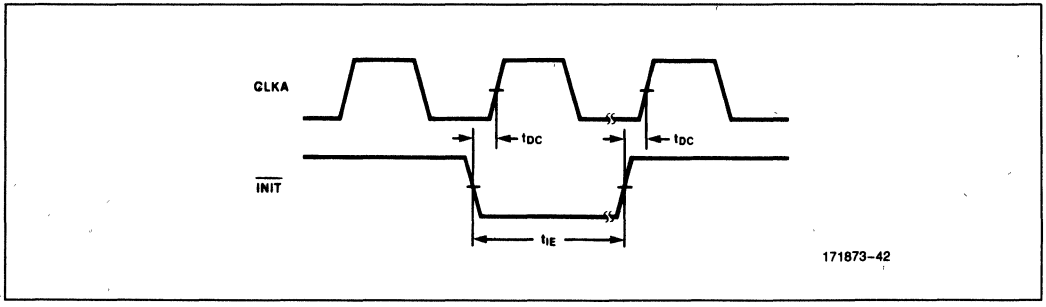


iAPX 43201 Input Timing (CLK_B)

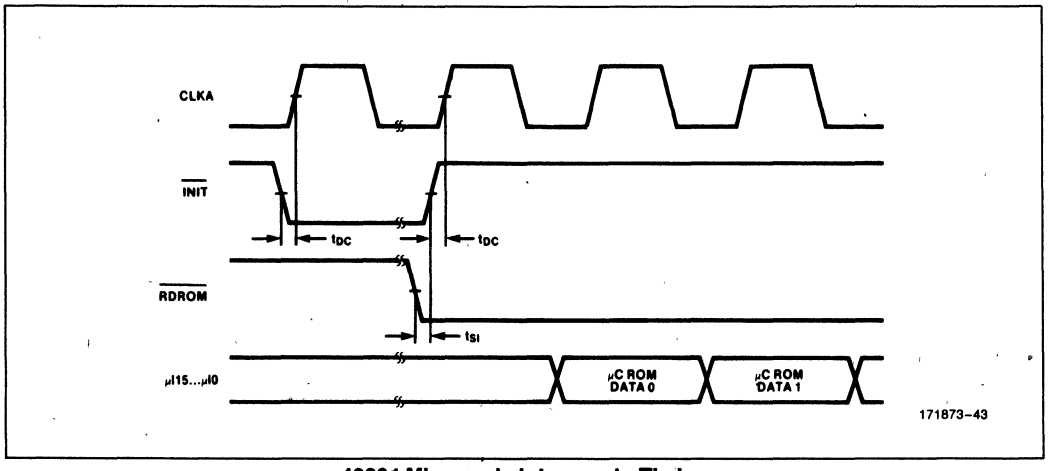
WAVEFORMS (Continued):



43201 Hardware Error Detection Timing

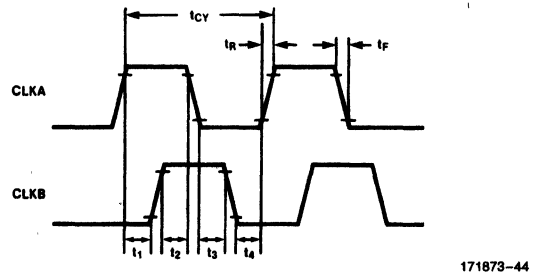


43201 Initialization Timing

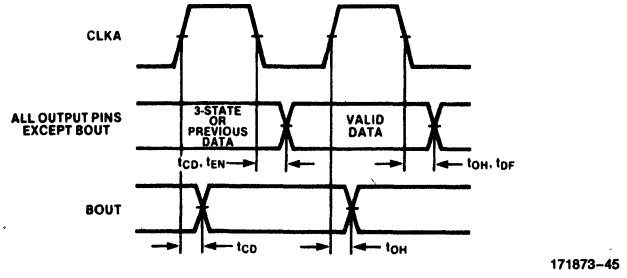


43201 Microcode Interrogate Timing

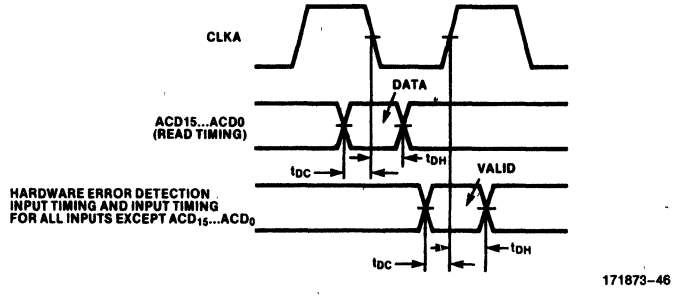
WAVEFORMS (Continued):



43202 Clock Input Specification



43202 Output Timing Specification



43202 Input Timing Specification

iAPX 432. General Data Processor Operator Set Summary

Character Operators	Short-Integer Operators	Integer Operators
Move Character	Move Short Integer	Move Integer
Zero Character	Zero Short Integer	Zero Integer
One Character	One Short Integer	One Integer
Save Character	Save Short Integer	Save Integer
AND Character	Add Short Integer	Add Integer
OR Character	Subtract Short Integer	Subtract Integer
XOR Character	Increment Short Integer	Increment Integer
XNOR Character	Decrement Short Integer	Decrement Integer
Complement Character	Negate Short Integer	Negate Integer
	Multiply Short Integer	Multiply Integer
Add Character	Divide Short Integer	Divide Integer
Subtract Character	Remainder Short Integer	Remainder Integer
Increment Character		
Decrement Character	Equal Short Integer	Equal Integer
	Not Equal Short Integer	Not Equal Integer
Equal Character	Equal Zero Short Integer	Equal Zero Integer
Not Equal Character	Not Equal Zero Short Integer	Not Equal Zero Integer
Equal Zero Character		Less Than Integer
Not Equal Zero Character	Less Than Short Integer	Less Than or Equal Integer
Less Than Character	Less Than or Equal Short Integer	Positive Integer
Less Than or Equal Character	Positive Short Integer	Negative Integer
Convert Character to Short Ordinal	Negative Short Integer	Move in Range Integer
	Move in Range Short Integer	Convert Integer to Short Integer
	Convert Short Integer to Integer	Convert Integer to Ordinal
		Convert Integer to Temporary Real
		Convert Integer to Character
		Convert Integer to Short Ordinal

Short-Ordinal Operators	Ordinal Operators	Short-Real Operators
Move Short Ordinal	Move Ordinal	Move Short Real
Zero Short Ordinal	Zero Ordinal	Zero Short Real
One Short Ordinal	One Ordinal	Save Short Real
Save Short Ordinal	Save Ordinal	
		Add Short Real—Short Real
AND Short Ordinal	AND Ordinal	Add Short Real—Temporary Real
OR Short Ordinal	OR Ordinal	Add Temporary Real—Short Real
XOR Short Ordinal	XOR Ordinal	Subtract Short Real—Short Real
XNOR Short Ordinal	XNOR Ordinal	Subtract Short Real—Temporary Real
Complement Short Ordinal	Complement Ordinal	Subtract Temporary Real—Short Real
		Multiply Short Real—Short Real
Extract Short Ordinal	Extract Ordinal	Multiply Short Real—Temporary Real
Insert Short Ordinal	Insert Ordinal	Multiply Temporary Real—Short Real
Significant Bit Short Ordinal	Significant Bit Ordinal	Divide Short Real—Short Real
		Divide Short Real—Temporary Real
Add Short Ordinal	Add Ordinal	Divide Temporary Real—Short Real
Subtract Short Ordinal	Subtract Ordinal	Negate Short Real
Increment Short Ordinal	Increment Ordinal	Absolute Value Short Real
Decrement Short Ordinal	Decrement Ordinal	
Multiply Short Ordinal	Multiply Ordinal	
Divide Short Ordinal	Divide Ordinal	
Remainder Short Ordinal	Remainder Ordinal	
	Index Ordinal	
Equal Short Ordinal	Equal Ordinal	
Not Equal Short Ordinal	Not Equal Ordinal	
Equal Zero Short Ordinal	Equal Zero Ordinal	
Not Equal Zero Short Ordinal	Not Equal Zero Ordinal	
Greater Than Short Ordinal	Less Than Ordinal	
Greater Than or Equal Short Ordinal	Less Than or Equal Ordinal	
Convert Short Ordinal to Integer	Convert Ordinal to Integer	
	Convert Ordinal to Temporary Real	

iAPX 432. General Data Processor Operator Set Summary (Continued)

Short-Real Operators	Real Operators	Temporary-Real Operators
Equal Short Real Equal Zero Short Real Less Than Short Real Less Than or Equal Short Real Positive Short Real Negative Short Real Convert Short Real to Temporary Real	Move Real Zero Real Save Real Add Real—Real Add Real—Temporary Real Add Temporary Real—Real Subtract Real—Real Subtract Real—Temporary Real Subtract Temporary Real—Real Multiply Real—Real Multiply Real—Temporary Real Multiply Temporary Real—Real Divide Real—Real Divide Real—Temporary Real Divide Temporary Real—Real Negate Real Absolute Value Real Equal Real Equal Zero Real Less Than Real Less Than or Equal Real Positive Real Negative Real Convert Real to Temporary Real	Move Temporary Real Zero Temporary Real Save Temporary Real Add Temporary Real Subtract Temporary Real Multiply Temporary Real Divide Temporary Real Remainder Temporary Real Negate Temporary Real Square Root Temporary Real Absolute Value Temporary Real Equal Temporary Real Equal Zero Temporary Real Greater Than Temporary Real Greater Than or Equal Temporary Real Positive Temporary Real Negative Temporary Real Convert Temporary Real to Ordinal Convert Temporary Real to Integer Convert Temporary Real to Short Real Convert Temporary Real to Real
Access Descriptor Movement Operators	**Type and Rights Manipulation Operators**	
Copy Access Descriptor Null Access Descriptor	Amplify Rights Restrict Rights Retrieve Type Definition	
Refinement Operators	**Object Creation Operators**	**Access Path Inspection Operators**
Create Generic Refinement Create Typed Refinement	Create Object Create Typed Object	Inspect Access Descriptor Inspect Object Equal Access Move to Embedded Data Value Move from Embedded Data Value
Access Interlock Operators	**Branch Operators**	**Interconnect Operators**
Lock Object Unlock Object Indivisibly Add Short Ordinal Indivisibly Add Ordinal Indivisibly Insert Short Ordinal Indivisibly Insert Ordinal	Branch Branch True Branch False Branch Indirect Branch Intersegment Branch Intersegment without Trace Branch Intersegment and Link Breakpoint	Move to Interconnect Move from Interconnect
Process Communication Operators	**Processor Communication Operators**	**Context Operators**
Send Receive Conditional Send Conditional Receive Surrogate Send Surrogate Receive Delay Process Read Process Clock Send Process Set Process Mode	Read Processor Status and Clock Send to Processor	Enter Environment Copy Process Globals Set Context Mode Call Call through Domain Return Return and Fault Adjust Stack Pointer Block Move

Additional Information

More information about the iAPX 432 Micromainframe architecture can be found in the following publications:

- iAPX 432 General Data Processor Architecture Reference Manual (Order Number 171860)
- iAPX 432 Interface Processor Architecture Reference Manual (Order Number 171863)
- iAPX 432 Interconnect Architecture Reference Manual (Order Number 172487)

Information on the electrical characteristics of other 432 components can be found in the following publications:

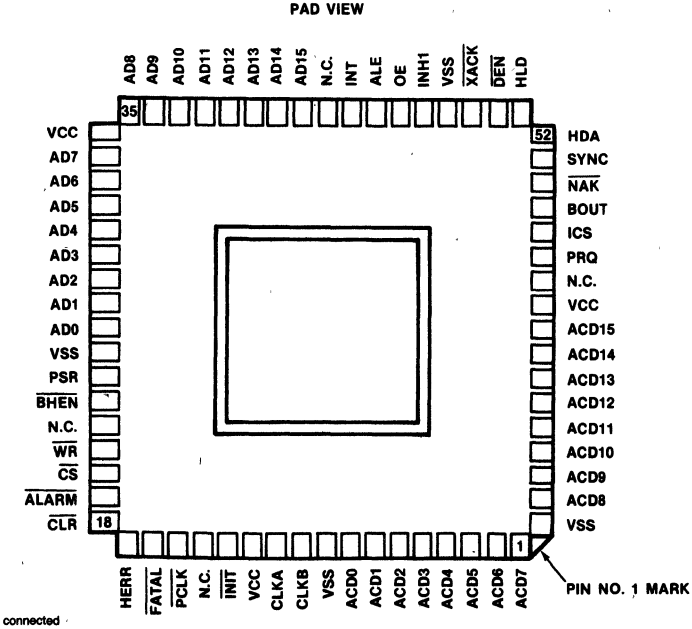
- iAPX 43203 Interface Processor Data Sheet (Order Number 171874)
- iAPX 43204/43205 Fault Tolerant Bus Interface and Memory Control Units (Order Number 210963)
- iAPX 43204/43205 BIU/MCU Electrical Specifications (Order Number 172867)

iAPX 43203
FAULT TOLERANT INTERFACE PROCESSOR

- **Multiprocessor Architecture Offers Fully Independent I/O**
- **High-Speed Data Channel Buffers Burst-Mode Transfers**
- **Software-Controlled Windows Provide Protected Access to 432 Memory**
- **16-Bit Data Bus Interfaces Easily to MULTIBUS® Systems**
- **Multiple Interface Processors Expand I/O Capacity**
- **Master/Checker Pairs Detect Hardware Errors Automatically**
- **Quad Modular Redundancy Ensures Immediate Recovery From Hardware Faults**

The Intel 43203 Interface Processor (IP) provides an independent and decentralized I/O channel for iAPX 432 Micromainframe systems by mapping a portion of a peripheral subsystem's address space onto central system memory. The 43203 IP can be used with the other members of the iAPX component family (i.e., the 43201/43202 General Data Processor, the 43204 Bus Interface Unit, and the 43205 Memory Control Unit) to design a completely fault-tolerant computer system.

The 43203 is a VLSI device, fabricated with Intel's highly reliable +5 volt, depletion load, N-channel, silicon gate HMOS technology, and is packaged in a 68-pin, leadless JEDEC hermetic chip carrier. Refer to Figure 1 for the JEDEC chip carrier representation of the 43203 pin configuration.



NOTE: N.C. pads must not be connected

171874-1

Figure 1. iAPX 43203 Interface Processor Pin Configuration

November 1983
Order Number: 171874-002

Table 1 lists a summary of all signal groups, signal names and their active states, and whether or not they are monitored by the Hardware Error Detection circuitry.

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
Processor Packet Bus Group			
ACD_{15} ACD_0	1–8 60–69	I/O	**Address/Control/Data Lines:** These 16 bidirectional signals carry physical memory addresses, control information (access length and type), and data to and from the memory bus. When the IP is in checker mode, the ACD pins are monitored by the hardware error detection logic and are in the high impedance state.
PRQ	57	O	**Processor Packet Bus Request:** is issued to gain access to the bus. Normally low, the PRQ pin is brought high during the same cycle as the first double-byte of address information appears on the ACD pins. PRQ remains high for only one cycle during the access, unless an address development fault occurs. In that case, the IP leaves PRQ high for a second cycle to indicate it has detected an addressing or segments rights fault in completing the address generation. PRQ is checked by the hardware detection logic and remains in a high impedance mode when the IP is in checker mode.
ICS	56	I	**Interconnect Status:** carries information on errors, data synchronization, and interprocessor communication. The interpretation of this signal depends on the current cycle of the bus transaction. See page 21 for a complete description.
B_{OUT}	55	O	**Enable Buffers for Output:** controls the direction of external buffers, if any are used. When BOUT is asserted, it indicates that the buffers must be directed to carry information outbound from the IP.
System Group			
V_{CC}	12, 34, 59		**Power:** These three pins supply 5-volt power to the IP, and all must be connected; the pins are not connected together within the IP.
V_{SS}	9, 48, 68		**Ground:** These three pins provide the ground reference for the IP, and all must be connected; the pins are not connected together within the IP.
$\overline{ALARM}$	19	I	**$\overline{Alarm}$:** monitors the condition of an unusual, system-wide condition such as power failure. Alarm is sampled on the rising edge of CLKA.
$\overline{FATAL}$	16	O I	**$\overline{Fatal}$:** is asserted by the IP under microcode control when the IP is unable to continue due to various error or fault conditions. Once $\overline{FATAL}$ is asserted, it can only be reset by the assertion of $\overline{INIT}$. When $\overline{INIT}$ is asserted, the $\overline{FATAL}$ pin functions as an input. During initialization, the IP samples the state of $\overline{FATAL}$ to determine if the 432 system side should be placed in MASTER or CHECKER mode. See $\overline{INIT}$ description.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
System Group (Continued)			
$\overline{PCLK}$	15	I	**Processor Clock:** The assertion of $\overline{PCLK}$ for one cycle causes the system timer within the IP to decrement. Assertion of $\overline{PCLK}$ for two or more cycles causes the system timer to be reset. $\overline{PCLK}$ must remain unasserted for at least 10 clock cycles before being asserted again. The IP samples $\overline{PCLK}$ on the rising edge of CLK_A.
$\overline{CLR}$	18	I	**Clear:** Assertion of $\overline{CLR}$ results in a microprogram trap causing the IP to immediately terminate any bus transactions or internal operations in progress. The IP resets to a known state, asserts $\overline{FATAL}$, and awaits an IPC for initialization. The IPC is not serviced for at least four clock cycles following the assertion of $\overline{CLR}$. Response to $\overline{CLR}$ is disabled by the first $\overline{CLR}$ assertion and is reenabled when the IP receives the first IPC (or $\overline{INIT}$ assertion). The IP samples $\overline{CLR}$ on the rising edge of CLK_A.
$\overline{INIT}$	13	I	**Initialize:** Assertion of $\overline{INIT}$ resets the internal state of the IP and starts execution of the initialization microcode. $\overline{INIT}$ must be asserted for a minimum of 8 clock cycles. After the $\overline{INIT}$ pin returns to its nonasserted state, the IP initializes all of its internal registers and windows, and waits for a local IPC. $\overline{INIT}$ is sampled on the rising edge of CLK_A. During $\overline{INIT}$ assertion, the IP samples the $\overline{FATAL}$ and HERR pins to establish the mode (MASTER or CHECKER) for each of the bus interfaces to the IP. See accompanying table.
			Representation of MASTER/CHECKER Modes at Initialization
			$\overline{FATAL}$ / HERR / iAPX 432 Side / Peripheral Subsystem Side
			0 / 0 / MASTER / MASTER 0 / 1 / MASTER / CHECKER 1 / 0 / CHECKER / MASTER 1 / 1 / CHECKER / CHECKER
HERR	17	O	**Hardware Error:** This line is used to signal a discrepancy between a value internally computed by a checker and that output by the master. The sampling for errors occurs at the most appropriate time for each of the pins being checked. HERR is an open drain output and requires an external pullup resistor. Nominally, the output is held low, but released upon the detection of a discrepancy. The timing of HERR depends on the source of the error. Once HERR is high, it remains high until external logic forces it low again. When HERR goes low, the present error condition is cleared and HERR is immediately capable of detecting and signalling another error.
		I	When $\overline{INIT}$ is asserted, the HERR pins becomes an input. During initialization, the IP samples HERR to establish the mode (MASTER or CHECKER) of the bus interface to the peripheral subsystem. See the discussion of $\overline{INIT}$.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
System Group (Continued)			
CLK_A	11	I	**Clock A:** is a square-wave clock which must operate continuously to preserve the operating state of the IP.
CLK_B	10	I	**Clock B:** is a square-wave clock which operates at the same frequency as CLK_A, but lags it by 90 degrees. CLK_B must operate continuously to preserve the operating state of the IP.
Peripheral Subsystem Bus Group			
AD_{15}-AD_0	26-42	I/O	**Address/Data:** These pins constitute a multiplexed address and data input/output bus. When the Attached Processor bus is idle or during the first part of an access, these pins normally view the bus as an address. The address is checked asynchronously to see if it matches any one of the five window address ranges. The address is latched on the falling edge of ALE, thereby maintaining the state of match or no match for the remainder of the access cycle. The addresses are unlatched on the falling edge of OE. Once SYNC has pulsed high, the AD_{15}-AD_0 pins become data input and output pins. When $\overline{WR}$ is high (read mode) and OE is asserted, data is accessed in the IP and the output buffers are enabled onto the AD pins. When $\overline{WR}$ is low (write mode), data is sampled by the IP after the rising edge of SYNC and while CLK_A is high. The address is always a 16-bit unsigned number. Data may be either 8 or 16 bits as defined by $\overline{BHEN}$ and AD_0. The 8-bit data may be transferred on either the high (AD_{15}-AD_8) or the low (AD_0-AD_7) byte, while the opposite byte is tristated. During the clock in which write data is sampled, data must be set up before the rising edge of CLK_A and must be held until the falling edge of that clock cycle. Read data is driven out from a CLK_A high and should be sampled on the next rising edge of CLK_A. Hardware error detection is not done synchronously to CLK_A; rather, it is sampled on the falling edge of OE. The internal AD pin hardware error detection signal is then clocked and output as HERR. Even at this point, it may not be synchronous with CLKA and so should be externally synchronized.
$\overline{BHEN}$	23	I	**Byte High Enable:** This pin, together with AD_0, determines whether 8 or 16 bits are to be accessed, and if it is 8 bits, whether it is to be accessed on the upper or lower byte position. $\overline{BHEN}$ is latched by the falling edge of ALE and unlatched by the falling edge of OE. See accompanying table for decoding.

Bus Data Controls

$\overline{BHEN}$	AD_0	Description
0	0	16-bit access
0	1	8 bits on upper byte, lower byte tristated
1	0	8 bits on lower byte, upper byte tristated
1	1	8 bits on lower byte, upper byte tristated

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
Peripheral Subsystem Bus Group (Continued)			
$\overline{CS}$	20	I	**Chip Select:** specifies that this IP is selected and that a read or write cycle is requested. $\overline{CS}$ is latched by the falling edge of ALE and unlatched by the falling edge of OE.
$\overline{WR}$	21	I	**Write:** specifies whether the access is to be a read or a write. $\overline{WR}$ is asserted high for a read and low for a write. The pin is latched by the falling edge of ALE and unlatched by the falling edge of OE.
PS Timing Group			
ALE	45	I	**Address Latch Enable:** The rising edge of ALE sets a flip-flop that enables $\overline{XACK}$ to become active. The falling edge of ALE latches the address on the AD_{15}-AD_0 pins and latches $\overline{WR}$, $\overline{BHEN}$, and $\overline{CS}$.
OE	46	I	**Data Output Enable:** During a read cycle OE enables read data on the AD_{15}-AD_0 pins when it is asserted. The falling edge of OE signifies the end of an access cycle (for either a read or a write) by: 1. Resetting the $\overline{XACK}$ enable flip-flop, thereby terminating $\overline{XACK}$. 2. Terminating $\overline{DEN}$ (if a read cycle). 3. Opening address latches $\overline{WR}$, $\overline{BHEN}$, and $\overline{CS}$.
SYNC	53	I	**Synchronized Qualifier Signal:** A rising edge on this signal must be synchronized to the IP CLK_A falling edge. SYNC qualifies the address, $\overline{BHEN}$, $\overline{CS}$, and $\overline{WR}$, indicating a valid condition. SYNC also initiates any internal IP action to process an access. In a read access, SYNC starts the request for data to the IP, and in a write access, data is expected one or two CLK_A cycles after SYNC pulses high. At initialization time, IP microcode sets the write sample delay to the slowest operation, two CLK_A cycles after SYNC, but this can be changed to one clock cycle by making a function request to the IP to change the write sample delay. When the hold/hold-acknowledge mechanism of the IP is used, and once HDA has pulsed high, a SYNC pulse is required to qualify the hold acknowledge, since the HDA pin can be asynchronous.
PS Synchronization Group			
$\overline{XACK}$	49	O	**Transfer Acknowledge:** This signal is used to acknowledge that a data transfer has taken place. For random or local accesses, $\overline{XACK}$ indicates that a transfer to or from 432 system memory has been completed. For buffered accesses where the XACK-Delay is not in the advanced mode, $\overline{XACK}$ signifies that the transfer from/to the prefetch/postwrite buffer in the IP has been completed. For buffered accesses which use advanced acknowledge mode (XD = 0) the formation of an advanced $\overline{XACK}$ signal is requested. This enables the IP to interface with a peripheral subsystem without wait states. The acknowledge will be advanced if the access is a read operation and the buffer contains the required data, or the access is a write operation and the buffer contains sufficient space to accept the write data. Of course, the access must also be valid.

Table 1. Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td colspan="4">PS Synchronization Group (Continued)</td></tr>
<tr><td></td><td></td><td></td><td>If $\overline{XACK}$ is preceded by a low pulse on $\overline{NAK}$, then $\overline{XACK}$ signifies that the access encountered a fault. If the access was a random access, other than window 4, the window is placed in a faulted state and any further attempts to access the window are ignored by the IP.

If the IP is programmed to be in advanced acknowledge mode (XD = 0) and $\overline{XACK}$ is not returned before the peripheral subsystem issued SYNC, then $\overline{XACK}$ will be postponed until valid data has been established on the AD_{15}-AD_0 bus.

Five conditions affecting $\overline{XACK}$ behavior are:

1. XACK-Delay, user-programmable through an IP function request. This parameter establishes the minimum operating XACK-delay with respect to the SYNC signal. See accompanying table.

2. XACK-enable-flip-flop, set by the rising edge of the ALE signal and reset by the falling edge of the OE signal.

3. Internal IP registers. These are used to determine validity of the peripheral subsystem access and establish access modes.

4. Type of access behavior: Random or Buffered, Memory or Interconnect.

5. Bus faults, nonexistent memory, etc.

Hardware error detection occurs during the first clock of SYNC assertion.

$\overline{XACK}$ Timing Parameters
<table>
<tr><th>Inhibit Mode</th><th>$\overline{WR}$</th><th>XD_1</th><th>XD_0</th><th>$\overline{XACK}$ Formation</th></tr>
<tr><td>0</td><td>X</td><td>0</td><td>0</td><td>Advanced Acknowledge ($\overline{XACK}$ can occur before SYNC).</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>Rising edge of SYNC</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>Rising edge of SYNC plus 1 Clock</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>Rising edge of SYNC plus 1 Clock</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>Rising edge of SYNC plus 2 Clocks</td></tr>
<tr><td>1</td><td>X</td><td>1</td><td>0</td><td>Rising edge of SYNC plus 2 Clocks</td></tr>
<tr><td>1</td><td>X</td><td>0</td><td>1</td><td>Rising edge of SYNC plus 2 Clocks</td></tr>
<tr><td>X</td><td>X</td><td>1</td><td>1</td><td>Illegal condition</td></tr>
</table>
Note: X = don't care condition</td></tr>
<tr><td>$\overline{NAK}$</td><td>54</td><td>O</td><td>Negative Acknowledge: This signal precedes $\overline{XACK}$ by at least one-half clock cycle, indicating that a transfer did not occur. $\overline{NAK}$ pulses low for only one clock period.

When the IP is in physical mode and making an interconnect access, negative acknowledge may be used to indicate that the access was made to a nonexistent interconnect address. This will allow a subsystem processor to determine the system configuration at system initialization.</td></tr>
</table>

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
PS Synchronization Group (Continued)			
			$\overline{NAK}$ can also be used to set a status bit and cause a special interrupt to transmit the information back to the system. $\overline{NAK}$ is driven synchronously from the falling edge of CLK_A. Hardware error detection occurs while CLK_A is high.
INH1	47	O	**Inhibit:** can be used to override other memories in the peripheral subsystem whose address space is overlapped by an IP window. INH1 is asserted asynchronously by nonclocked logic when a valid mappable address is detected. After initialization, the IP microcode sets the INH1 mode for each window by loading registers in the IP for each window. Once the subsystem is allowed to make a function request, it can selectively enable or disable the inhibit mode on each window. INH1 is gated off by $\overline{CS}$. The selection of inhibit mode for window 0, when in buffered mode, causes a built-in XACK-delay which delays the acknowledge from going active until two clock periods after the rising edge of SYNC. This was done to facilitate the use of most MULTIBUS systems using INH1, as they require that the acknowledge be delayed. When the Advanced $\overline{XACK}$ mode is programmed, the inhibit mode should not be used on window 0 when in buffered mode, since the acknowledge will not be effectively delayed. Hardware error detection occurs during the first clock of SYNC assertion.
PS Control Group			
$\overline{DEN}$	50	O	**Data Enable:** This signal enables the external data buffers used in systems where the address and data are not multiplexed (e.g., a MULTIBUS system). $\overline{DEN}$ assertion begins no sooner than the first clock of SYNC assertion while CLK_A is high, given that a valid, mappable address range has been detected. $\overline{DEN}$ is terminated either with the falling edge of OE or after $\overline{XACK}$ assertion. Hardware error detection occurs during the first clock of SYNC assertion.
HLD	51	O	**Hold Request:** The hold/hold-acknowledge mechanism is an interlocking mechanism between the peripheral subsystem and the IP. HLD is used by the IP to gain control of the subsystem bus to ensure that no subsystem processors will make an access to the IP while it alters internal registers. HLD is put out synchronously with the rising edge of CLK_A. Hardware error detection sampling occurs while CLK_A is low. In certain systems, it may not be necessary to use the HLD function interlocking. In those cases, HDA can be tied high and no SYNC pulse will be required for HDA qualification. The hardware detects this condition by noting that the HDA pin was high a half-clock before HLD requests a hold. In this mode, the HLD output still functions and can be monitored if desired.

Table 1. Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
PS Control Group (Continued)			
HDA	52	I	**Hold Acknowledge:** When the IP's request for a hold has been granted by the peripheral subsystem, HDA is asserted. The signal need only be a high pulse and can be asynchronous to CLK_A, but must be followed by a SYNC pulse in order to qualify it synchronously.
INT	44	O	**Interrupt:** This signal is used to interrupt the Attached Processor to request servicing. The output is a pulse two CLK_A's wide, and is driven synchronously from the rising edge of CLK_A. Hardware error detection occurs while CLK_A is low.
PSR	24	O	**Peripheral Subsystem Reset:** is asserted by the IP under microcode control. When asserted, the peripheral subsystem should be reset. When used for debugging, it may be desirable to use this pin to set a status bit in an external register or perhaps to cause a special interrupt. PSR is normally asserted by the IP when the peripheral subsystem is believed to be faulty and will not respond to other means of control. PSR is issued synchronously with the rising edge of CLK_A. Hardware error detection sampling occurs while CLK_A is low.

FUNCTIONAL DESCRIPTION

As its name implies, the 43203 Interface Processor is a logical and physical interface which links a peripheral subsystem to the iAPX 432 central system. (The internal architecture of the IP is illustrated in Figure 2.) The peripheral subsystem functions as an independent and decentralized I/O channel much in the same way as I/O channels in traditional mainframes.

The diagram in Figure 3 represents the IP as a logical device and illustrates the signal interface to the Processor Packet Bus (left side) and the peripheral subsystem (right side). The IP is connected to the peripheral subsystem bus so that it occupies a contiguous range of memory addresses, up to 64k bytes in length. A peripheral subsystem reference to one of these addresses is a reference to the IP, and

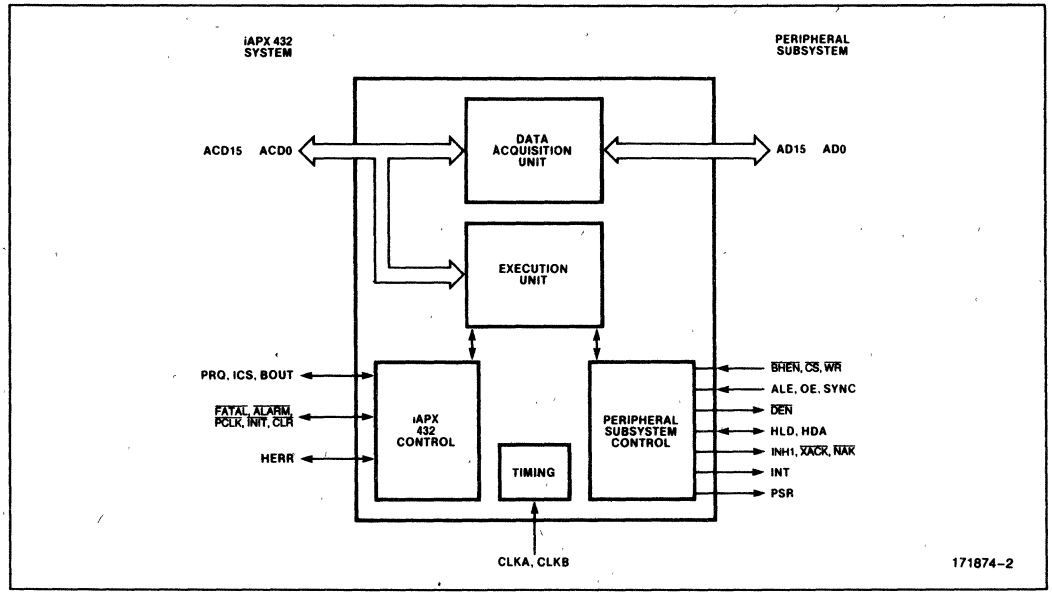


Figure 2. iAPX 43203 IP Functional Block Diagram

the IP responds to the address like a memory. The IP also provides an interrupt request line which is routed to its Attached Processor (AP) in the peripheral subsystem. The "other side" of the IP is connected to the central system in exactly the same manner as an iAPX 432 General Data Processor (GDP). By means of these connections, the IP links the peripheral subsystem physically to the central system; at the same time, it is positioned to monitor all data flow across the system boundary.

It is important to recognize that the IP is an **interface mechanism,** not an active device in the sense of a CPU. The IP does not fetch instructions; instead, it executes commands issued by AP software. Although physically connected only by a bus and interrupt line, the relationship of the IP and AP is very close. Indeed, it is often convenient to think of them as constituting a logical I/O processor.

Peripheral Subsystems

A computer system based on the iAPX 432 Micromainframe consists of a 432 central system and one or more peripheral subsystems. Figure 4A illustrates a hypothetical configuration, which employs two peripheral subsystems. The 432 system hardware is composed of one or more General Data Processors (GDPs), one or more Interface Processors (IPs), and a common memory shared by the processors. In most 432 systems, including all fault-tolerant systems, the processors and memory are interconnected through multiple iAPX 43204 Bus Interface Units (BIUs) and iAPX 43205 Memory Control Units (MCUs). (See Figure 4B.)

Software in a 432 system can be viewed as a collection of one or more **processes** that execute on the GDPs. A fundamental principle of the 432 architecture is that the 432 environment is self-contained; neither processors nor processes have any direct contact with the "outside world." In concept, the 432 system is enclosed by a wall that protects objects in memory from possible damage by uncontrolled I/O operations.

In a 432-based system, the bulk of processing required to support input/output operations is delegated to peripheral subsystems; this includes device control, timing, interrupt handling, and buffering. A peripheral subsystem is an autonomous computer system with its own memory, I/O devices and controllers, at least one processor, and software. The number of peripheral subsystems employed in any given application depends on how I/O-intensive the application is: the number may be varied as needs change and is independent of the number of GDPs in the system.

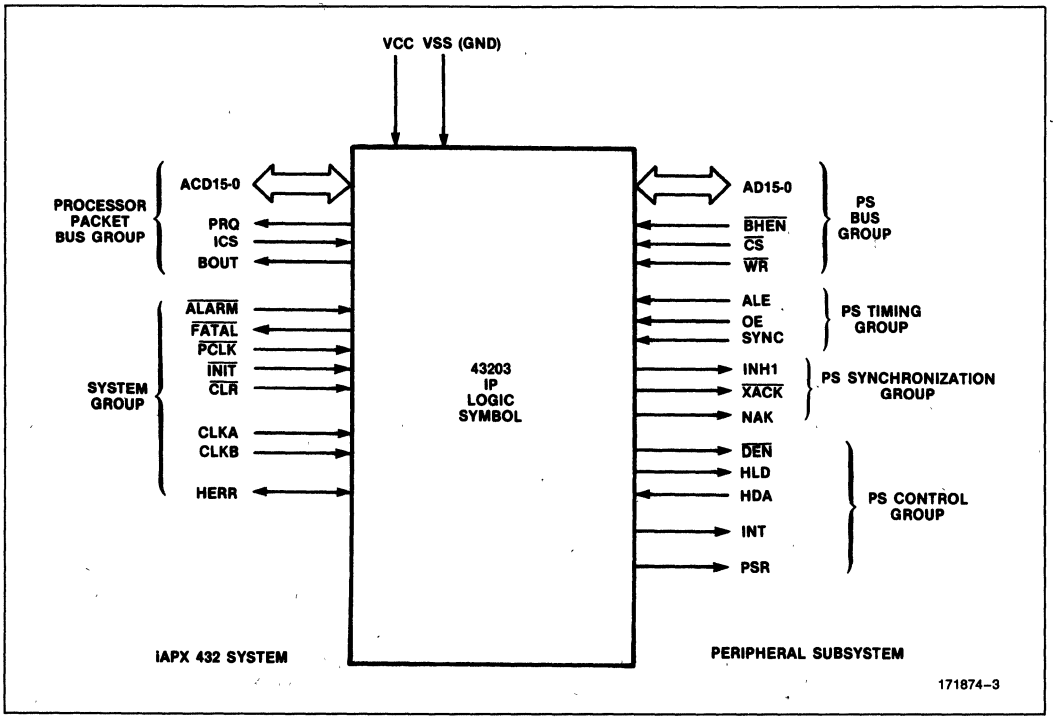


Figure 3. iAPX 43203 IP Logic Symbol

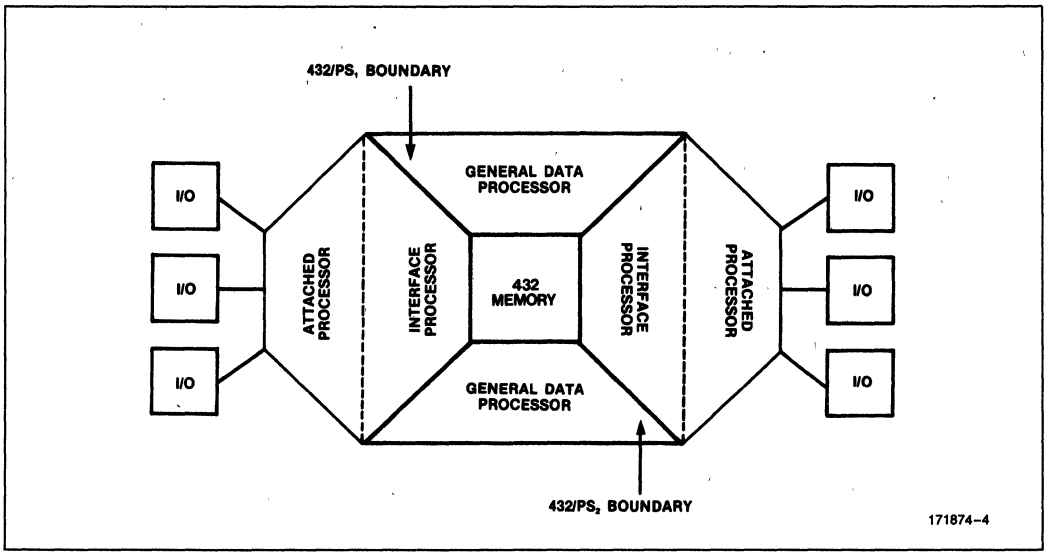


Figure 4A. A Logical View of a 432 System and 2 Peripheral Subsystems

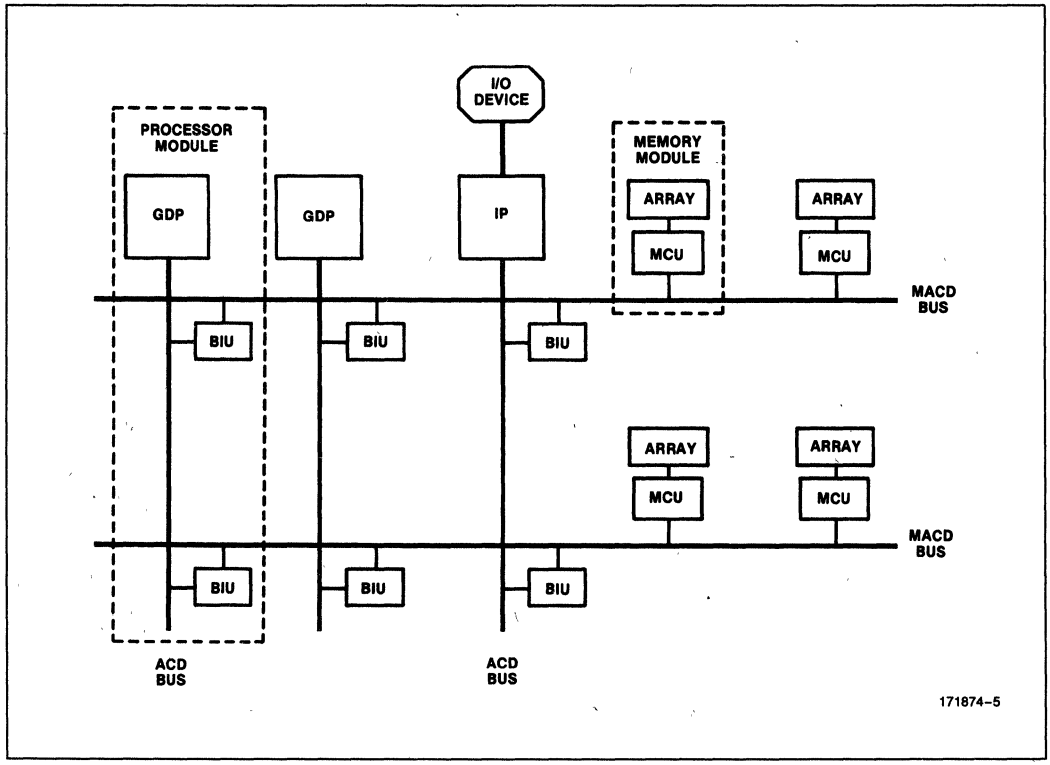


Figure 4B. A Physical View of a 2 Bus System with 2 General Data Processors and 1 Peripheral Subsystem

A peripheral subsystem resembles a conventional mainframe channel in that it assumes responsibility for low-level I/O device support and executes in parallel with system processors. Unlike a simple channel, however, each peripheral subsystem can be configured with a complement of hardware and software resources that precisely fits application cost and performance requirements.

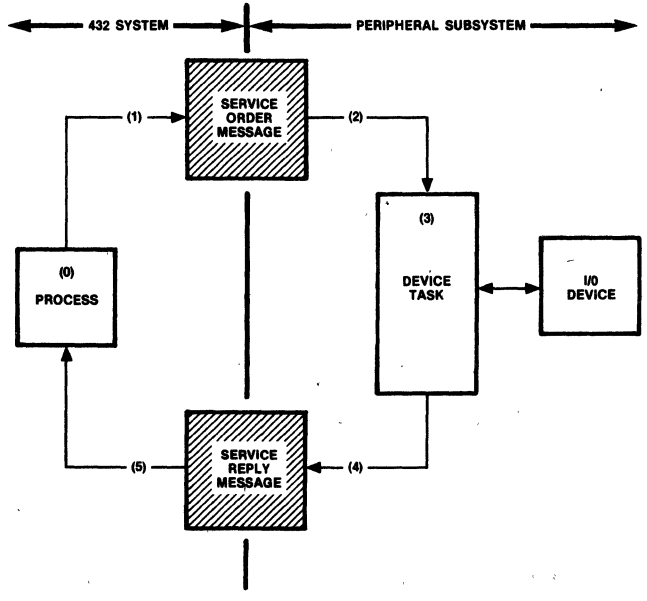
The IP is driven by peripheral subsystem software. To support the transfer of information through the wall that separates a peripheral subsystem from the 432 central system, the IP provides a set of software-controlled windows. A window is used to expose a single object (typed data structure) in 432 system memory so that its contents may be transferred to or from the peripheral subsystem. To preserve the integrity of the capability-based protection mechanisms in the 432 central system, the IP provides the peripheral subsystem with windowed access only to the data part of 432 objects.

In addition, an IP provides a set of functions which are also invoked by peripheral subsystem software. While their operations vary considerably, these functions (and the returned results) generally permit objects in 432 memory to be manipulated as entities, and they enable communication between system processes and software executing in a peripheral subsystem.

It is important to note that both the window and function facilities utilize and strictly enforce 432 addressing and protection systems. Thus, a window provides **protected access** to an object and a function provides a **protected method** by which peripheral subsystem software can interact with the 432 system.

Basic I/O Model

As Figure 5 illustrates, input/output operations in a 432 system are based on the notion of passing messages between **processes** executed within the 432's central system and **device tasks** executed in a peripheral subsystem. A device task can be thought of as the operation of peripheral subsystem hardware and software responsible for managing an I/O device. In contrast, the I/O device itself either produces or consumes data. For example, an I/O device may be a real device (e.g., a terminal), a file, or a pseudo-device (e.g., a spooler).



(0) Process running on GDP needs I/O service.
(1) Process formulates message describing service, sends it to device tasks.
(2) Device task receives service order, interprets it.
(3) Device task transfers data according to service order parameters.
(4) Device task formulates reply message containing result of transfer operation, sends it back to originating process.
(5) Originating process receives reply, interprets it, executes accordingly.

Figure 5. Basic I/O Service Cycle

A message sent from a GDP process requesting I/O service contains information that describes the requested operation (e.g., "read file XYZ"). The device task interprets the message and carries out the operation. If an operation generates input data, the device task returns the task as a message to the originating process. (The device task may also return a message to positively acknowledge completion of a request.)

The 432's object-based architecture provides a very general and powerful mechanism for passing messages between processes. While a given peripheral subsystem may or may not have its own message facility, there is no requirement that it be inherently compatible with the 432. By interposing a peripheral subsystem interface through the IP, the standard 432 interprocess communication system can function with any device task (see Figure 6).

Attached Processor

Almost any general-purpose processor, such as the iAPX 86, iAPX 186, or iAPX 286, can be used as an Attached Processor in an iAPX 432 system, and it need not be dedicated exclusively to working with the Interface Processor. It might, for example, also execute device task software or user applications. Although multiple IPs can service a single AP for increased I/O throughput, only one processor (if a peripheral subsystem uses multiple processors) should be designated to serve as the AP. Other processors (or active agents, such as DMA controllers) may be given access to IP windows, but **control** of the IP should be centralized with the AP.

As Figure 7 shows, the AP is "attached" to the IP in a logical sense only. The physical connections are standard bus signals and one interrupt line (which would typically be routed to the AP via an interrupt controller).

Continuing the concept of the logical I/O processor, the AP fetches instructions, provides the instructions needed to alter the flow of execution, and performs arithmetic, logic, and data transfer operations within the peripheral subsystem.

The IP completes the logical I/O processor by providing data paths between the peripheral subsystem and the central 432 system. In effect, the IP also extends the AP's instruction set so that software running on the logical I/O processor can operate in the 432 system.

As shown in Figure 7, the IP provides both a peripheral subsystem bus interface and a standard 432 processor packet bus interface. By bridging the two buses, the IP provides the hardware link that permits data to flow between the 432 central system and the

The IP connects to the 432 central system in exactly the same way as a 432 GDP. Thus, in addition to being able to access 432 system memory, the IP supports other 432 hardware-based facilities, including interprocessor communication (IPC), alarm signaling, and functional redundancy checking.

On the peripheral subsystem side, the IP provides a very general bus interface that can be adapted to any standard microprocessor bus, including Intel's MULTIBUS and MULTIBUS II architectures, as well as the component buses of the MCS-85 and iAPX 86 families. The IP is connected to the peripheral subsystem bus as if it were a memory component; it occupies a block of memory addresses up to 64k bytes long. Like memory, the IP usually behaves passively within the peripheral subsystem and is driven by peripheral subsystem memory references that fall within its address range.

While the IP generally responds like a memory component, the IP's interrupt signal notifies its AP that an event requiring attention has occurred. Interrupt-handling software on the AP then reads the status information provided by the IP to determine the nature of the event.

To summarize, the Attached Processor and the Interface Processor interact with each other by means of address references generated by the AP and interrupts generated by the IP. Since the IP responds to memory references, other active peripheral subsystem agents (bus masters), such as DMA controllers, may obtain access to 432 system memory via the IP's windows.

Peripheral Subsystem Interface

A peripheral subsystem interface is a combination of hardware and software that acts as an adaptor, enabling message-based communication between a process in the 432 system and a device task in a peripheral subsystem.

The peripheral subsystem interface is managed by software, known generically as the I/O controller. The I/O controller executes on the Attached Processor and uses the facilities of the AP and IP to control the flow of data between the 432 central system and the peripheral subsystem.

The 432 hardware imposes no constraints on the structure of the I/O controller. To help simplify the organization and modification of software, implementers may wish to consider arranging it as a collection of tasks running under the control of a multitasking operating system. Intel's iMAX operating system for the 432, for example, provides interfaces to either the iRMX-86 or iRMX-88 operating systems.

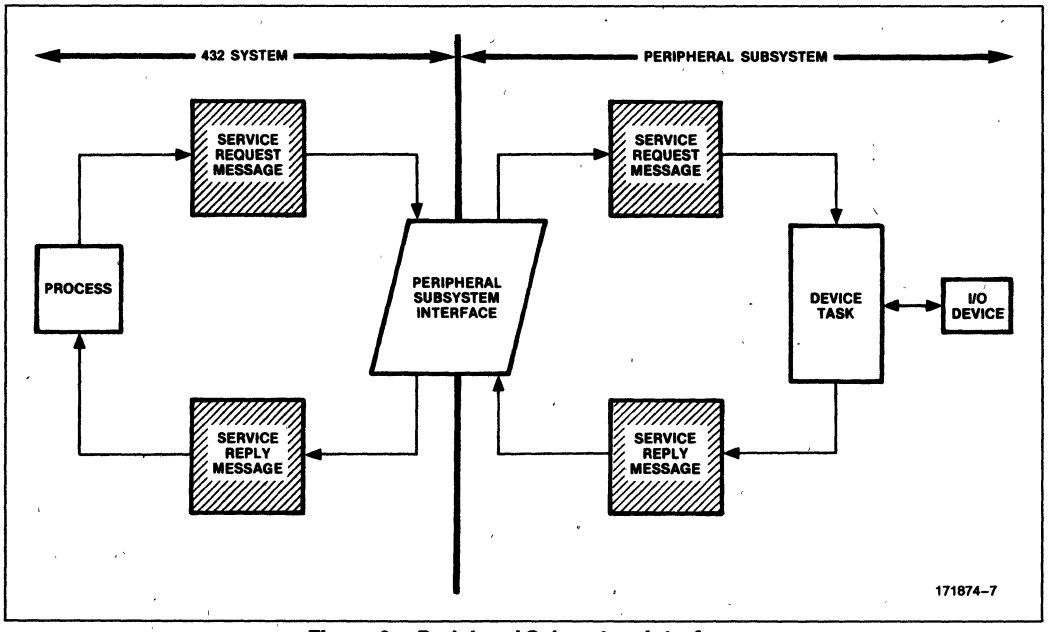


Figure 6. Peripheral Subsystem Interface

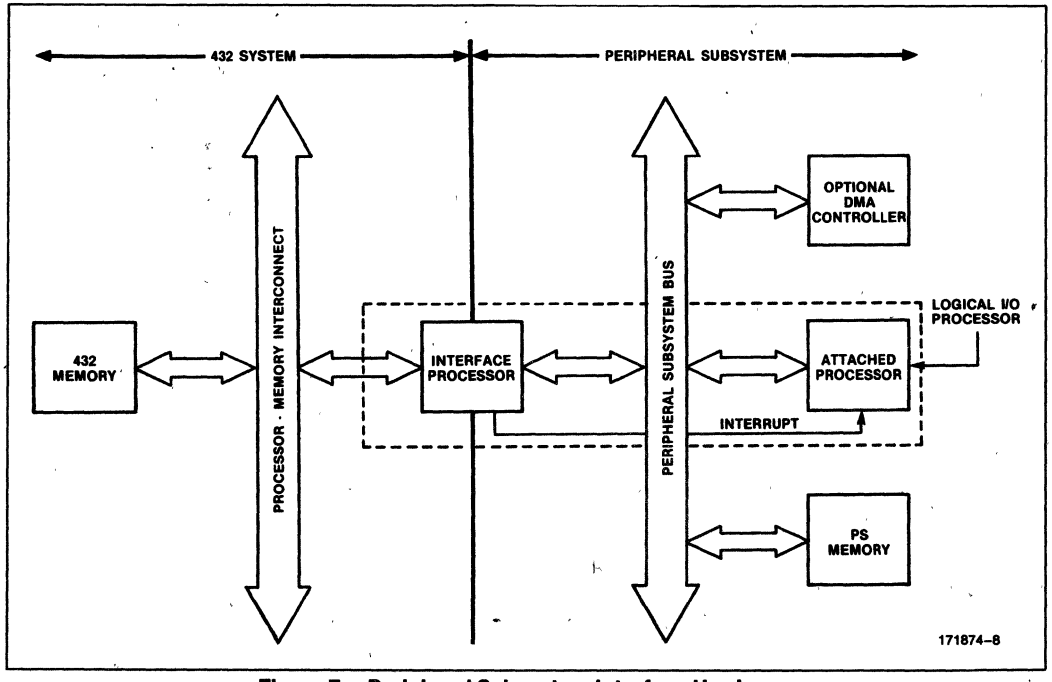


Figure 7. Peripheral Subsystem Interface Hardware

This type of organization supports an asynchronous message-based communication facility. Extending this approach to the device task results in a consistent, system-wide communication model. However, communication within the I/O controller and between the I/O controller and device tasks is completely application-defined. It might be implemented via synchronous procedure calls with "messages" passed in the form of parameters.

However it is structured, the I/O controller interacts with the 432 central system through three major facilities provided by the Interface Processor: execution environments, windows, and functions.

Execution Environments

Within the 432 system the Interface Processor provides a process addressing environment supporting the operation of the I/O controller. This environment is embodied as a set of **system objects** that are used and manipulated by the IP. At any time, the I/O controller is represented in 432 memory by IP process objects and their associated context objects. Like a GDP, the IP is itself represented by a processor object. Representing the IP and its controlling software in this way creates an execution environment analogous to the environment of a process running on a GDP. This environment provides a standard framework for addressing, protection, and communication within the 432 system.

Like a GDP, an IP supports multiple process environments. The I/O controller selects the environment in which a function is to be executed. This permits, for example, the establishment of separate environments corresponding to the individual device tasks in the peripheral subsystem. If an error occurs while the IP controller is executing a function on behalf of one device task of the I/O controller, that error is confined to the associated process, and processes associated with other device tasks are not affected.

Windows

Every transfer of data between the 432 central system and a peripheral subsystem is performed through an IP window. A window defines a correspondence, or mapping, between a subrange of consecutive peripheral subsystem memory addresses (within the range of addresses occupied by the IP) and the data part of an object in 432 system memory (see Figure 8). When an agent in the peripheral subsystem (e.g., the I/O controller) reads a windowed address, it obtains data from the associated object; writing into a windowed address transfers data from the peripheral subsystem to the windowed object. The action of the IP, in mapping the peripheral subsystem address to the system object, is transparent to the agent making the reference. As far as it is aware, it is simply reading or writing memory.

Since a window is referenced as memory, any individual transfer may be made between an object and peripheral subsystem memory, an object and a peripheral subsystem register, or an object and an I/O device. While the latter might be appealing from the standpoint of efficiency, it should be used with caution.

Using a window to connect an I/O device and an object in 432 memory directly has the undesirable effect of propagating real-time constraints imposed by the device beyond the subsystem boundary into the 432 central system and may seriously complicate error recovery. Then too, there is only a finite number of windows and most applications will need to manage them as scarce resources not always instantly available. This means that at least some I/O device transfers may need to be buffered in peripheral subsystem memory until a window becomes available. It may be simplest to buffer **all** I/O device transfers in memory and use the windows to transfer data between the peripheral subsystem memory and 432 system memory at regular intervals.

There are four IP windows that can be mapped onto four different objects. The I/O controller may alter the windows during execution to obtain access to different objects. References to windowed subranges may be interleaved in time and may be driven by different agents in the peripheral subsystem. For example, the AP and a DMA controller may be driving transfers concurrently, subject to the same bus arbitration constraints that would apply if they were accessing memory.

Functions

A fifth window, the control window, provides the I/O controller with access to the Interface Processor's **function request facility.** The I/O controller requests the execution of an IP function by writing operands and an opcode into predefined locations in the control window's subrange. This procedure is very similar to writing commands and data to a memory-mapped peripheral controller (e.g., a floppy disk controller). Upon completion of the function, the IP interrupts the AP and provides status information that the IP controller can read through the control window. The IP can respond concurrently to transfer requests to the other four windows while it is executing a function. In addition, data transfers through windows 0 through 3 may be interleaved with function request sequences through the control window.

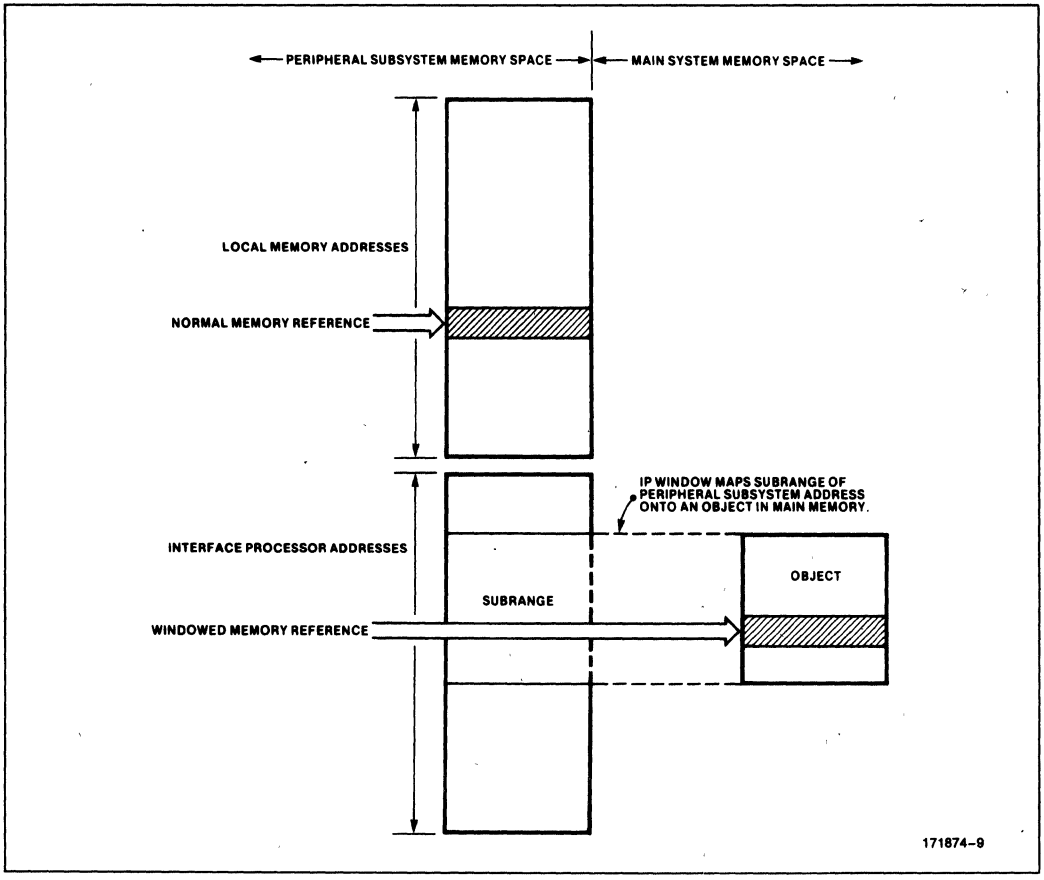


Figure 8. Interface Processor Window

The IP's function set permits the I/O controller to:

- Alter windows
- Exchange messages with GDP processes
- Manipulate objects

These functions can be viewed as extensions to the Attached Processor's instruction set that enable the I/O controller to operate in the 432 central system.

The combination of the IP's function set and windows, the AP's instruction set, and possibly additional facilities provided by a peripheral subsystem operating system, permits greater flexibility in designing I/O systems. By using the more sophisticated IP functions, powerful I/O controllers capable of relieving the 432 system of much I/O-related processing can be built.

I/O Data Flow Summary

Figure 9 summarizes the relationship of hardware and software components that cooperate to move data between an I/O device and 432 system memory. Notice how the peripheral subsystem interface not only bridges the 432 central system/peripheral subsystem boundary, but also hides the characteristics of one system from the other. As far as a device task is concerned, its job is to move data between memory and an I/O device; it may be completely unaware that it is connected to a 432 system.

This means that existing device tasks may be utilized in a 432 system with little or no modification, and that programmers working on device tasks need not be trained in the operation of the 432. Similarly, a GDP process that needs an I/O service need have no knowledge of the details and characteristics of the target I/O device. As far as it is concerned, it "performs" I/O in the same way that it communicates with a cooperating process—by sending and receiving messages through the 432 interprocess communication facility.

Other IP Facilities

The preceding sections have described the Interface Processor as it is used most of the time. The IP also has two additional capabilities that are used in special circumstances: physical reference mode and interconnect access.

Physical Reference Mode

Normally, an IP operates in **logical reference mode** using capability-based addressing. In other words, it utilizes an access descriptor to specify a particular 432 object rather than a physical location in memory. There are times, however, when logical referencing is impossible because the objects used by the hardware to perform logical-to-physical address translation are absent (or, less likely, damaged). In these situations, the IP can be used in physical reference mode.

An IP operating in physical reference mode circumvents the protection mechanisms of the 432 system. The IP provides a reduced set of functions, and makes no distinction between the data part and the access part of an object. In physical mode, a window maps directly onto a range of contiguous physical

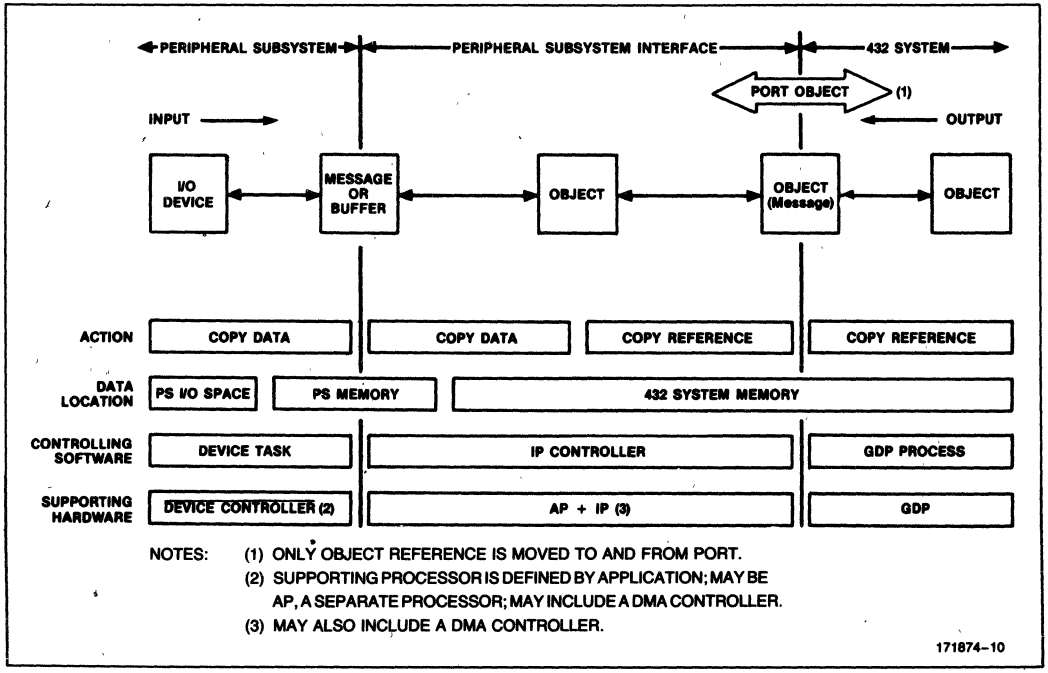


Figure 9. I/O Data Flow Summary

memory addresses (rather than object structures in 432 system memory). The IP selects a segment by specifying a 24-bit physical address when it establishes a window, and interprets subsequent subrange references as 16-bit displacements (there is no length checking) from the segment's base address. This simple base-plus-displacement addressing is similar to traditional computer addressing techniques.

Physical reference mode is used most often during system initialization to load images of objects from a peripheral subsystem into 432 system memory. Once the required objects are available, processors can begin normal logical reference mode operations. Logical mode cannot be used until the object tables required for logical-to-physical address translation have been constructed and loaded into 432 system memory.

Interconnect Access

In addition to memory, the iAPX 432 architecture defines a second, independent address space called the **processor/memory interconnect address space.** The interconnect space allows interconnect objects containing one or more interconnect registers to be maintained. Interconnect registers are double-byte quantities aligned on double-byte boundaries. With the exception of a few reserved addresses, the definition and use of interconnect locations is not predefined for the IP.

The IP (like the GDP) requires two register locations in the interconnect space to be defined for any system:

- The processor ID register (interconnect address 0)
- The interprocessor communication register (interconnect address 2)

The remainder of the interconnect address space may be used to store or acquire other information such as configuration parameters, error logs, and other application-specific quantities.

Window 1 is software-switchable between the memory and the interconnect spaces. In logical reference mode, the interconnect space is addressed in the same object-oriented manner as the memory space with the IP automatically performing the logical-to-physical address translation.

To access the interconnect space, the I/O controller must specify an access selector for an interconnect object that exposes a segment of the interconnect space to the IP. The normal window addressing scheme is then used to locate individual interconnect registers within an object. Switching window 1 to interconnect access mode gives the IP access to interconnect objects. Writing or reading window 1 then is equivalent to the MOVE TO INTERCONNECT and MOVE FROM INTERCONNECT operators of the GDP.

In physical reference mode, the interconnect space is addressed as a linear array of even-addressed, double-byte interconnect registers. As with physical reference mode memory accesses, the switchable window is established with a 24-bit address. Peripheral subsystem references to the corresponding subrange are likewise interpreted by the IP as 16-bit displacements from the base address to individual interconnect registers.

Memory Structure

The architecture of the iAPX 432 defines a two-level memory space. Software operates in a segmented environment in which a logical address specifies the location of an object (data structure), and the processor automatically translates this logical address into a physical address. A physical address is 24 bits long, allowing a maximum physical memory of 16 Megabytes. When requesting access to either read or write memory, a 432 processor transmits the beginning byte of the memory byte to be referenced along with the length of the access. An Interface Processor can request to read or write up to eight bytes in a single memory access.

The multiprocessor architecture of the iAPX 432 places requirements on the memory system to ensure the integrity of data. If several processor were permitted to read and modify the same data structure without coordination, the data could become inconsistent or erroneous. Therefore, indivisible read-modify-write operations are necessary to manipulate system objects.

When an RMW-read is processed for a location in memory, any other RMW-reads to that location must be delayed to that location until a RMW-write to that location has been received (or until an RMW timeout has occurred). While the memory system is awaiting the RMW-write, however, other types of reads and writes are permitted.

Even so, if an operand is a double-byte or longer, the memory system must still ensure that the entire operand has been read or written before once again allowing access to the same location. For example, if two simultaneous writes to the same location occur, the memory system must guarantee that the set of locations used to store the operand does not get changed to some interleaved combination of the two written values.

Designing Fault-Tolerant Systems

When used together, the five components in the iAPX 432 family provide all the logic necessary to build a system that will tolerate the failure of any single component or bus, yet continue to execute programs without error and without interruption. No software intervention is required: fault detection, isolation, and reconfiguration of the system is performed entirely by the hardware.

Each Interface Processor is able to detect hardware errors automatically because of a capability known as Functional Redundancy Checking (FRC), so called because a second or redundant IP checks the operations of the first or master IP. Functional Redundancy Checking provides the low-level hardware support upon which hardware fault-tolerant modules are constructed.

During initialization, each IP is assigned to operate as either a master or a checker (see Figure 10). While a master operates in a conventional manner, a checker places all output pins that are being checked into a high-impedance state. Those pins which are to be checked on a master and checker are parallel-connected, pin for pin, such that the checker is able to compare its master's output pin values with its own. If on any cycle, the values differ, the checker asserts HERR and the faulty components can be immediately disabled. Thus, any hardware errors can be detected as they occur and before they have had the opportunity to corrupt the operation of other components in the system.

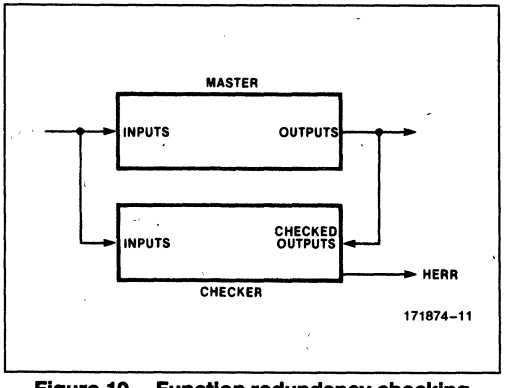


Figure 10. Function redundancy checking detects hardware errors automatically

While FRC can be used alone to provide automatic error detection, a completely fault-tolerant system must also be able to reconfigure itself, replacing the set of failed components with another pair that is still working. In order to do so, the 432's architecture enables two pairs of master/checker components to be combined to form primary and shadow processors in a configuration known as Quad Modular Redundancy (QMR). See Figure 11.

Every module in a QMR system is paired with another self-checking module of the same type. The pair of self-checking modules operates in lock step and provides a complete and current backup for all state information in the module. The mechanism is known as module shadowing because a shadow is ready to fill in if the primary fails (or vice versa). Fault detection and recovery occurs transparently to both application and system software. When a fault is detected, the faulty pair is automatically disabled, and the remaining pair takes over. Only then is system software notified that a failure has occurred.

A more complete discussion of the fault-tolerant capabilities of the iAPX 432 can be found in the **iAPX 43204-iAPX 43205 Fault Tolerant Bus Interface and Memory Control Units** data sheet (Order Number 210963).

Processor Packet Bus Definition

Processors sharing the same memory must contend for access to that memory over one or more system buses. Therefore, efficient bus utilization is essential in a multiprocessing system. A simple and efficient approach to building a 432 interconnect system is to use the iAPX 43204 Bus Interface Unit; the VLSI component provides the necessary circuitry to interconnect 432 processors with from one to eight memory buses. Some system designers, however, may prefer to take other approaches to the interconnect design to optimize the cost/performance ratio of the hardware for their specific application. With that requirement in mind, Intel formulated an iAPX 432 packet bus protocol which supports a wide range of system bus architectures.

To reduce bus occupancy and increase the performance range of 432 systems, the packet bus protocol separates processor requests and replies into separate packets. A processor can issue a request packet and leave the system bus free until the reply packet is returned from memory.

As a second method of maximizing the efficiency of bus utilization, the packet bus protocol allows variable length packets of data. If a processor wishes to read a 64-bit operand, it can be done with a single request and reply packet. Thus, fewer individual storage requests are required to process long operands. This aspect of the protocol enables processors to interface easily to 16-bit, 32-bit, or even 64-bit system buses.

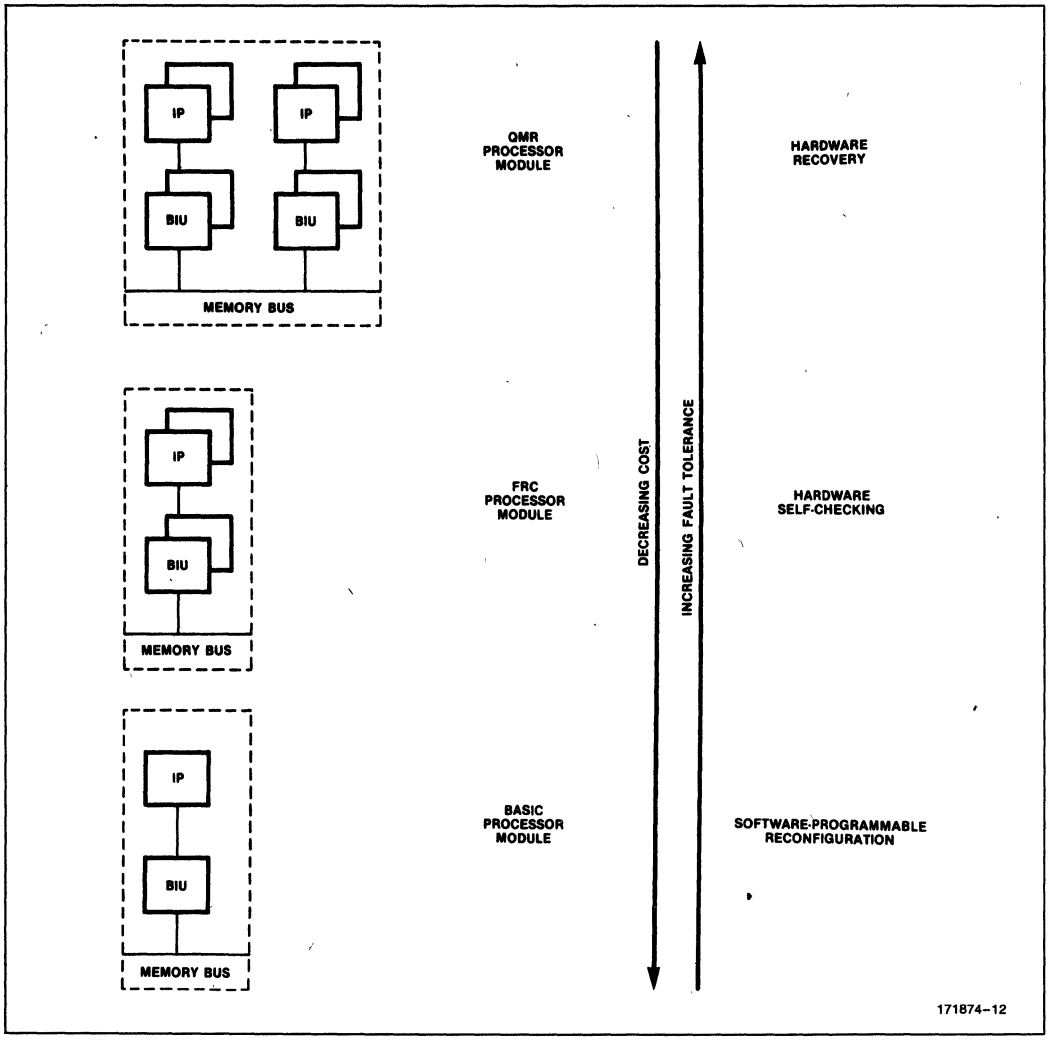


Figure 11. Fault-Tolerant Alternatives

This section describes the 19 signal lines that compose the Processor Packet bus and their timing relationships. While this section defines all valid bus activities, the processors do not necessarily perform all allowed activities; nevertheless, slaves to the Processor Packet bus must support all state transitions to ensure compatibility.

The Processor Packet bus consists of three control lines:

- PRQ (Processor Packet bus request)
- B_{OUT} (Enable Buffers for Output)
- ICS (Interconnect Status)

PRQ has two functions whose use depends upon the application; for example, PRQ either indicates the first cycle of a transaction on the bus or the cancellation of a transaction initiated during the previous cycle. Of the three control lines, BOUT has the simplest function, serving as a direction control for buffers in larger systems which require more electrical drive than the processor components can provide. The ICS signal has three different interpretations depending on the state of the Processor Packet bus transaction. It may indicate whether or not:

- An interprocessor communication (IPC) is waiting,
- A slave requires more time to service the processor's request, or
- A bus error has occurred.

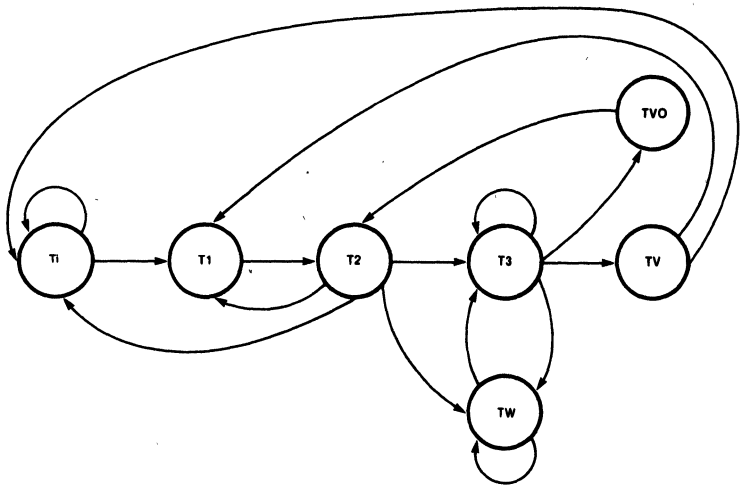
The bus also includes 16 three-state Address/Control/Data lines (ACD_{15}–ACD_0). These lines emit information to specify the type of cycle being initiated; transmit addresses, data to be written, and control information; and during a read operation, receive data returned to the processor. Details of the ACD operation are summarized in Figure 12.

Address/Control/Data Lines

In the first cycle (T1 or Tvo) of a Processor Packet bus transaction (indicated by the rising edge of PRQ), the eight high-order ACD bits (ACD_{15}–ACD_8) specify the type of the current transaction. In this first cycle, the low-order ACD bits (ACD_7–ACD_0) contain the least significant eight bits of the 24-bit address.

During the next cycle (T2), the remainder of the address is presented on the ACD pins, aligned so that the most significant byte of the address is on ACD_{15}–ACD_8 while the mid-significant byte is on ACD_7–ACD_0. If PRQ is asserted during T2, the access is cancelled and the ACD lines are not defined.

During the third bus cycle (T3 or Tw) of a Processor Packet bus transaction, the processor presents a high impedance to the ACD lines for read transactions and asserts data for write transactions.



171874-13

Initial State	Next State	Trigger
Ti	T1	Bus cycle desired
	Ti	No bus cycle desired
T1	T2	Unconditional
T2	T3	ICS high
	Tw	ICS low
	T1	Cancelled, Access Pending (Not generated by IP)
	Ti	Cancelled, No Access Pending
T3	T3	Additonal transfer required, ICS high
	Tw	Additional transfer required, ICS low
	Tv	All transfers completed, no overlapped access
	Tvo	Current write with overlapped access
Tv	Ti	No access pending
	T1	Access pending
Tvo	T2	Unconditional
TW	Tw	ICS low
	T3	ICS high

Figure 12. Processor Packet Bus State Diagram

171874-002

Once the bus has entered T3 or Tv, the sequence of state transactions depends on the type of cycle requested during the preceding T1 or Tvo. Accesses ranging in length from 1 to 32 bytes may be requested, although the IP will request no more than 8 bytes in a single access (see Table 2). If a transfer of more than one double byte has been requested, T3 must be entered for every double byte that is transferred. ICS dictates whether the processor simply enters T3 or first enters Tw to wait.

After all data is transferred, the processor enters either Tv or Tvo. Tvo can be entered only when the processor is prepared to accomplish an immediate write transfer (overlapped access). During Tvo, the ACD lines contain address and specification information aligned in the same fashion as T1. If the processor does not require an overlapped access, the bus state move to Tv (the ACD lines will be high impedance). After Tv, a new bus cycle can be initiated with T1, or the processor may enter the idle state (Ti).

Interconnect Status (ICS)

As discussed earlier, ICS has three possible interpretations depending on the current state of the bus transaction (see Table 3). Even so, under most conditions ICS indicates whether or not an IPC is pending; a valid low during any of these cycles with IPC significance signals the processor that an IPC has been received. While an iAPX 432 processor is only required to record and service one IPC or reconfiguration request at a time, logic in the interconnect system must record and sequence multiple (and possibly simultaneous) IPC occurrences and reconfiguration requests. Thus, the logic that implements ICS must accommodate global and local IPC arrivals and requests for reconfiguration as individual events:

1. Assert IPC significance on ICS for the arrival of an IPC or reconfiguration request.

2. When the iAPX 432 processor reads interconnect address register 2, it will respond to one of the status bits for the IPC or reconfiguration request signalled on ICS in the following order:

   BIT 2 (1 = reconfigure, 0 = do not reconfigure)

   BIT 1 (1 = global IPC pending, 0 = no global IPC)

   BIT 0 (1 = local IPC pending, 0 = no local IPC)

3. The logic in the interconnect system must clear the highest order status bit that was serviced by the iAPX 432 processor, and if an additional IPC message has arrived, the interconnect logic must signal an additional IPC to the processor by setting ICS high for at least one cycle and then setting ICS low for at least one cycle, while ICS has IPC significance.

Table 2. ACD Specification Encoding

ACD 15	ACD 14	ACD 13	ACD 12	ACD 11	ACD 10	ACD 9	ACD 8
Access	**Op**	**RMW**		**Length**		**Modifiers**	
0 – Memory	0 – Read	0 – Normal		000-1 Byte 001-2 Bytes 010-4 Bytes 011-6 Bytes 100-8 Bytes		ACD 15 = 0: 00-Inst Seg Access 01-Stack Seg Access	
1 – Interconnect Space	1 – Write	1 – RMW		101-10 Bytes* 110-16 Bytes* 111-32 Bytes*		10-Context Ctl Seg Access 11-Other	
				* Not implemented		ACD 15 = 1: 00-Reserved 01-Reserved 10-Reserved 11-Interconn Register	

Table 3. ICS Interpretation

State	Significance	Level	
		High	Low
Ti, T1, T2	IPC	No IPC writing	IPC writing
T3, Tw	Stretch	Don't Stretch	Stretch
Tv, Tvo	Err	Bus Error	No Error

Processor Packet Bus Request (PRQ)

PRQ is normally low and goes high only during T1, T2, and Tvo. High levels during Tvo and T1 indicate the first cycle of an access. A high level during T2 indicates that the current cycle is to be cancelled. See Table 4.

Table 4. PRQ Interpretation

State	PRQ	Condition
Ti	0	Always
T1	1	Initiate access
T2	0	Continue access
	1	Cancel access
T3	0	Always
Tw	0	Always
Tv	0	Always
Tvo	1	Initiate overlapped access

Enable Buffers for Output (BOUT)

B_{OUT} is provided to control external buffers when they are present. Table 5 and Figures 13 through 18 show its state under various conditions.

Table 5. B_{OUT} Interpretation

B_{OUT}	Always High	Low-to-High Transition or Low	High-to-Low Transition or Low	High-to-Low Transition or High
Write	T1, T2, T3, Tw, Tvo	Ti	None	Tv
Read	T1, T2	Ti, Tv	T3, Tw	None

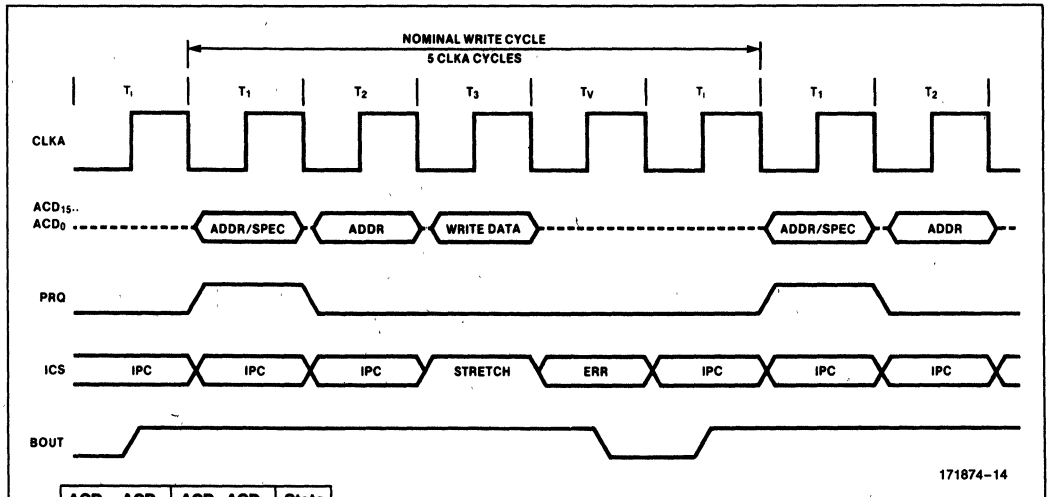
Processor Packet Bus Timing

Each timing diagram shown on the following pages illustrates the timing relationships on the Processor Packet bus during various types of transactions. This approach to transfer timing allows maximum time for the transfer to occur and yet guarantees hold time.

Any agent connected to the Processor Packet bus is recognized as either a processor (a GDP or IP) or a slave (e.g., the memory subsystem).

In all transfers between a processor and a slave, the data to be driven is clocked for three-quarters of a cycle before it is sampled. This allows adequate time for the transfer and ensures sufficient hold time after sampling. The BOUT timing is unique because BOUT functions as a direction control for external buffers.

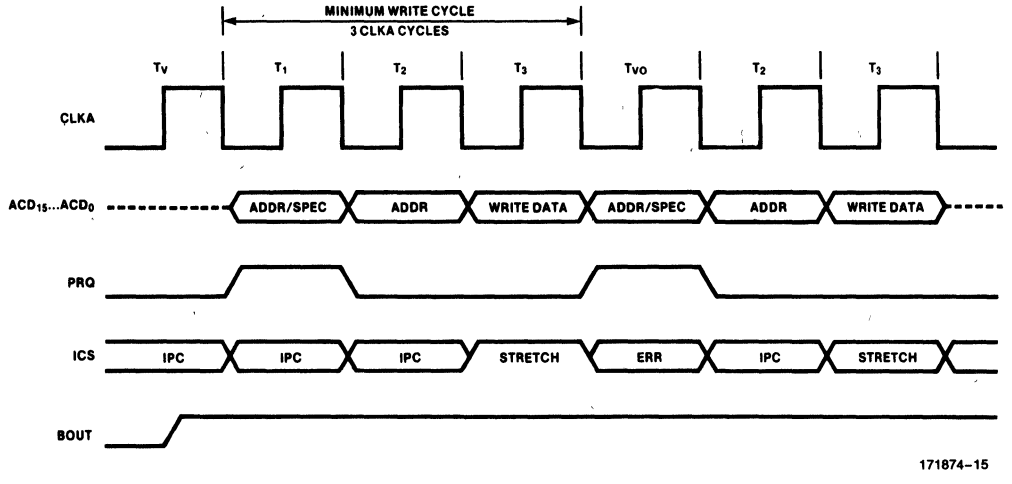
Detailed set-up and hold times can be found in the AC Charcteristics section.



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data 1*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2

*Undefined if single byte write

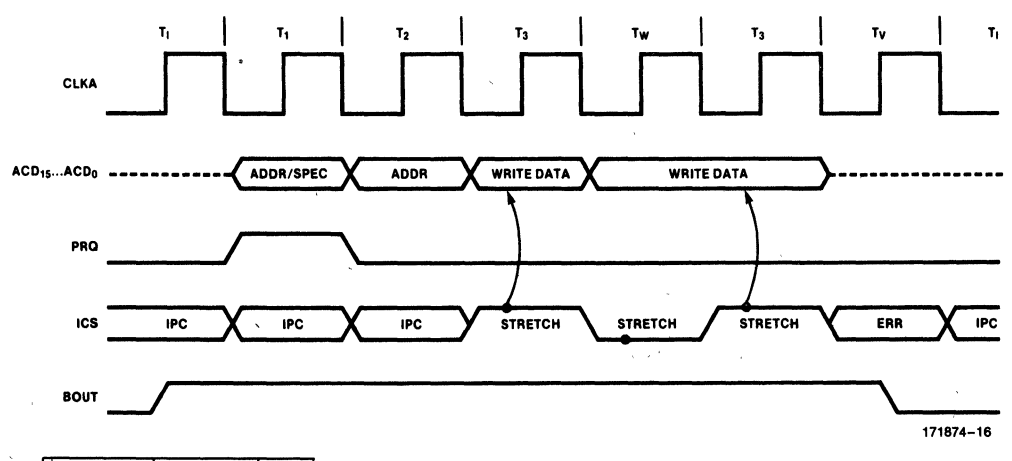
Figure 13. Nominal Write Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Tv**
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1*	Lo-data1	T3
Spec	Lo-adr	Tvo
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3

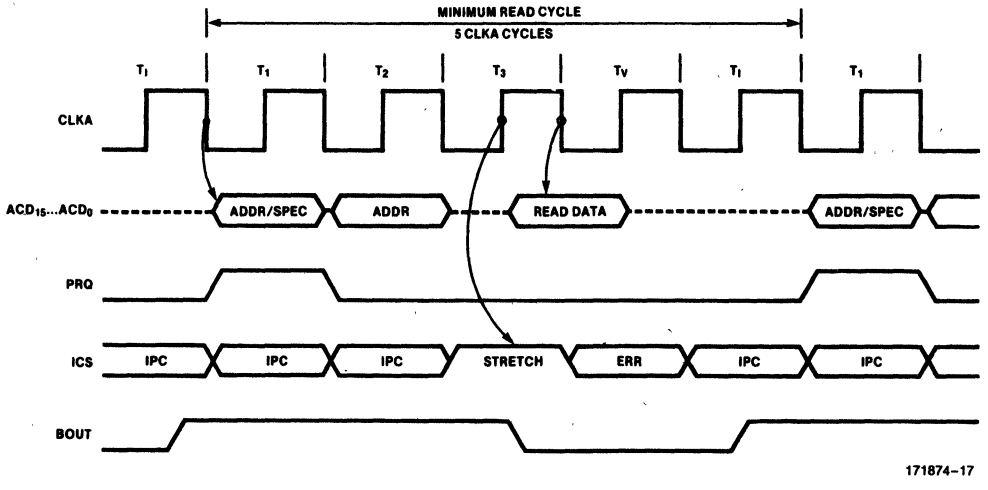
*Undefined if single byte write
**(Preceded by read cycle)

Figure 14. Minimum Write Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data1	Lo-data1	T3
Hi-data2	Lo-data2	Tw
Hi-data2	Lo-data2	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

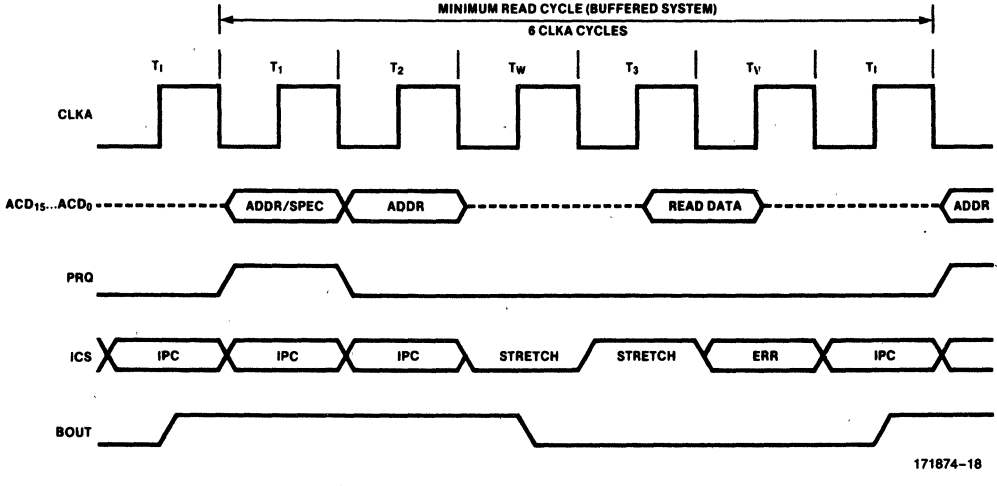
Figure 15. Stretched Write Cycle Timing



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1

*Undefined if single byte read

Figure 16. Minimum Read Cycle Timing

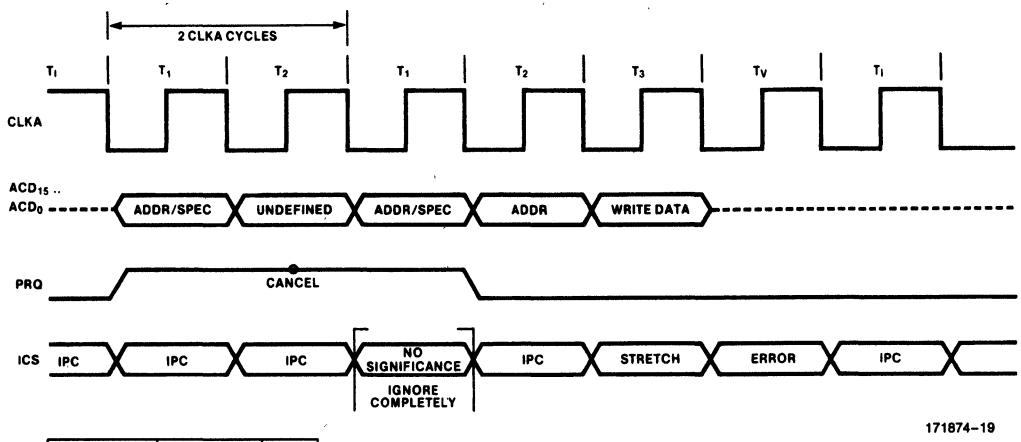


ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Hi-adr	Mid-adr	T2
Hi-Z	Hi-Z	Tw
Hi-data1*	Lo-data1	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

*Undefined if single byte read

Figure 17. Stretched Read Cycle Timing

171874-002



ACD_{15} ACD_8	ACD_7 ACD_0	State
Hi-Z	Hi-Z	Ti
Spec	Lo-adr	T1
Undefined	Undefined	T2**
Spec	Lo-adr	T†
Hi-adr	Mid-adr	T2
Hi-data*	Lo-data	T3
Hi-Z	Hi-Z	Tv
Hi-Z	Hi-Z	Ti

*Undefined if single byte write
**Access Cancelled
†New Access Started (Slave must support this subsequent access even though all processors may not implement it.)

Figure 18. Minimum Faulted Access Cycle

ELECTRICAL CHARACTERISTICS

Tables 6 through 8 and Figure 19 through 24 provide electrical specifications and include I/O timing, read/write timing, and the maximum ratings for the IP.

Table 6. 43203 Absolute Maximum Ratings

Absolute Maximum Ratings	
Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage on Any Pin with Respect to GND	−1V to +7V
Power Dissipation	2.5 Watts

Table 7. iAPX 43203 DC Characteristics

V_{CC} = 5V ± 10%		T_a = 0°C to 70°C		
Spec	**Description**	**Min**	**Max**	**Units**
V_{ILC}	Clock Input Low Voltage	−0.3	+0.5	V
V_{IHC}*	Clock Input High Voltage	3.5	V_{CC}+0.5	V
V_{IL}	Input Low Voltage	−0.3	0.8	V
V_{IH}	Input High Voltage	2	V_{CC}+0.5	V
V_{OL}	Output Low Voltage	—	0.45	V
V_{OH}	Output High Voltage	2.4	V_{CC}	V
I_{CC}	Power Supply Current	—	450	mA
I_{IL}	Input Leakage Current	—	±10	μA
I_O	Outut Leakage Current	—	±10	μA
I_{OL}	@0.45 Vol			
	HERR	—	8	mA
	$\overline{FATAL}$	—	4	mA
	AD_{15} . . . AD_0	—	4	mA
	OTHER	—	2	mA
I_{OH}	@2.4 V_{OH}	—	−0.1	mA

*For operation at 5 MHz or slower, the 43203 may be operated with a V_{IHC} minimum of 2.7 volts.

171874-002

Table 8. iAPX 43203 AC Characteristics

V_{CC} = 5 ± 10%
T_A = 0°C to 70°C

Loading: AD_{15} . . . AD_0 . . . . . 20 to 100 pF
Other . . . . . 20 to 70 pF

Symbol	Description	8 MHz Min	8 MHz Max	7 MHz Min	7 MHz Max	5 MHz Min	5 MHz Max	Unit
Global Timing Requirements								
t_{CY}	Clock Cycle Time	125	500	143	500	200	500	ns
t_r, t_f	Clock Rise and Fall Time	—	10	—	10	—	10	ns
T1, T2								
T3, T4	Clock Pulse Widths	26	250	30	250	45	250	ns
t_{IS}	$\overline{INIT}$ to Signal Hold Time	15	—	20	—	20	—	ns
t_{SI}	Signal to $\overline{INIT}$ Setup Time	10	—	10	—	10	—	ns
t_{IE}	$\overline{INIT}$ Enable Time	8	—	8	—	8	—	t_{CY}
System Side Timing Requirements								
t_{DC}	Signal to Clock Setup Time	5	—	5	—	5	—	ns
t_{CD}	Clock to Signal Delay Time	—	55	—	85	—	85	ns
t_{DH}	Clock to Signal Hold Time	25	—	35	—	35	—	ns
t_{OH}	Clock to Signal Output Hold Time	15	—	15	—	20	—	ns
t_{EN}	Clock to Signal Output Enable Time	15	—	20	—	20	—	ns
t_{DF}	Clock to Signal Data Float Time	—	55	—	75	—	75	ns
Peripheral Subsystem Side Timing Requirements								
t_{AS}	AD_{15} . . . AD_0, $\overline{CS}$, $\overline{WR}$, $\overline{BHEN}$ Setup Time to ALE Low	0	—	0	—	0	—	ns
t_{CH}	AD_{15} . . . AD_0, $\overline{CS}$, $\overline{WR}$, $\overline{BHEN}$ Hold Time to ALE Low	32	—	35	—	35	—	ns
t_{SS}	SYNC High Setup Time to CLK_A High	50	—	60	—	60	—	ns
t_{SH}	SYNC Low Hold Time to CLK_A High	30	—	40	—	40	—	ns
t_{SW}	SYNC High Pulse Width	50	t_{SS} + 1.5 t_{CY}	80	t_{SS} + 1.5 t_{CY}	60	t_{SS} + 1.5 t_{CY}	ns
t_{DS}	Write Data Setup to Sampling CLK_A High	10	—	20	—	20	—	ns
t_{DX}	Write Data Hold to Sampling CLK_A Low (Advanced $\overline{XACK}$)	10	—	20	—	20	—	ns
t_{DHX}	Write Data Hold to $\overline{XACK}$	5	—	5	—	5	—	ns
t_{ASY}	AD_{15} . . . AD_0, $\overline{CS}$, $\overline{WR}$, $\overline{BHEN}$ Setup to SYNC	120	—	160	—	160	—	ns

171874-002

Table 8. iAPX 43203 AC Characteristics (Continued)

Symbol	Description	8 MHz Min	8 MHz Max	7 MHz Min	7 MHz Max	5 MHz Min	5 MHz Max	Unit
Peripheral Subsystem Timing Responses								
t_{SDH}	CLK_A High to HLD, INT, PSR	—	75	—	90	—	90	ns
t_{AIH}	Valid $AD_{15} \ldots AD_0$, $\overline{CS}$ to Chip IWH1 Valid Delay	—	80	—	85	—	85	ns
t_{EDE}	OE to $\overline{DEN}$ Delay	—	65	—	70	—	70	ns
t_{END}	OE to Enable $AD_{15} \ldots AD_0$ Buffers Delay (Read Cycle)	—	70	—	75	—	75	ns
t_{DAD}	OE to Disable $AD_{15} \ldots AD_0$ Buffers Delay (Read Cycle)	—	52	—	52	—	52	ns
t_{CED}	CLK_A High to Enable $AD_{15} \ldots AD_0$ Buffers Delay	—	70	—	75	—	75	ns
t_{CVD}	CLK_A High to Valid Read Data Delay	—	80	—	90	—	90	ns
t_{OX}	OE Inactive to $\overline{XACK}$ Inactive Delay	—	80	—	90	—	90	ns
t_{DDS}	$AD_{15} \ldots AD_0$ Disable Setup to $\overline{DEN}$ High	0	—	0	—	0	—	ns
t_{XDE}	$\overline{XACK}$ Low to $\overline{DEN}$ High (Write Cycle)	—	35	—	40	—	40	ns
t_{CDE}	CLK_A High to $\overline{DEN}$ Low	—	70	—	75	—	75	ns
$\overline{XACK}$ Timing Characteristics								
	Buffered Accesses with XD = 0							
t_{AX}	ALE High to $\overline{XACK}$ Valid	0	65	0	70	0	70	ns
t_{DSX}	$AD_{15} \ldots AD_0$ Read Data Valid Setup to $\overline{XACK}$ Valid (When internal state does not allow $\overline{XACK}$ before SYNC)	20	—	20	—	20	—	ns
t_{ADX}	Valid $AD_{15} \ldots AD_0$ to $\overline{XACK}$ Valid (When Internal State allows $\overline{XACK}$ before SYNC)	—	120	—	140	—	140	ns
t_{DSX}	Buffered Accesses (with XD = 1 or XD = 2) or Random Accesses $AD_{15} \ldots AD_0$ Read Data Valid Setup to $\overline{XACK}$	20	—	20	—	20	—	ns
	Faulted Accesses							
t_{SDL}	CLK_A Low to $\overline{NAK}$	—	75	—		—	90	ns
t_{SNX}	Setup of $\overline{NAK}$ to $\overline{XACK}$	50	—	50	—	50	—	ns

NOTES:

1. All timing parameters are measured at the 1.5V level except for CLK_A and CLK_B which are measured at the 1.8V level.

2. 5 MHz components are marked CR43203-5; 7 MHz components are marked CR43203-7, and 8 MHz components CR43203-8.

3. Write data is sampled for only one clock cycle. The PS must meet t_{DHX} specification, thereby guaranteeing t_{DX}.

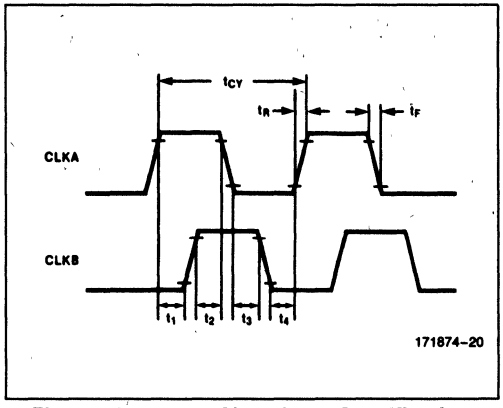


Figure 19. 43203 Clock Input Specification

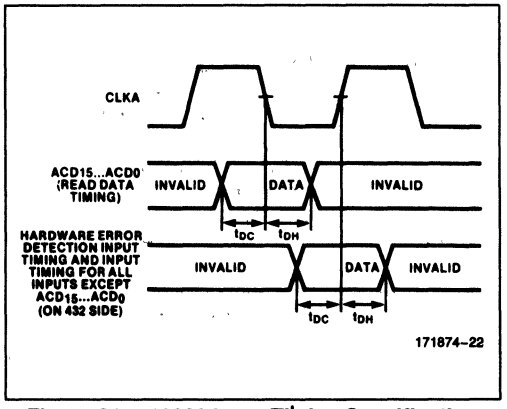


Figure 21. 43203 Input Timing Specification

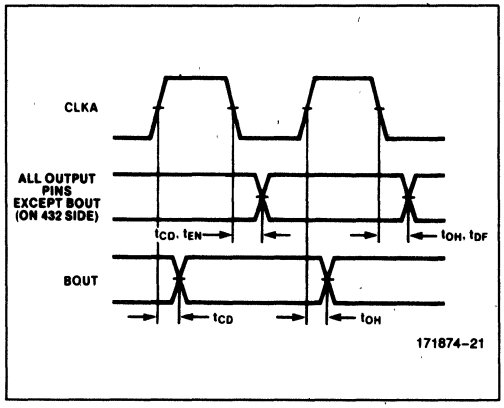


Figure 20. 43203 Output Timing Specification

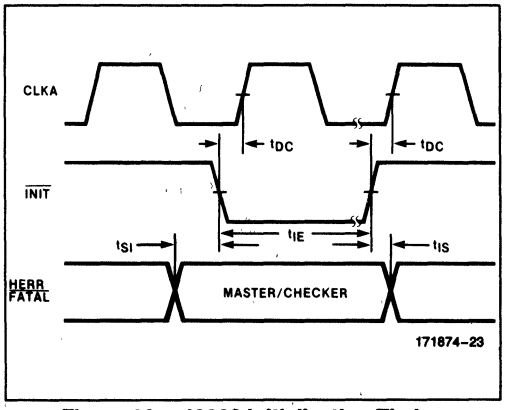


Figure 22. 43203 Initialization Timing

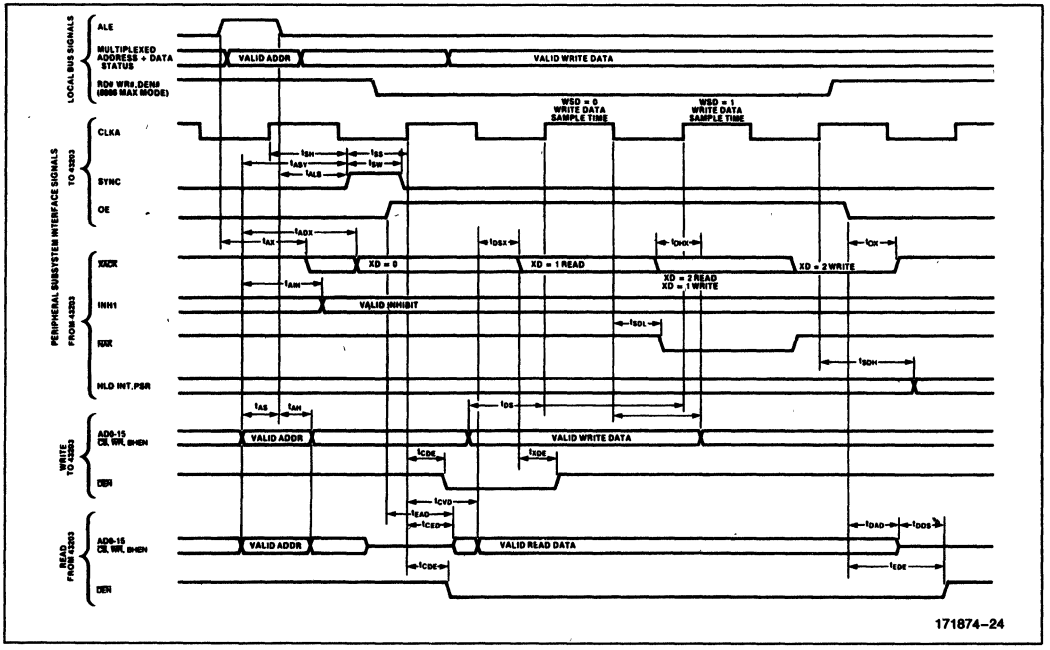
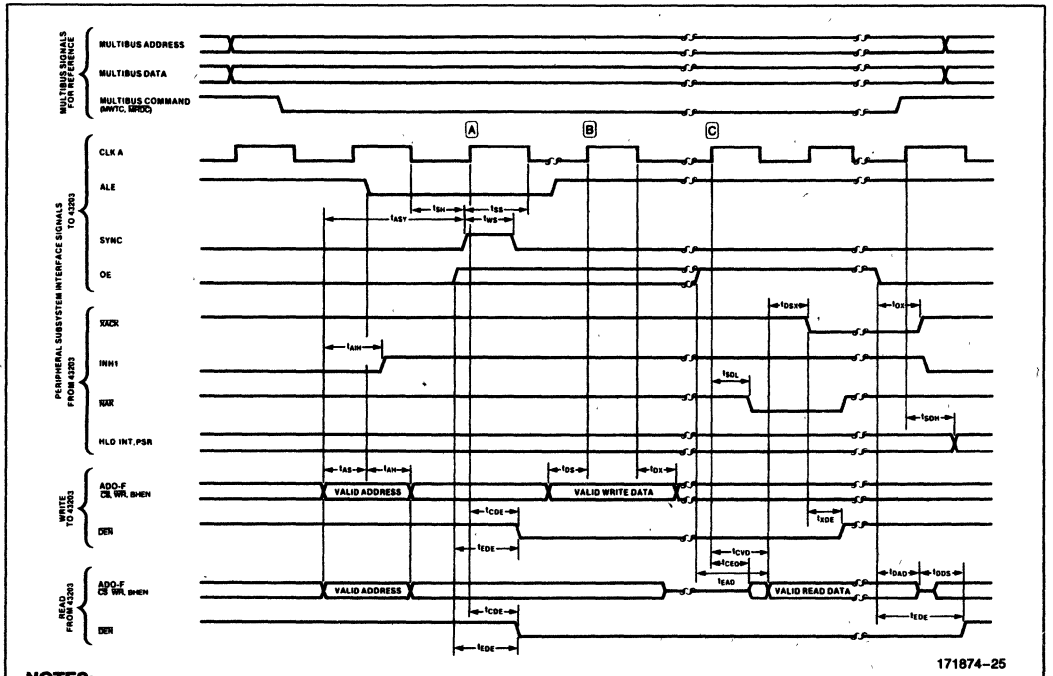


Figure 23. Local Processor Bus Timing



NOTES:

1. A and C could be the same clock edge for a buffered mode read.
2. B clock edge could be one or two clocks after clock edge A, depending on the value of the write sample delay.

Figure 24. Multibus® Interface Timing

INSTRUCTION SET

Table 9 compares the operators available in the IP's function set to those provided in the GDP's instruction set. Since windows are unique to Interface Processors, the ALTER MAP AND SELECT OBJECT function has no counterpart in the GDP. Conversely, the IP has no functions for performing arithmetic operations (except for the atomic function INDIVISIBLY ADD SHORT ORDINAL) or logical operations on numeric or character data types, nor does it have any operators to alter the flow of execution (e.g., branch or call functions).

To the extent that these classes of operators are needed in a peripheral subsystem interface, they can be provided by a combination of the AP's instruction set and the IP's window facility. By opening a window, for example, on a message received from a GDP process, the I/O controller can use AP instructions to test and branch on the value of a message filed read through the window.

Through its windows, an IP provides the basic ability to read and write the contents of the data parts of objects. Using its function request facility, however, an IP can manipulate Access Descriptors (ADs), which reference objects. The IP can examine a complex (multi-segment) object, gaining access to its component segments. It can also perform amplification on both hardware-recognized typed objects and software-recognized types. When manipulating objects of a protected type, an I/O controller is acting as a type manager, and its objects must be coordinated with the 432 type manager for the object.

The Interface Processor provides the I/O controller with both process and processor communication facilities. Interprocess communication is asynchronous and is performed with the aid of ports, system objects that provide synchronization and message queues. Any object may be sent as a message from a process to a port.

Interprocessor communication messages are predefined (see Table 10). The I/O controller can send one of these messages to an individual processor, or it can broadcast the message to all processors in the system.

The IP also provides an optimized data passing facility by using objects of type IP—Message and the OPEN MESSAGE and CLOSE MESSAGE operators.

Table 9. IP/GDP Operator Comparison

Operator	Implementation
WINDOW DEFINITION OPERATOR	
Alter Map and Select Object	IP
ACCESS DESCRIPTOR MOVEMENT OPERATORS	
Copy Access Descriptor	GDP + IP
Null Access Descriptor	GDP
Move to Embedded Data Value	Similar
RIGHTS MANIPULATION OPERATORS	
Amplify Rights	GDP + IP
Restrict Rights	GDP
TYPE DEFINITION MANIPULATION OPERATORS	
Retrieve Type Definition	GDP
REFINEMENT OPERATORS	
Create Refinement	GDP
Create Typed Refinement	GDP
OBJECT CREATION OPERATORS	
Create Object	GDP
Create Typed Object	GDP
ACCESS INSPECTION OPERATORS	
Inspect Access Descriptor	GDP + IP
Inspect Object	GDP + IP
Equal Access	GDP

Table 9. IP/GDP Operator Comparison (Continued)

Operator	Implementation
ACCESS INTERLOCK OPERATORS	
Lock Object	GDP + IP
Unlocked Object	GDP + IP
Indivisibly Add Short Ordinal	GDP + IP
Indivisibly Add Ordinal	GDP
Indivisible Insert Short Ordinal	Similar
Indivisible Insert Ordinal	GDP
CONTEXT OPERATORS	
Enter Environment	GDP + IP
Enter Global Environment	GDP + IP
Set Context Mode	GDP
Adjust Stack Pointer	GDP
Call	GDP
Call through Domain	GDP
Return	GDP
Return and Fault	GDP
PERIPHERAL SUBSYSTEM MODE OPERATOR	
Set Peripheral Subsystem Mode	IP
PROCESS COMMUNICATION OPERATORS	
Send	GDP + IP
Receive	GDP + IP
Conditional Send	GDP + IP
Conditional Receive	GDP + IP
Surrogate Send	GDP + IP
Surrogate Receive	GDP + IP
Delay Process	GDP
Send Process	GDP
Dispatch	IP
Set Process Mode	GDP
Read Process Clock	GDP
Open Message	IP
Close Message	IP
PROCESSOR COMMUNICATION OPERATORS	
Send to Processor	GDP + IP
Read Processor Status	GDP + IP
INTERCONNECT OPERATORS	
Move to Interconnect	GDP*
Move from Interconnect	GDP*
BLOCK—MOVE—OPERATORS	GDP
BRANCH OPERATORS	GDP
CHARACTER OPERATORS	GDP
SHORT-ORDINAL OPERATORS	GDP
SHORT-INTEGER OPERATORS	GDP
ORDINAL OPERATORS	GDP
INTEGER OPERATORS	GDP
SHORT-REAL OPERATORS	GDP
REAL OPERATORS	GDP
TEMPORARY REAL OPERATORS	GDP

Legend:

GDP + IP	IP and GDP implementations are identical.
IP	IP implements operator, GDP does not.
GDP	GDP implements operator, IP does not.
similar	While conceptually similar, IP implements operator differently than GDP.
*	Window 1 of IP provides equivalent interconnect access.

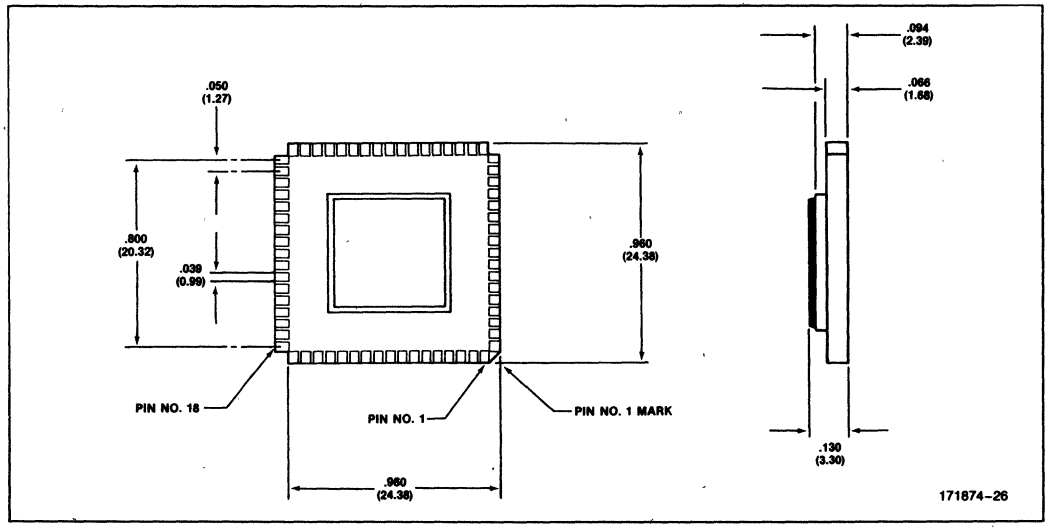
Table 10. IPC Message Codes

The following list contains the Interprocessor Communication message codes in decimal along with a short description of the message:

0 Wakeup
1 Start
2 Stop
3 Accept Global IPCs
4 Ignore Global IPCs
5 Requalify Object Table Cache
6 Reset Processor
7 Requalify Processor
8–14 Defined for GDP but ignored by IP

Package

The 43203 is packaged in a 68-pin, leadless JEDEC type A hermetic chip carrier as shown below.



Additional Information

More information about the iAPX 432 Micromainframe architecture can be found in the following publications:

- iAPX 432 Interface Processor Architecture Reference Manual (Order Number 171863)
- iAPX 432 General Data Processor Architecture Reference Manual (Order Number 171860)
- iAPX 432 Interconnect Architecture Reference Manual (Order Number 172487)

Information on the electrical characteristics of other 432 components can be found in the following publications:

- iAPX 43201/43202 General Data Processor Data Sheet (Order Number 590125)
- iAPX 43204/43205 Fault Tolerant Bus Interface and Memory Control Units (Order Number 210963)
- iAPX 43204/43205 BIU/MCU Electrical Specifications (Order Number 172867)

iAPX 43204, iAPX 43205
FAULT TOLERANT
BUS INTERFACE AND MEMORY CONTROL UNITS

- **Software Transparent Detection And Recovery From Any Single Point Failure**
- **Supports Up To 31 Processors For A Large Performance Range**
- **Configure From 1 To 8 Buses For High Bandwidth And Fault Tolerance**
- **Single, Dual, and Quad Redundant Configurations Tailor System Designs to Meet A Spectrum of Fault Tolerance And Cost Objectives**
- **VLSI System Simplifies Design With Low TTL Count**
- **Dynamic RAM Refresh with Error Correction and Scrubbing**

The 43204 Bus Interface Unit (BIU) and 43205 Memory Control Unit (MCU) are two VLSI devices that support the construction of fault tolerant, multiple processor 432 systems. Together they support: multiprocessor arbitration, dynamic RAM control, and ECC with a minimal amount of TTL. Fault tolerant systems can be built that tolerate the failure of any single component or bus. The BIU and MCU detect the failure and automatically switch to a redundant processor, bus, or memory. Hardware failures are completely masked from application software.

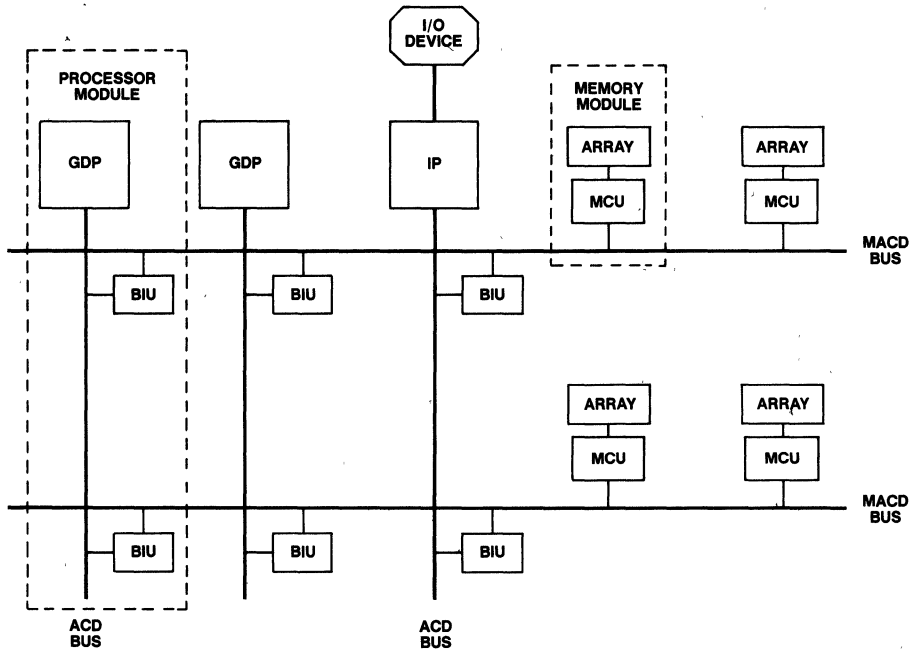


Figure 1. A 2 Bus System with 2 General Data Processors and 1 I/O Subsystem.

October 1983
ORDER NUMBER: 210963-002

INTRODUCTION

The first phase of the iAPX 432 program introduced two processor types: the General Data Processor (GDP) and the Interface Processor (IP). The GDP was implemented with two VLSI components: iAPX 43201 and iAPX 43202. The IP was implemented as a single VLSI component: the iAPX 43203. These three VLSI components implement the *processor architecture* for the iAPX 432. System builders have constructed multiple processor systems by surrounding the VLSI processors with discrete logic, which provided the interface to shared memory and the interprocessor communication paths. The method for interconnecting iAPX 432 processors and memories was unique for each system, since no standard had been defined.

This data sheet describes a pair of VLSI components: the iAPX 43204 Bus Interface Unit (BIU) and the iAPX 43205 Memory Control Unit (MCU) that form a unifying *interconnect architecture* for building iAPX 432 systems. Together, these components form the basis for constructing multiple-processor fault-tolerant iAPX 432 systems.

The iAPX 432 together with the BIU/MCU interconnect architecture provide:

- *Integrated fault tolerance.* The VLSI interconnect components (BIU/MCU) integrate all the detection and recovery logic required to build a system that can tolerate any single component failure.
- *Software transparent fault tolerance.* Hardware performs all fault detection and recovery functions transparent to application software. The machine never stops.
- *Configurability.* The BIU and MCU support a range of fault tolerance and performance options to meet a diverse set of cost, performance, and reliability needs.
- *Standard VLSI solution.* Very little external logic is required.
- *Reliable software.* The iAPX 432 system's "need to know" (capability) addressing confines errors, protecting the system from errant software.

The object-based architecture of the iAPX 432 provides a robust and flexible environment for cooperating, concurrent software systems. The iAPX 432 processors use a cooperating, self-dispatching mechanism to automatically share the workload between the available processors. The number of processors available in the system is transparent to software.

The BIU and the MCU extend the logical flexibility and robustness of the iAPX 432 processors into the physical implementation of iAPX 432 systems. The BIU and MCU allow the iAPX 432 hardware to modularly and transparently extend the processing power (from 1 to 63 modules of processors or memories), bus bandwidth (1 to 8 backplane buses), and fault-tolerant capabilities of the system. Figure 1 shows an example of a two bus three processor (2 GDPs + 1 IP) system.

As Figure 2 shows, an iAPX 432 system based on the interconnect architecture may be expanded gracefully. A system with one processor and one memory may be built with a single memory bus. Transparent multiprocessing may be achieved by simply adding processor modules. When additional memory is required, memory modules may be added onto the single memory bus. When more memory bandwidth is required, an additional memory bus(es) can be added. None of these alternative systems require any change to application software.

In an iAPX 432 system, each processor is unaware of the manner in which the memory address space is actually implemented. Hardware located in the BIUs determines how processor addresses are mapped to buses and memory systems.

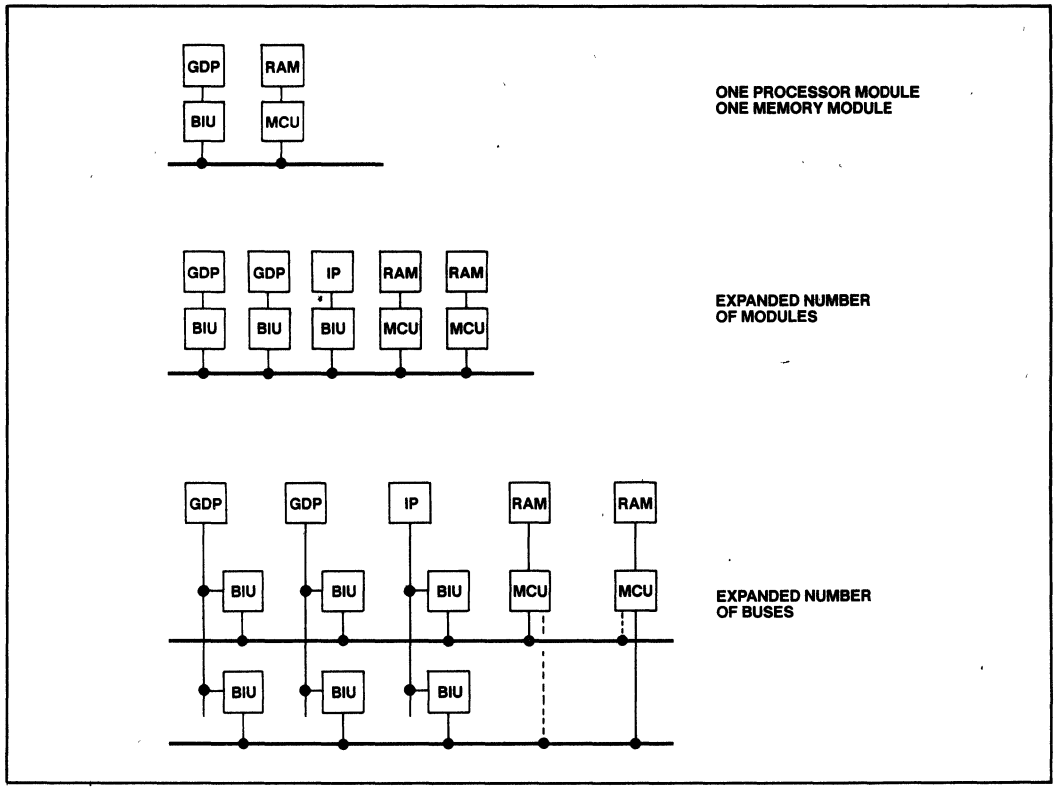


Figure 2. Modular Expansion

BUS INTERFACE UNIT

The Bus Interface Unit or BIU provides the switching function of the IAPX 432 interconnect system. That is, it accepts access requests from an iAPX 432 processor and, based on the physical address, it decides which memory bus(es) will be used to perform the access. The BIU is also responsible for arbitrating the usage of the memory bus. Finally, the BIU is responsible for propagating error information throughout the sytem.

MEMORY CONTROL UNIT

The Memory Control Unit or MCU interfaces memory storage arrays to the memory bus. The storage arrays will typically be constructed with high-density dynamic RAM (DRAM) components. All types of DRAMs are supported: 16K, 64K, 256K, even partially good components. The MCU manages the storage array as a logical collection of 32 data bits, 7 bits of error correcting code (ECC), and an optional spare bit. The MCU can automatically *refresh* the dynamic storage array. In addition, the MCU can *scrub* single-bit errors from the storage array as a background task. Scrubbing is accomplished by periodically reading the storage array, correcting all single-bit errors, and detecting and reporting all double-bit errors. The MCU accepts variable length data requests from the memory bus and performs the necessary access sequencing to read or write the data into the storage array. A modest amount of external logic is required to interface the MCU to the storage array RAMs — for simple configurations, as few as 12 external TTL packages are required.

MEMORY BUS

The memory bus (sometimes referred to as the MACD bus) provides the principal communication path, carrying all memory access requests and interprocessor communication. The memory bus connects BIUs to MCUs. Each node in the interconnect system tracks each operation on the memory bus to which it is attached. Thus, unlike most bus protocols, each BIU and MCU keeps track of all outstanding requests on the bus — not just the ones made by the BIU or MCU itself. Control for the bus is fully distributed; there is no centralized bus controller.

INTEGRATED FAULT TOLERANCE

BIUs and MCUs also form the basis for building *fault-tolerant* iAPX 432 systems. *Functional Redundancy Checking (FRC)* provides the low-level hardware support on which hardware fault-tolerant modules are constructed. In Figure 3, notice that a redundant processor module is formed by *replication* of the VLSI GDP and BIUs. A redundant memory module is formed by duplicating the VLSI MCU. The unshaded GDPs, BIUs, and MCUs act as *masters.* The shaded components act as *checkers,* which observe their master and report any disagreement they detect in the values the master produced.

When any error occurs, a special *error reporting network* notifies all nodes in the system of the discrepancy. Figure 4 illustrates the flow of error information in the interconnect system. In phase 1, an error is detected at a node in the interconnect system. The example illustrates an error detected at BIU(2,1); i.e., the BIU on memory bus 2 in processor

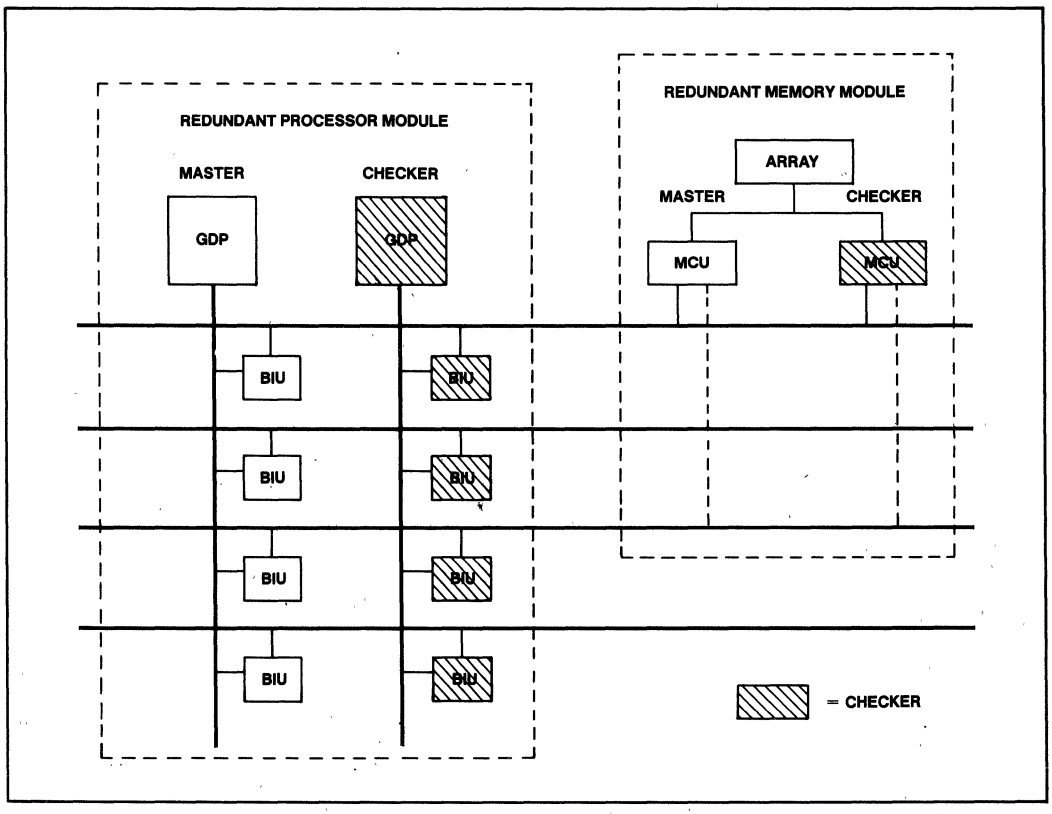


Figure 3. FRC Configuration Pairing

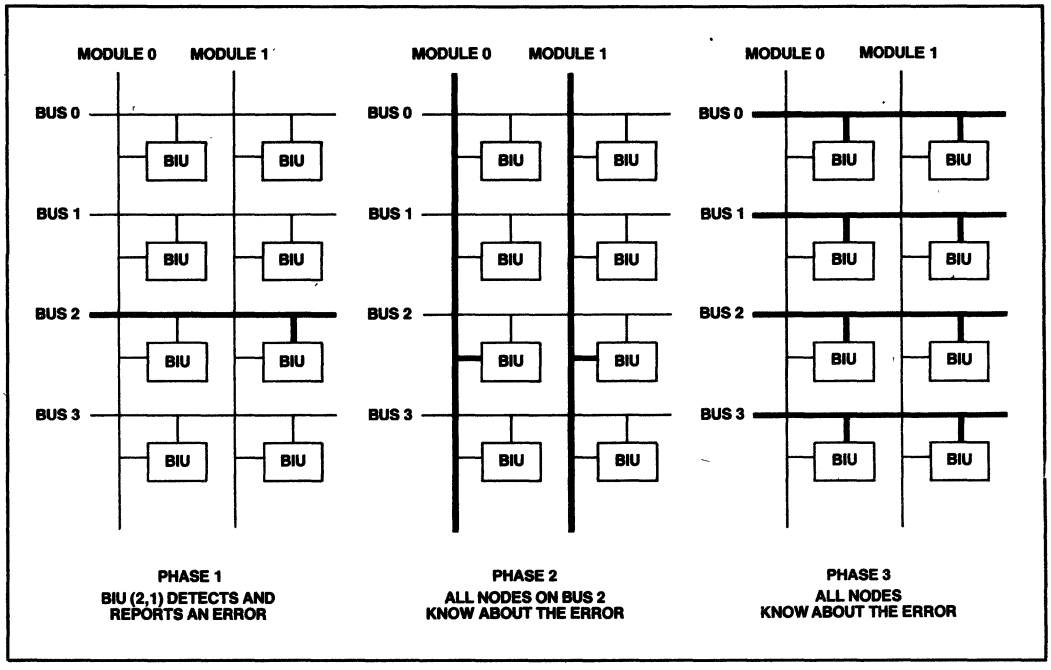


Figure 4. Three Phase Error Reporting Mechanism

module 1. The detecting component reports the error to all components attached to the same bus (a bold line indicates an active error reporting path). At this point, if all error reporting nodes are intact, all nodes have received the error message. In phase 2, all components that received the phase 1 error message *rebroadcast* the message along their module paths. Finally, in phase 3, each component that has received an error message rebroadcasts the message along its bus path. This second rebroadcast ensures that all nodes receive the error message even if a single module or bus error report line has failed. At the end of phase 3, all interconnect components in the system have been informed of the error. The actual error reporting paths are separate from, but run parallel to, the MACD and ACD busses so that error reports may propagate even if a bus is inoperative. In addition, the reporting paths may be duplicated to remove any single-point dependency in delivering an error report.

RECOVERY

The recovery process begins after an error report message has been broadcast around the system. Recovery is a distributed operation — each node in the system reads the error report message and decides what recovery actions need to be taken.

For recovery to be successful, there must be redundant resources available in the system. There are three redundancy mechanisms in the BIU and MCU: *bus retry buffers, ECC, and module shadowing.* The first two are useful in recovering from transient errors, while module shadowing allows recovery from permanent errors.

Figure 5 illustrates how every module in the system may be paired with another self-checking module of the same type. This pair of self-checking modules operates in lock step and provides a complete and current backup for all state information in the module. This mechanism is known as *module shadowing* because a shadow is ready to fill in if the primary fails, or vice versa. Fault detection and recovery is performed totally transparent to both application and system software. When the recovery is complete, system software is notified that a failure occurred. Figure 6 shows sample module failures and automatic hardware recovery.

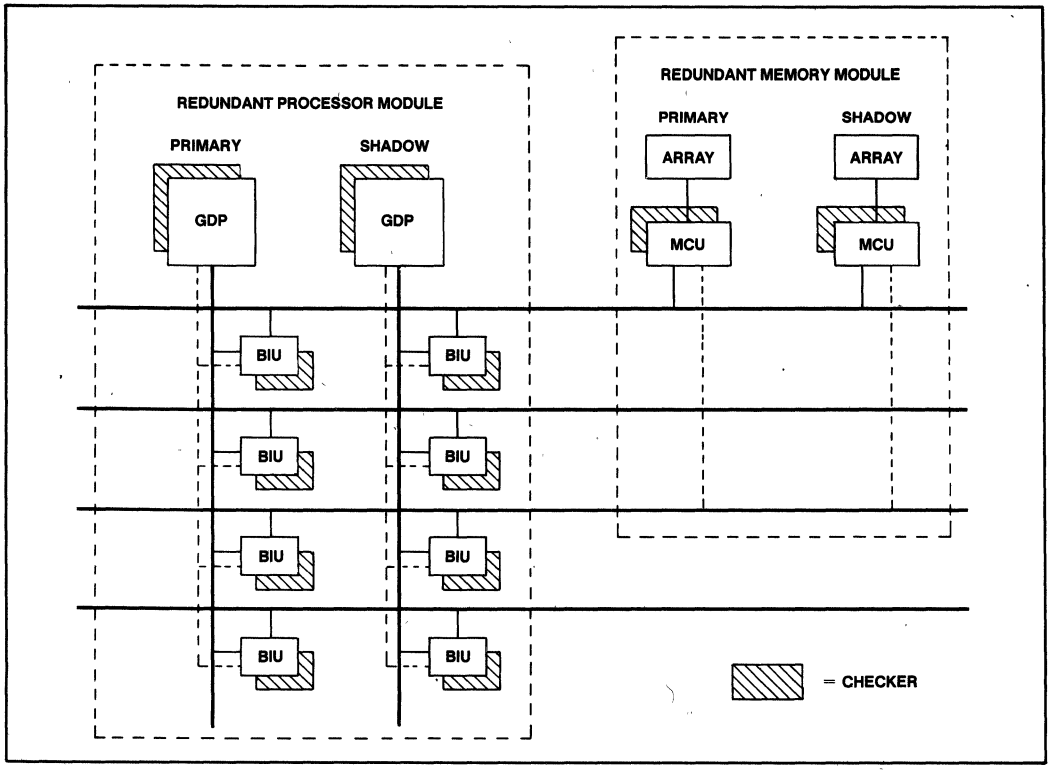


Figure 5. QMR Configuration Pairing

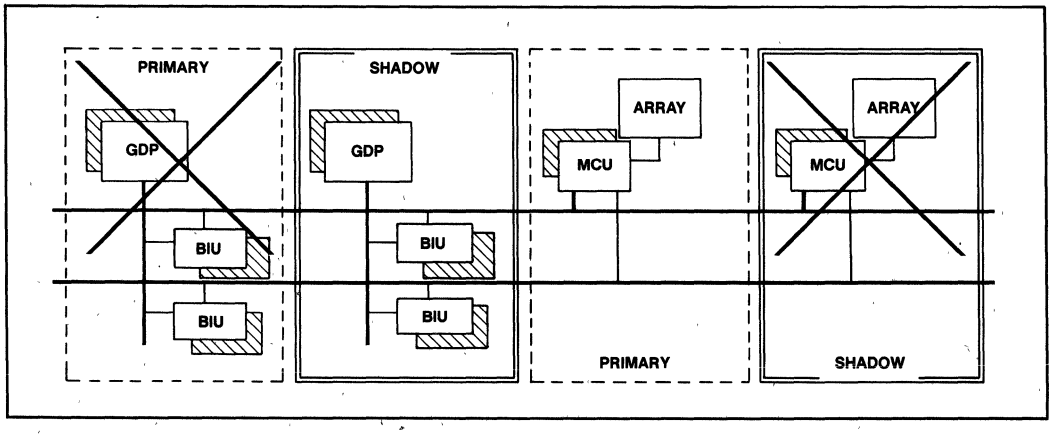


Figure 6A. Module Failures Are Detected

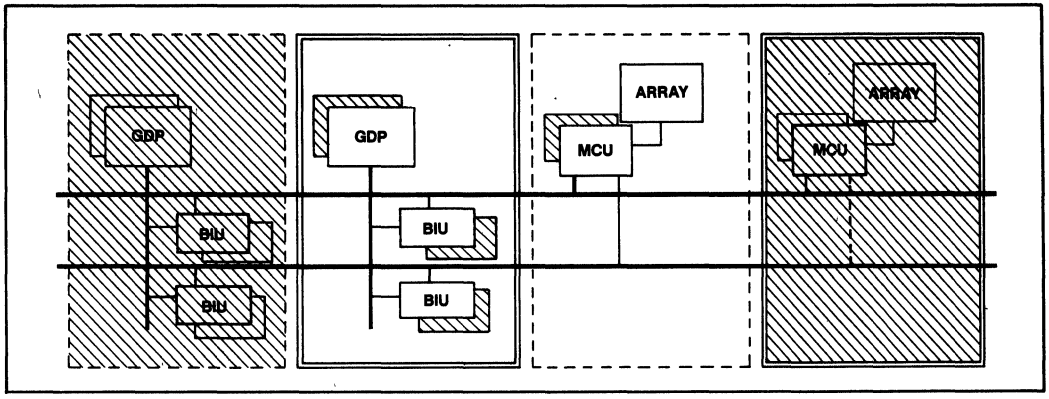


Figure 6B. Failed Modules Are Disabled

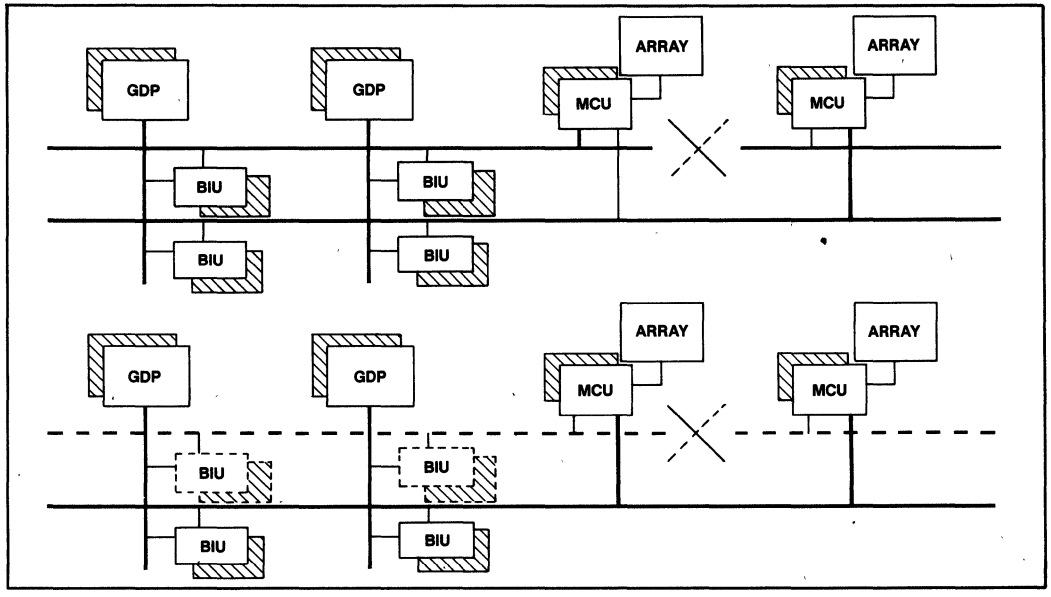


Figure 7. Bus Reconfiguration

A fault-tolerant module is also called a QMR module (Quad Modular Redundant) because most components (except memory) are replicated four times. There are two self-checking modules and each of these has a master and a checker.

Each memory bus in the system may be paired with another memory bus. Memory modules are physically connected to both buses although logically they are attached to only one bus at a time. During normal operation the buses run independently. Both contribute to the total memory bandwidth available in the system. If a bus fails, the memory modules attached to that bus will automatically switch to the other bus which is still operating. Figure 7 illustrates how the BIU and MCU reconfigure the system when a bus fails.

CONFIGURABLE FAULT TOLERANCE

Figure 8 illustrates the range of alternatives available to system designers when they build iAPX 432 systems. The most fault-tolerant systems are built from a QMR configuration of processors that can tolerate any single component failure without crashing the system. BIUs and MCUs provide full hardware error detection and recovery transparent to software.

The lowest cost configurations can be built using basic processor modules without FRC or QMR. This type of configuration will crash if a component fails, but can be made "self-healing" by adding intelligent software to the I/O subsystem. Unlike QMR, self-healing does not protect against system crashes, but it does allow the system to recover from a failure in a short period of time. The "healing" takes place in 3 steps. First, a watchdog timer in an I/O subsystem alerts I/O subsystem software that the central system has failed. Second, the I/O subsystem checks BIU/MCU error logging registers and runs diagnostics to identify which resource (e.g., processor, bus or memory) has failed. Third, the I/O subsystem reinitializes the system using the configuration control within the BIU and MCU to configure out the failed resource. The system is up and running without human intervention after only a short period of down time.

The basic configuration is the lowest cost alternative, but for some applications it is desirable to have a very high degree of confidence that calculations are performed correctly. A QMR system will do this since all components have a checker that alerts the system whenever a mistake is made. However, a QMR configuration may be overkill for some applications that can tolerate an occasional system failure, as long as the computations are correct when they do complete. FRC configurations offer an alternative in between the basic and QMR approaches. Adding a second set of checker components to each module improves the error detection capabilities of the system providing "high confidence" computing. No single hardware failure will go undetected and corrupt the results of a critical computation. FRC insures that any error is caught before it can propagate to another module in the system. FRC alone does not provide automatic hardware recovery like a QMR system, but it does detect errors as soon as they occur so that the system does not become corrupted. It is then the responsibility of system software to implement a "self-healing" strategy where the faulty resource is disabled and the system reinitialized.

The software configurability of a BIU/MCU system allows a system to use a combination of the above strategies. For example, software can configure a system as a full QMR system in the morning for critical applications, and then switch to an FRC only system in the afternoon. This doubles the system throughput (twice as many processors are working in parallel) without making any hardware changes.

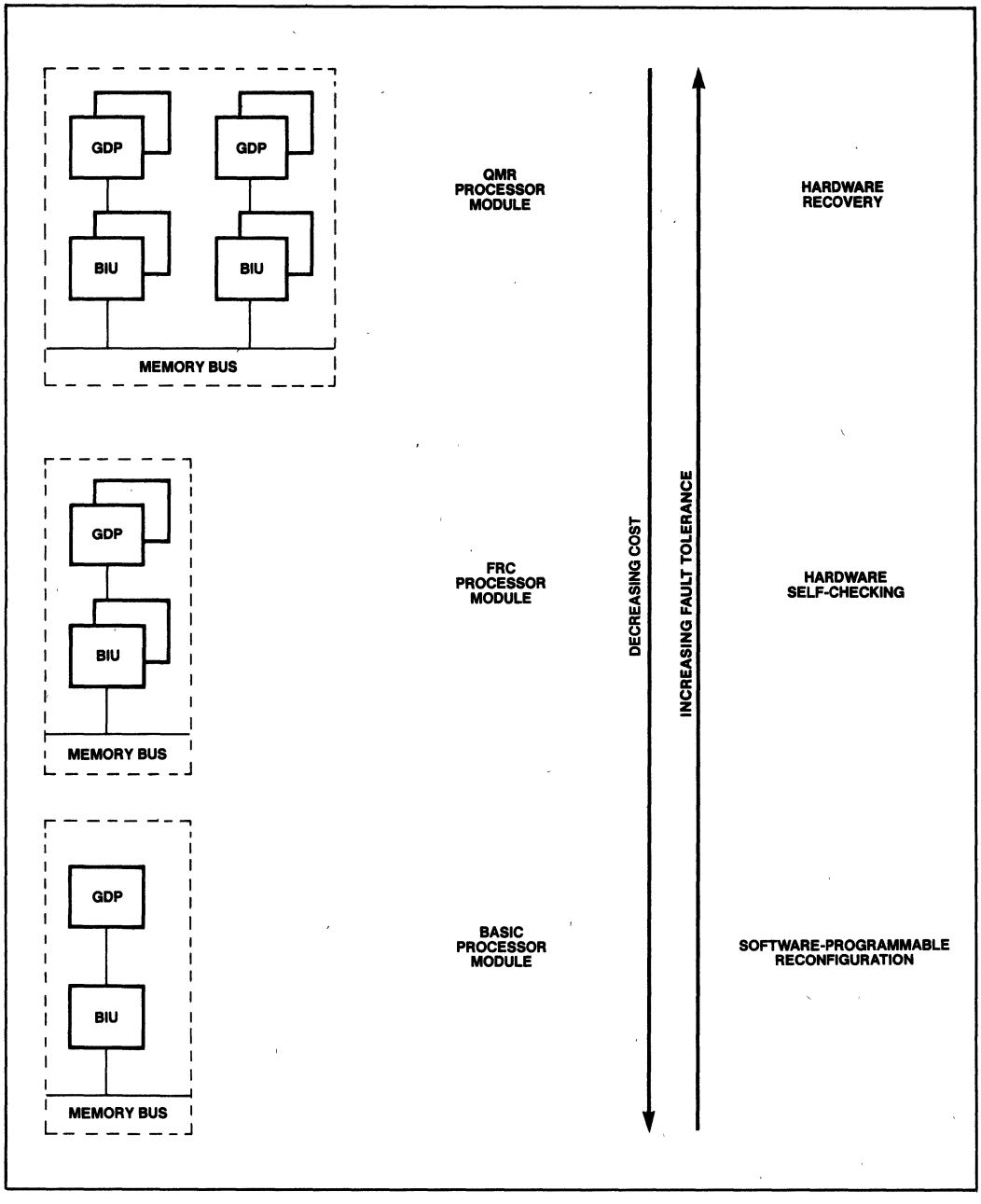


Figure 8. Fault-Tolerant Alternatives

FAULT-TOLERANT SYSTEM DESIGN RESPONSIBILITIES

The interconnect architecture and the VLSI components provide a stable base for developing fault-tolerant iAPX 432 systems. The iAPX 432 interconnect components address the issues concerning fault tolerance which are encountered when constructing the iAPX 432 central system. *A number of system-wide issues remain the responsibility of the iAPX 432 system designer.* These issues include:

- A fault-tolerant I/O system
- Fault-tolerant power supplies and distribution method
- A fault-tolerant method for clock generation and distribution
- The electrical and physical provisions for on-line repair

SUMMARY

The iAPX 432 interconnect architecture provides a standard VLSI method for constructing multiple processor VLSI computer systems. The iAPX 432 interconnect architecture is implemented by a pair of VLSI components, the Bus Interface Unit (BIU) and the Memory Control Unit (MCU). Together with iAPX 432 processors, these components permit the construction of modular, extensible, multiprocessor computer systems. The components are designed to support the construction of fully fault-tolerant iAPX 432 systems. However, there is no penalty in performance or in cost for those applications that do not require fault tolerance.

The 432 fault-tolerant mechanisms are designed to provide a flexible and complete solution to the problems of fault-tolerant hardware. For basic systems (those without checkers for error detection or QMR for recovery), a user may decide to use only a few detection mechanisms and provide recovery only for transient errors. This functionality comes at no additional cost in the VLSI interconnect system. To reduce maintenance cost and increase system availability, a system may use all of the detection mechanisms (i.e. may add checker components) but may not add any extra recovery capability (i.e. may not marry self-checking modules into a fault-tolerant QMR module). Continuous operation is available to the user who adds the extra recovery capabilities.

None of the fault-tolerant mechanisms reduce system performance. Systems that do not require the highest level of fault tolerance are not penalized in any way (cost, size, or performance) for the unused fault-tolerant capabilities. Increased levels of fault tolerance are achieved by replicating the iAPX 432 VLSI components. The hardware fault tolerance in the iAPX 432 is transparent to application software. The system's fault-tolerant capabilities may be changed without any changes to the application software system.

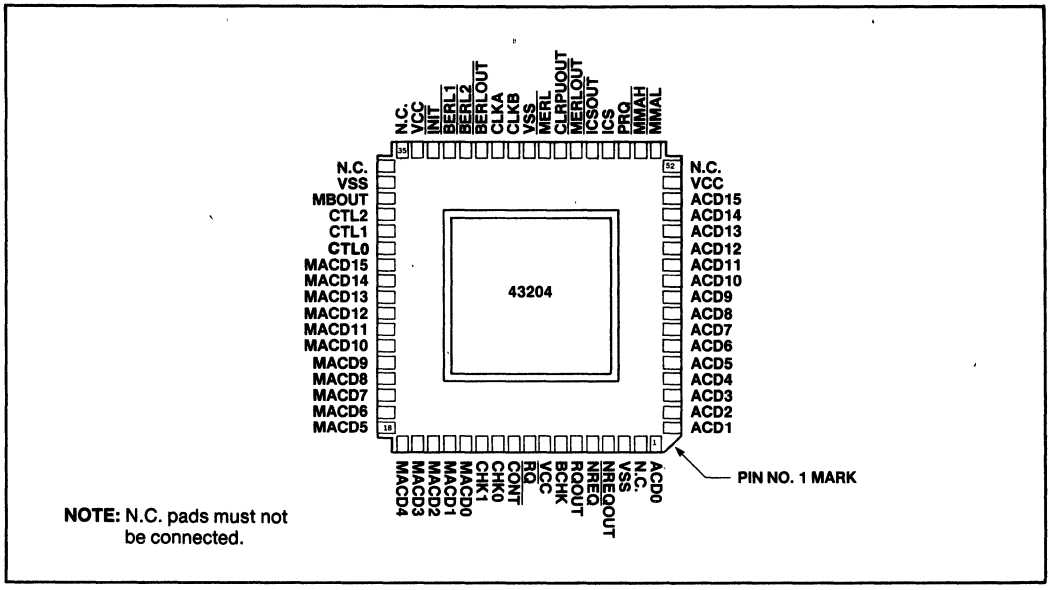


Figure 9. 43204 Pin Configuration

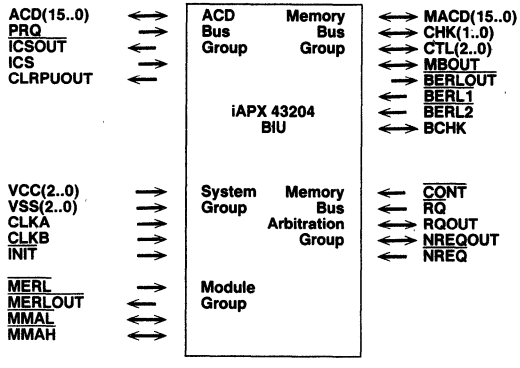


Figure 10. iAPX 43204 BIU Logic Symbol

Table 1. iAPX 43204 BIU Pin Description

Symbol	Type	Name and Function
Memory Bus Group		
MACD15 . . MACD0	I/O*	These 16 bidirectional signals carry physical memory addresses, control information (access length and type), and data to and from the memory bus.
CHK1..CHK0	I/O*	These 2 bidirectional signals carry parity bit check information which detects errors in transfers on MACD15 . . MACD0 and CTL2 . . CTL0. The 2 parity check bits are computed to satisfy the following equations (X = Exclusive OR): MACD15 X MACD13 X MACD11 X MACD9 X MACD7 X MACD5 X MACD3 X MACD1 X CTL1 X CHK1 = 0 MACD14 X MACD12 X MACD10 X MACD8 X MACD6 X MACD4 X MACD2 X MACD0 X CTL2 X CTL0 X CHK0 = 1 The BIU and MCU generate and check even parity (an even number of ones) across the 10 odd-numbered MACD, CTL, and CHK signals, and odd parity (an odd number of ones) across the 11 even-numbered MACD, CTL, and CHK signals.
CTL2..CTL0	I/O*	The 3 MACD bus control signals carry a code that controls the sequencing of the memory bus.
MBOUT	I/O**	MBOUT controls the direction of external buffers for the MACD, CHK, and CTL signals. When MBOUT is asserted, it indicates that the buffers must be directed to carry information outbound from the component to the memory bus.
$\overline{\text{BERLOUT}}$	0	$\overline{\text{BERLOUT}}$ supplies bit-serial bus error messages when the component detects a memory bus error, a storage array error, or a memory module error.
$\overline{\text{BERL1}}$, $\overline{\text{BERL2}}$,	I I	$\overline{\text{BERL1}}$ and $\overline{\text{BERL2}}$ are duplicate paths on which the component receives bit-serial bus error messages from the memory bus. When duplicated paths are not required, these two pins must be supplied with the same bus error report information.
BCHK	I/O*	BCHK provides a mechanism which checks that external buffers are operating. BCHK is toggled once each clock cycle by the component that is driving it. In an FRC pair, the master component drives BCHK. The checker component in the FRC pair receives BCHK. Routing BCHK from the master component, through one buffer in each external buffer package, and to the checker component, forms a serial network. If the oscillating BCHK signal fails to traverse the external buffer network, the buffer path is suspect and a bus error will be signalled. Buffer checking can be disabled by interconnect register programming.

Table 1. iAPX 43204 BIU Pin Description (Continued)

Symbol	Type	Name and Function
Memory Bus Arbitration Group		
$\overline{CONT}$	I	The $\overline{CONT}$ input indicates if the external arbitration network has detected that two or more simultaneous requests have been made for the use of the memory bus. When contention is indicated, all contending components will perform a binary arbitration sequence (based on each component's unique 6-bit module ID) to decide which component will be granted first use of the memory bus.
$\overline{RQ}$	I	The $\overline{RQ}$ input indicates if any agent is requesting the use of the memory bus. There are three valid combinations for $\overline{RQ}$ and $\overline{CONT}$: **$\overline{RQ}$ $\overline{CONT}$ Interpretation** 1 1 No request 0 1 One BIU is making a request 0 0 Two or more BIUs are making a request The MCU does not generate any memory bus requests. The MCU tracks the action of $\overline{RQ}$ and $\overline{CONT}$, and the CTL(2 . . 0) signals to determine when it is allowable to use the memory bus to reply to a request.
RQOUT	I/O**	When asserted, RQOUT signals that the component requires the use of the memory bus. RQOUT is intended to drive an external open-collector inverter which is wire-ORed to form a combined $\overline{RQ}$ line. The RQOUT signal from all BIUs attached to a memory bus must be logically combined to form a contention signal. (Contention occurs when two or more BIUs issue RQOUT simultaneously). The logic to detect contention among BIUs must be supplied by the customer.
NREQOUT	I/O*	The NREQOUT signal indicates that the component has received a new request from its associated processor. NREQOUT is intended to drive an external open-collector inverter, which is wire-ORed (with the same signal from other BIUs on the memory bus) to form $\overline{NREQ}$.
$\overline{NREQ}$	I	$\overline{NREQ}$ is an input that signals the beginning of a new time-ordered request cycle in which a request from one or more processors must be managed.

Table 1. iAPX 43204 BIU Pin Description (Continued)

Symbol	Type	Name and Function
Module Group		
$\overline{\text{MERL}}$	I	The $\overline{\text{MERL}}$ input accepts bit-serial module error messages. See the $\overline{\text{BERLOUT}}$ pin description for the format of the serial error messages.
$\overline{\text{MERLOUT}}$	O	The $\overline{\text{MERLOUT}}$ output broadcasts bit-serial module error messages to all BIUs contained within the same module (attached to the same processor).
$\overline{\text{MMAL}}$	I/O+	$\overline{\text{MMAL}}$ operates in the same manner as $\overline{\text{MMAH}}$ except that when $\overline{\text{MMAL}}$ is asserted it indicates that the lower addressed portion of a multiple module access is in progress on the memory bus. The two BIUs that are cooperating in a multiple module access observe both $\overline{\text{MMAH}}$ and $\overline{\text{MMAL}}$. Both signals are deasserted after each BIU has completed its portion of the access on the memory bus to which it is connected. In read accesses, after both signals are deasserted, the BIU with the lower addressed portion of the access presents data to the processor first. The BIU with the higher addressed portion of the access tracks the other BIU by counting the number of bytes returned to the processor (noting ICS, Interconnect Status, see below). When the BIU with the lower-addressed portion of the access completes its transfer, the next BIU begins automatically. Multiple module read accesses which begin at an odd addressed byte boundary cause the two cooperating BIUs to simultaneously return data to the processor. At the address boundary for which they share access responsibility, the low BIU returns its last byte on ACD7 . . ACD0, and the high BIU returns its first byte on ACD15 . . ACD8. After $\overline{\text{MMAH}}$ and $\overline{\text{MMAL}}$ have been deasserted, one (or both) of the BIUs may reassert the signals, each to indicate that its portion of the multiple module access encountered an error. This indication will be returned to the processor during error significance time for ICS.
$\overline{\text{MMAH}}$	I/O+	When asserted, $\overline{\text{MMAH}}$ indicates that one of the BIUs in a module is performing the high order address part of a multiple module access. A multiple module access occurs when a processor request spans an address range such that two memory buses, each connected via a different BIU must be engaged. When it is deasserted, $\overline{\text{MMAH}}$ indicates that the high portion of the access has been completed on the memory bus.

Table 1. iAPX 43204 BIU Pin Description (Continued)

Symbol	Type	Name and Function
ACD Bus Group		The ACD Bus Group contains the set of signals with which a compatible iAPX 432 processor connects to the BIU. See the iAPX 43201/43202 General Data Processor Data Sheet (Order Number 590125) and the iAPX 43203 Interface Processor Data Sheet (Order Number 590130) for information about compatible iAPX 432 processors.
ACD15 . . ACD0	I/O	These 16 signals form the processor-to-BIU communication path that carries all memory and interconnect accesses.
PRQ	I	PRQ indicates the start of a processor request to the BIU.
$\overline{\text{ICSOUT}}$	O	$\overline{\text{ICSOUT}}$ is intended to drive an external open-collector inverter to form ICS. All BIUs in a processor module contribute to the wired-OR ICS signal.
ICS	I	ICS supplies interconnect status to both the BIU and its associated iAPX 432 processor. ICS carries information on errors, data synchronization, and interprocessor communication to the processor. It is also monitored by each BIU for coordinating multiple module accesses.
CLRPUOUT	O	CLRPUOUT is intended to drive an open collector inverter and form a wired-OR $\overline{\text{CLR}}$ signal to the associated iAPX 432 processor. All BIUs in a processor module contribute to the wired-OR $\overline{\text{CLR}}$ signal. Using CLRPUOUT, a BIU can synchronize the FRC master and checker processor components.
System Group		
VCC2..VCC0		Three VCC pins supply 5-volt power to the BIU/MCU. All three pins must be connected. The three VCC pins are not connected together inside the component.
VSS2..VSS0		Three VSS pins provide ground to the BIU/MCU. All three pins must be connected. The three VSS pins are not connected together inside the component.
CLKA	I	CLKA is a square-wave clock for the BIU/MCU. CLKA must operate continuously to preserve the operating state of the component.
CLKB	I	CLKB is a square-wave clock for the BIU/MCU. CLKB is the same frequency as CLKA but lags CLKA by 90 degrees. CLKB must operate continuously to preserve the operating state of the component.
$\overline{\text{INIT}}$	I	$\overline{\text{INIT}}$ is a signal that causes the BIU/MCU to initialize. In addition, $\overline{\text{INIT}}$ is used to enable external logic which provides configuration information to the component.

Legend:
I = Input signal
O = Output signal
I/O = Input/Output signal
* = FRC errors cause module error
** = FRC errors cause bus error
$\pm$ = External passive pullup required (10K Ohms)
$\underline{\quad}$ = Asserted low

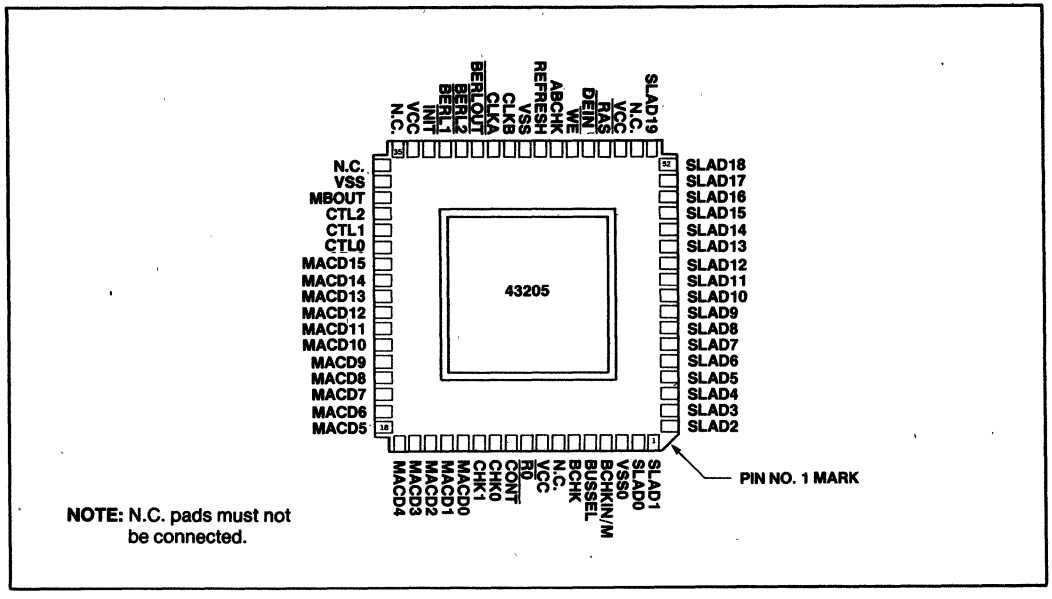


Figure 11. 43205 Pin Configuration

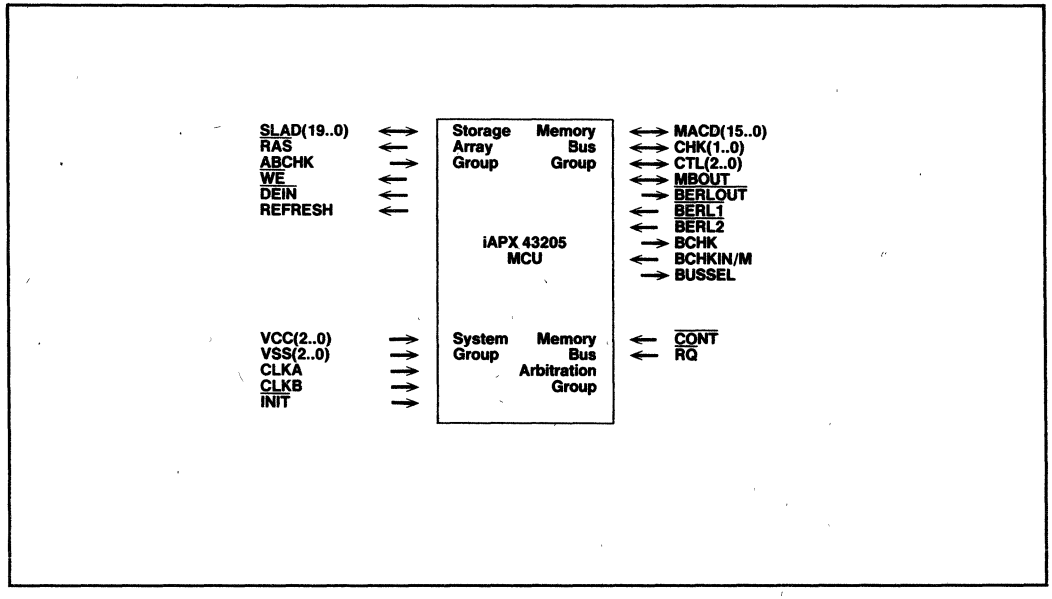


Figure 12. iAPX 43205 MCU Logic Symbol

Table 2. iAPX 43205 MCU Pin Description

Symbol	Type	Name and Function
Memory Bus Group		
MACD15 . . MACD0	I/O*	These 16 bidirectional signals carry physical memory addresses, control information (access length and type), and data to and from the memory bus.
CHK1..CHK0	I/O*	These 2 bidirectional signals carry parity bit check information which detects errors in transfers on MACD15 . . MACD0 and CTL2 . . CTL0. The 2 parity check bits are computed to satisfy the following equations (X = Exclusive OR): MACD15 X MACD13 X MACD11 X MACD9 X MACD7 X MACD5 X MACD3 X MACD1 X CTL1 X CHK1 = 0 MACD14 X MACD12 X MACD10 X MACD8 X MACD6 X MACD4 X MACD2 X MACD0 X CTL2 X CTL0 X CHK0 = 1 The BIU and MCU generate and check even parity (an even number of ones) across the 10 odd-numbered MACD, CTL, and CHK signals, and odd parity (an odd number of ones) across the 11 even-numbered MACD, CTL, and CHK signals.
CTL2..CTL0	I/O*	The 3 MACD bus control signals carry a code that controls the sequencing of the memory bus.
MBOUT	I/O**	MBOUT controls the direction of external buffers for the MACD, CHK, and CTL signals. When MBOUT is asserted, it indicates that the buffers must be directed to carry information outbound from the component to the memory bus.
$\overline{\text{BERLOUT}}$	O	$\overline{\text{BERLOUT}}$ supplies bit-serial bus error messages when the component detects a memory bus error, a storage array error, or a memory module error.
$\overline{\text{BERL1}}$ $\overline{\text{BERL2}}$	I I	$\overline{\text{BERL1}}$ and $\overline{\text{BERL2}}$ are duplicate paths on which the component receives bit-serial bus error messages from the memory bus. When duplicated paths are not required, these two pins must be supplied with the same bus error report information.
BCHK	O*	BCHK provides a mechanism which checks that external buffers are operating. BCHK is toggled once each clock cycle by the component. By routing BCHK through one buffer in each external buffer package to BCHKIN/M, a serial network is formed. If the oscillating BCHK signal fails to traverse the external buffer network, BCHKIN/M will detect the error and signal a bus error. The BCHK signal does not toggle when the component is being initialized by either the $\overline{\text{INIT}}$ signal or an internal initialization request. Buffer checking can be disabled by a parameter acquired by the MCU during initialization. The MCU will disable buffer checking after it detects a permanent module error.
BCHKIN/M	I	BCHKIN/M checks the oscillating BCHK signal after it has been routed through each of the external buffers for the memory bus. If any errors are detected, a module error will be signalled. During initialization, the BCHKIN/M pin accepts the MASTER information. If it is high during initialization, then the component will become the master of an FRC pair of components; otherwise it will become a checker.

Table 2. iAPX 43205 MCU Pin Description (Continued)

Symbol	Type	Name and Function
Memory Bus Group		
BUSSEL	O	BUSSEL controls which of two memory buses (normal or backup) the MCU is to use. The middle bit of an internal 3-bit normal bus identifier (ID) is logically combined with an internal bus state code to produce BUSSEL. The bus state code records the state and health of both the normal and backup buses. When the component switches to an alternate bus (changes the bus state code), BUSSEL is changed accordingly.
Memory Bus Arbitration Group		
$\overline{CONT}$	I	The $\overline{CONT}$ input indicates if the external arbitration network has detected that two or more simultaneous requests have been made for the use of the memory bus. When contention is indicated, all contending components will perform a binary arbitration sequence (based on each component's unique 6-bit module ID) to decide which component will be granted first use of the memory bus.
$\overline{RQ}$	I	The $\overline{RQ}$ input indicates if any agent is requesting the use of the memory bus. There are three valid combinations for $\overline{RQ}$ and $\overline{CONT}$: **$\overline{RQ}$ $\overline{CONT}$ Interpretation** 1 1 No request 0 1 One BIU is making a request 0 0 Two or more BIUs are making a request The MCU does not generate any memory bus requests. The MCU tracks the action of $\overline{RQ}$ and $\overline{CONT}$, and the CTL(2 . . 0) signals to determine when it is allowable to use the memory bus to reply to a request.
Storage Array Group		
SLAD19. . SLAD0	I/O*	The 20 SLAD signals form the communication path between the MCU and its associated storage array. The SLAD bus multiplexes addresses to the storage array with data (32 bits) and ECC (7 bits) which are to be read from or written to the array.
$\overline{RAS}$	O*	When $\overline{RAS}$ is asserted, it indicates the start of a storage array cycle. $\overline{RAS}$ may combine with external sequencing logic to control the operation of the storage array.
ABCHK	I	ABCHK is an input used to verify the external RAM control logic. WE and CAS are used to generate the ABCHK signal. In addition, the functionality of the external buffers associated with the storage array may be validated by routing the oscillating BCHK signal through each of the buffers in a similar manner as on the MACD bus side of the MCU. If an error is detected, ABCHK can be corrupted and in this fashion report the error. An alternate method of checking the storage array buffers is to use buffer packages with no more than four buffers per package so that the special ECC protection in the MCU may detect buffer failures.
$\overline{WE}$	O*	When $\overline{WE}$ is asserted, the MCU indicates that a write operation is to be performed in the storage array.

Table 2. iAPX 43205 MCU Pin Description (Continued)

Symbol	Type	Name and Function
Memory Bus Arbitration Group		
$\overline{DEIN}$	O*	When $\overline{DEIN}$ is asserted, the MCU indicates that the SLAD19 . . SLAD0 signals are ready to accept information from the storage array into the MCU.
REFRESH	O*	When REFRESH is asserted the MCU indicates that the storage array cycle is a refresh cycle. In systems with multiple bank dynamic RAM storage arrays, the REFRESH signal may be used to command the storage array sequencing logic to perform an appropriate cycle (e.g., RAS-only refresh for all banks). In a storage array with a single bank of dynamic RAMs the REFRESH signal need not be used.
System Group		
VCC2..VCC0		Three VCC pins supply 5-volt power to the BIU/MCU. All three pins must be connected. The three VCC pins are not connected together inside the component.
VSS2..VSS0		Three VSS pins provide ground to the BIU/MCU. All three pins must be connected. The three VSS pins are not connected together inside the component.
CLKA	I	CLKA is a square-wave clock for the BIU/MCU. CLKA must operate continuously to preserve the operating state of the component.
CLKB	I	CLKB is a square-wave clock for the BIU/MCU. CLKB is the same frequency as CLKA but lags CLKA by 90 degrees. CLKB must operate continuously to preserve the operating state of the component.
$\overline{INIT}$	I	$\overline{INIT}$ is a signal that causes the BIU/MCU to initialize. In addition, $\overline{INIT}$ is used to enable external logic which provides configuration information to the component.

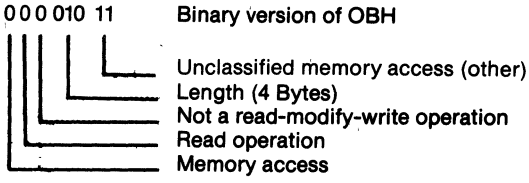
Legend: I = Input signal
O = Output signal
I/O = Input/Output signal
* = FRC errors cause module error
** = FRC errors cause bus error
$\overline{\quad}$ = Asserted low

iAPX 43204 FUNCTIONAL DESCRIPTION

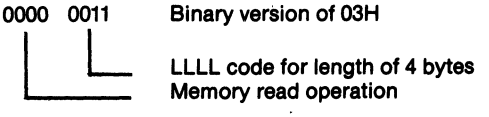
This section describes how the iAPX 43204 Bus Interface Unit operates through a set of functional diagrams that trace the operation of the pins clock-by-clock. To understand the notation on the various waveforms, refer to Figure 13. It illustrates the general operation of the BIU as it accepts a memory read request from a processor, forwards the request to a Memory Control Unit (MCU), and returns the reply data to the processor.

The cycle numbers (0, 1, 2, . . .) at the top of each diagram enumerate the clock cycles. When a common group of signals is plotted, its value is displayed inside a data waveform. For example, the BIU signals MACD15 . . . MACDO are all high in cycles 0 through 4. In cycle 5, the MACD bus carries the value 0300H and in cycle 6 the value 0000H.

Notice that in Figure 13, the BIU receives a processor request (PRQ) in cycle 0. The ACD bus carries the information 0B00H in cycle 1 (the specification field on the high-byte and addresses A7 . . . A0 on the low-order byte) and 0000H in cycle 2 (the address bits A23 . . . A8) for the access. Reorganizing the information, it is apparent that the BIU has been provided a processor request with a specification field of OBH and a 24-bit physical address of 000000H. Referring to the GDP or IP data sheets, the specification field can be decoded as follows:



Once the processor has presented the request to the BIU, the BIU presents ICS (Interconnect Status) to the processor indicating that the processor is to wait for the data to be returned by the BIU and the memory system. In cycle 4, the BIU issues the NREQOUT signal (not shown) and observes NREQ and RQ immediately. Though not shown, no contention with any other processor is observed on the CONT pin. In cycle 5, the BIU asserts MBOUT to control its external TTL buffers to drive the memory bus and presents a two-cycle memory read request message. The MACD bus carries the memory bus specification code (high-order byte) and the physical memory address bits (A7 . . . A0) in cycle 5, and the remaining 16 address bits (A23 . . . A8) in cycle 6. Referring to the Memory Bus appendix of the Interconnect ARM, the memory bus specification field of 03H is decoded as follows:



In cycles 10 and 11, the MCU that serviced the request presents the data that it read from the storage array, least significant bytes first. The returned bytes in the example are: 2AH, 03H, 9AH, 8CH. In cycles 15 and 16, the BIU presents the same data to the processor and indicates the availability of each byte with ICS.

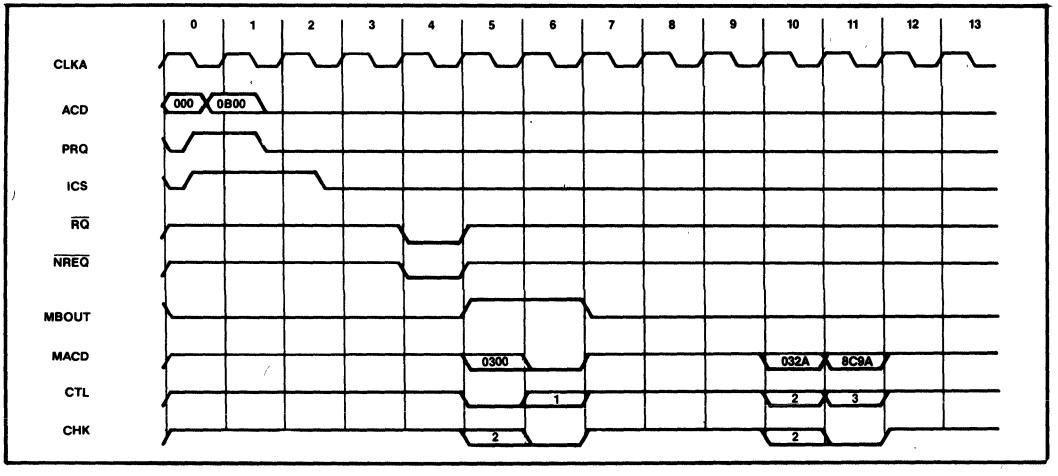


Figure 13. 4-Byte Memory Read

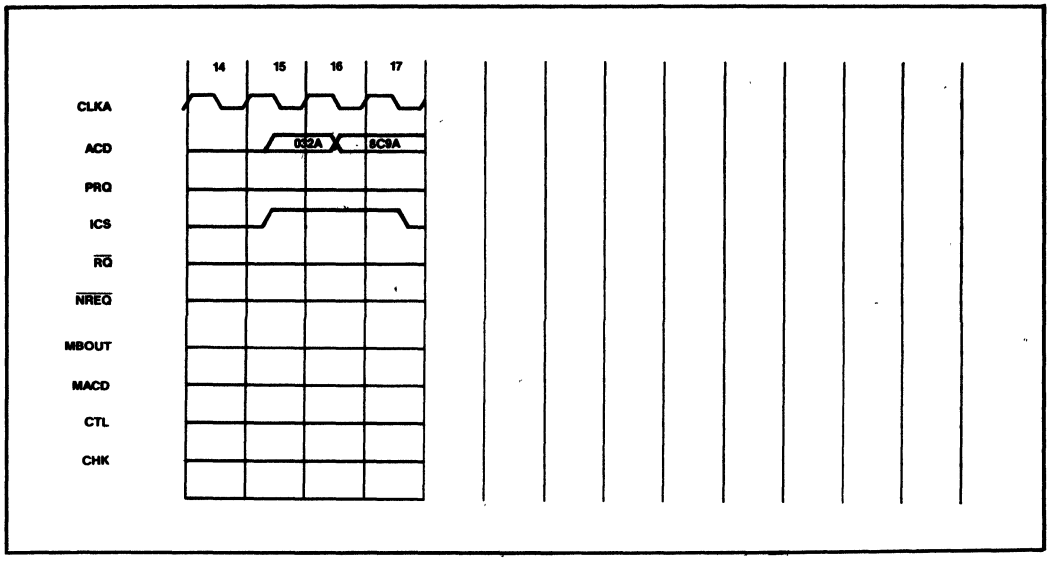


Figure 13. 4-Byte Memory Read (continued)

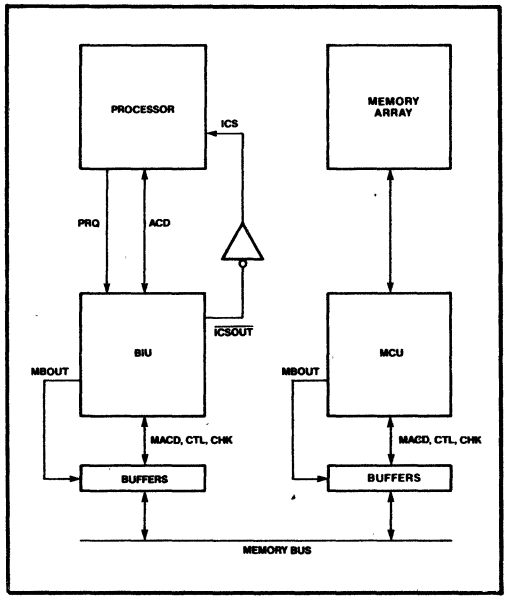


Figure 14. Hardware Configuration for Memory Read/Write Examples

Figure 15 illustrates the steps that occur when a processor requests a 4-byte write at physical memory address 0. The hardware system for this example includes one processor, one BIU, one memory bus, one MCU, and one storage array (see Figure 15). In cycle 0, the processor emits the processor request signal (PRQ=1) along with the specification field and the low-order address byte on the ACD signals. The specification field (high-byte of the first double-byte) in this example is 4BH and the low-order address byte (low-byte of the first double-byte) is 00H. In cycle 1, the processor emits the high- and mid-order address bytes on the ACD signals. In this example, the high- and mid-order address ;bytes are 00H. The BIU deasserts ICSOUT (not shown) which, after an external logical inversion, provides the ICS signal (Interconnect Status) to the processor. The deasserted ICS is used to stretch the processor access cycle (generate wait states). The processor presents the first double-byte of data to be written in cycle 4, but because ICS is deasserted, the processor must stretch the data into cycle 5 before the BIU will accept it. In cycle 5, the processor provides the second double byte of data to be wirtten (0000H). Notice that the BIU deasserts ICS to hold the processor until the entire request is actually satisfied by the MCU and memory array.

In cycle 4, the BIU presents the request to the memory bus arbitration logic. Since only one processor is in the system, the NREQ (output) and RQ input signals are asserted but no contention (CONT signal is not shown) is detected. Thus, the write access proceeds immediately to the memory bus (MACD, CTL, and CHK signals). In cycles 5 through 8, MBOUT is asserted by the BIU to direct external buffers to pass the BIU request to the memory bus. The first two double-bytes of the request contain a specification byte and three address bytes. The information in these two double-bytes is a modified version of that received from the processor ACD bus. The high-order byte of the memory address is contained in the low-order byte of the first double-byte. The mid- and low-order bytes of the memory address are contained, respectively, in the high-order and low-order bytes of the second double byte. MBOUT remains asserted while the two double-bytes of the write request are provided by the BIU. The MCU performs the access and returns a write reply (CTL=6) in cycle 10. The BIU asserts ICS for the processor in cycle 14, allowing the processor to escape from the stretched write cycle that it had entered earlier. In this case, no error occurred during the access. However, had there been an error, the level of ICS in cycle 15 would indicate the error to the processor.

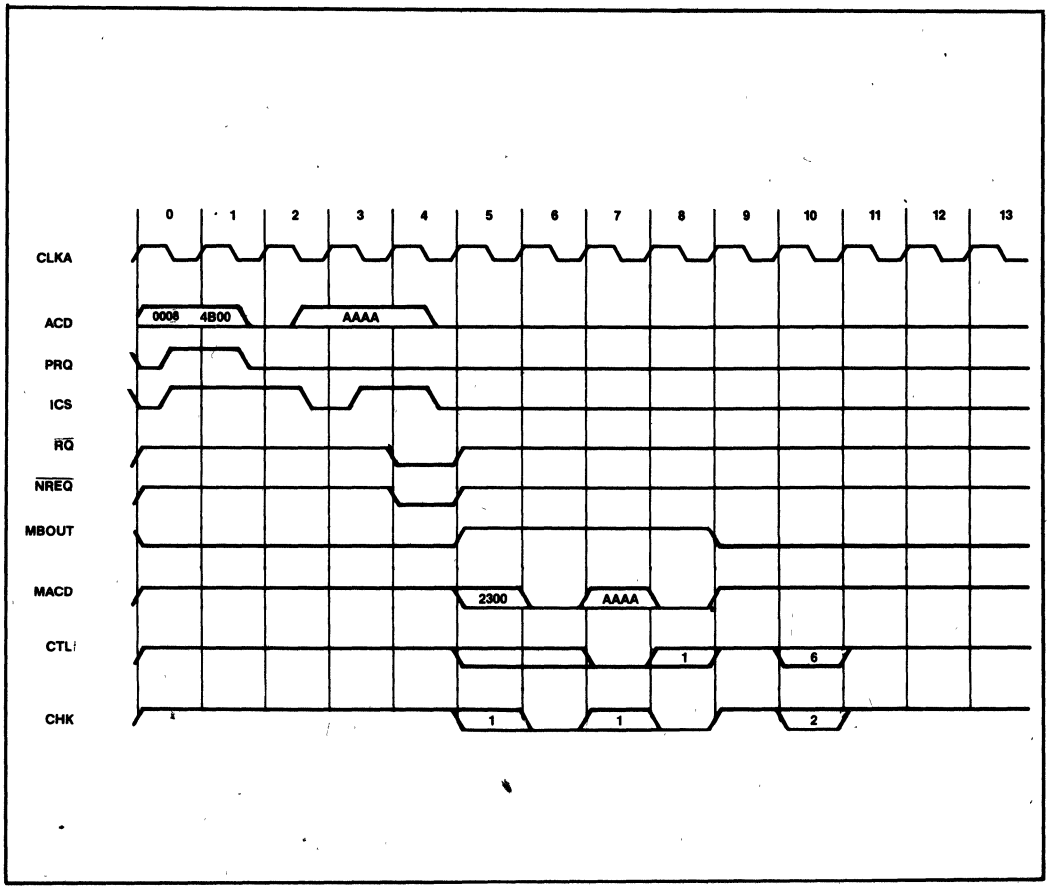


Figure 15. 4-Byte Memory Write

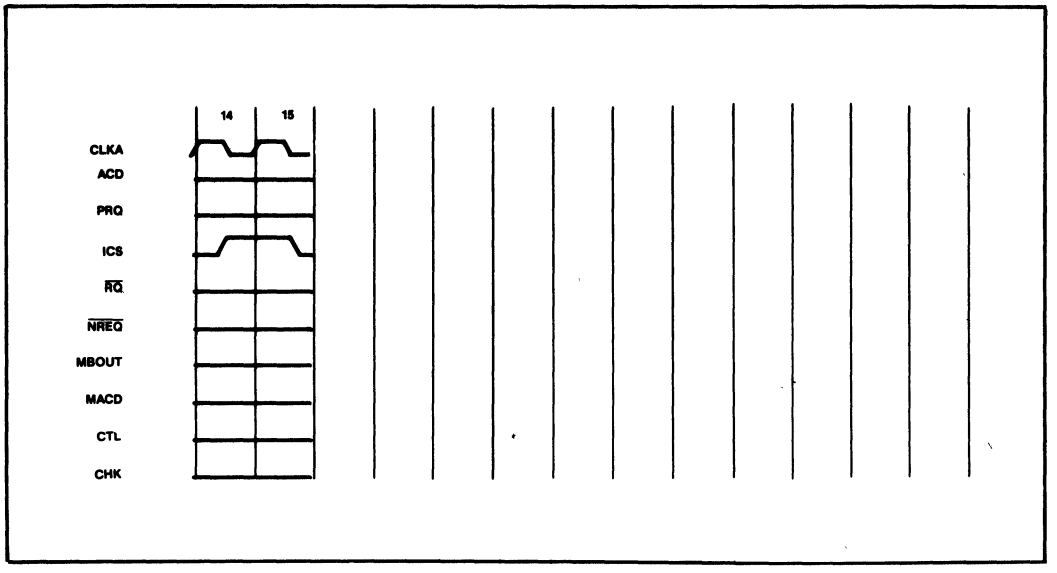


Figure 15. 4-Byte Memory Write (continued)

MULTIPLE MODULE ACCESS READ— 2-WAY INTERLEAVING

The Figures 16 and 17 illustrate the interconnect system's ability to support interleaving of memory requests among two memory busses. The hardware configuration pictured in Figure 16 contains one processor, two BIUs, two memory busses, two MCUs, and two storage arrays. The two BIUs are initialized to perform 2-way interleaving of memory requests based on bit 6 of the physical address that the processor provides. Specifically, that portion of a request with bit 6 of the physical address equal to zero is serviced by memory bus 0. That portion of a request with bit 6 of the physical address equal to one is serviced by bus 1.

In the example that follows, an 8-byte read at physical memory address 00003DH is requested. As Figure 16 indicates, the two BIUs each recognize the portion of the request that is supported by the bus to which they are connected. The BIU on bus 0 (BIU0) services the first 3 bytes of the access (03DH . . . 03FH) and the BIU on bus 1 (BIU1) services the remaining 5 bytes (040H . . . 044H). In the interleaving process, the BIUs reorder the physical address bits, based on the selected interleaving, to present requests to the MCU in a linear address space. A BIU transforms its portion of the request by replacing address bit 23 of the physical address with address bit 6, and shifting original address bits 23 . . . 7 to the right by one position. The reordered result (6,23 . . . 7,5 . . . 0) is forwarded to the respective memory bus. The BIU1 translation of physical address 000040H to memory bus 2 address 800000H illustrates the address interleaving operation.

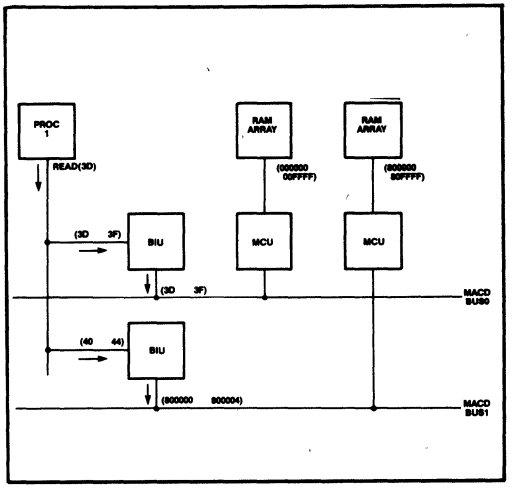


Figure 16. Hardware Configuration for MMA Read

Referring to Figure 17 for this operation, notice that the processor emits its request just as in earlier examples. However, in this case, two BIUs recognize that they each must participate in the request. Thus, BIU0 asserts MMAL and BIU1 asserts MMAH indicating that a multiple module access is required to complete this access. The two BIUs then present their requests to their respective memory busses after performing the address interleaving as described above and each access continues independently. Once both accesses have completed, the BIUs cooperate to return the data to the processor. BIU0 provides the first 3 bytes and BIU1 returns the final 5 bytes. Notice, in cycle 23, that BIU0 provides the low byte (25H) and BIU1 provides the high-byte (F5H). Thereafter, BIU1 provides its remaining bytes with the byte significance that is required by the processor. By this cooperation, the processor is unaware that two BIUs were involved in the operation.

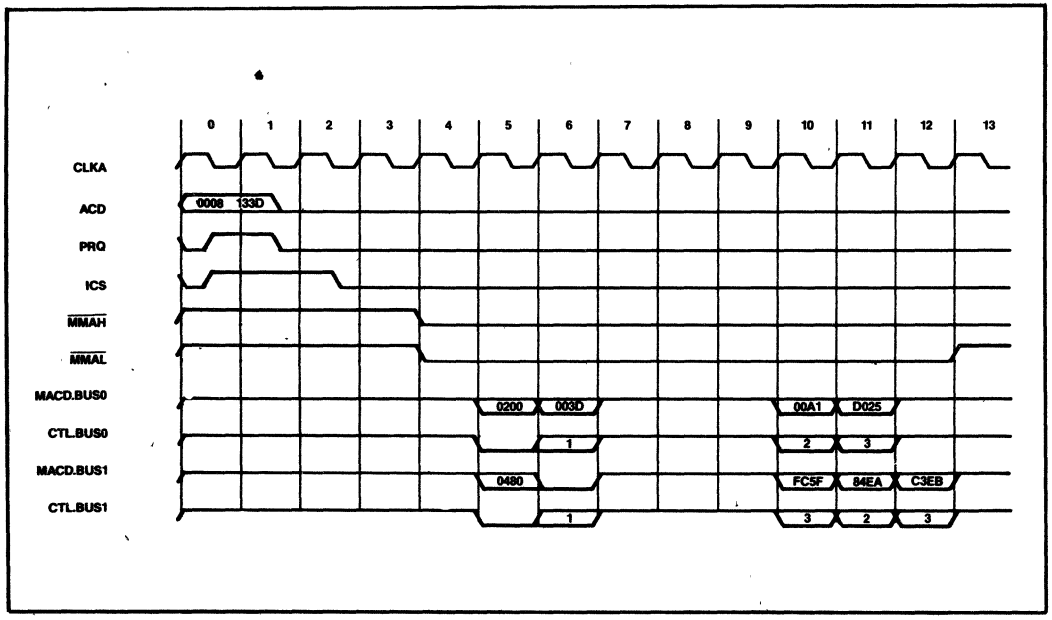


Figure 17. Multiple Module Access Read—2-Way Interleaving

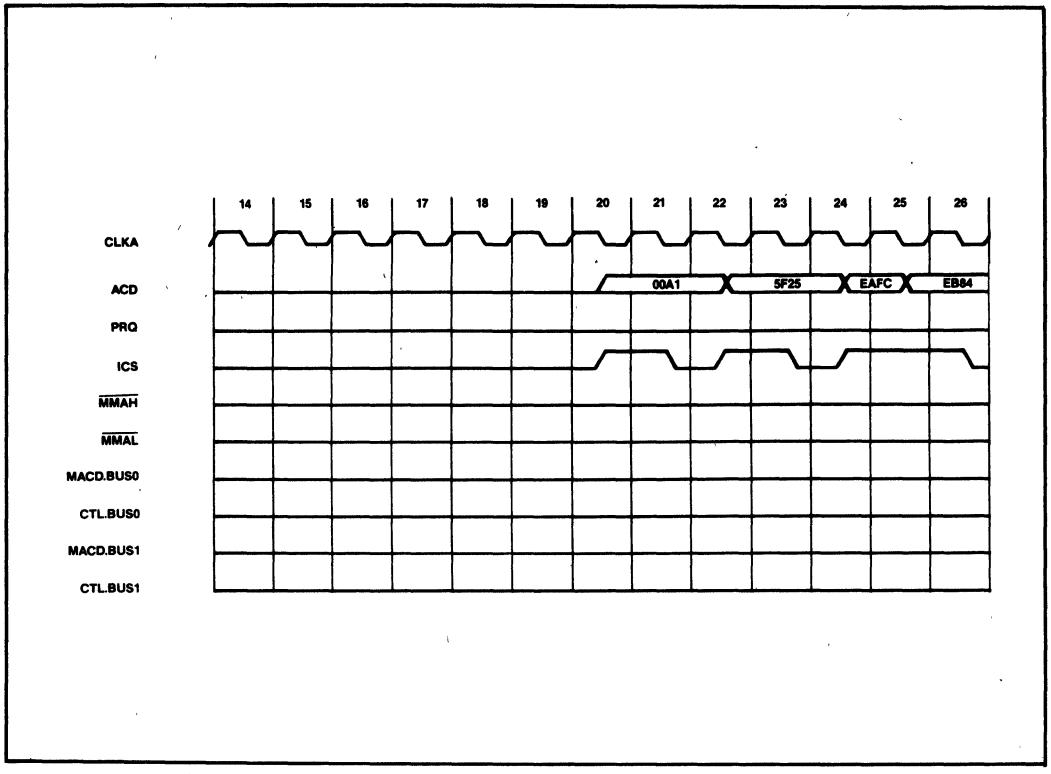


Figure 17. Multiple Module Access Read—2-Way Interleaving (cont)

MULTIPLE MODULE ACCESS WRITE—2-WAY INTERLEAVING

The following multiple module access is also performed on the same hardware configuration shown in Figure 16 with modified interleaving characteristics. In this case, the system is initialized with 2-way interleaving on address bit 7 (see Figure 19). Therefore, the 8-byte multiple module access write to physical address 00007DH is transformed into two memory bus requests: bytes 000040H . . . 000044H on bus 0 and bytes 80003DH . . . 80003FH on bus 1. Again, the $\overline{MMAH}$ and $\overline{MMAL}$ signals coordinate the multiple module access and each bus operates independently (see Figure 20).

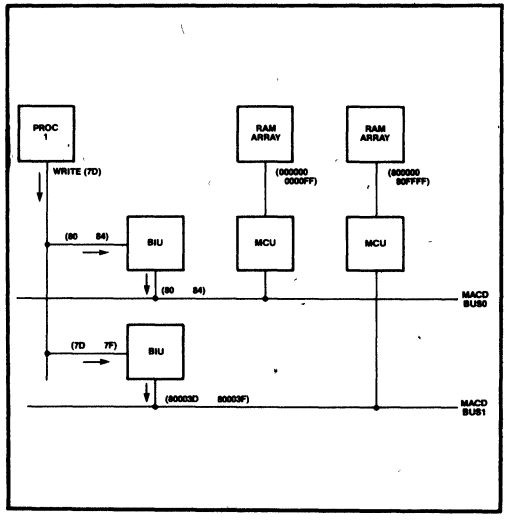


Figure 18. Hardware Configuration for MMA Write

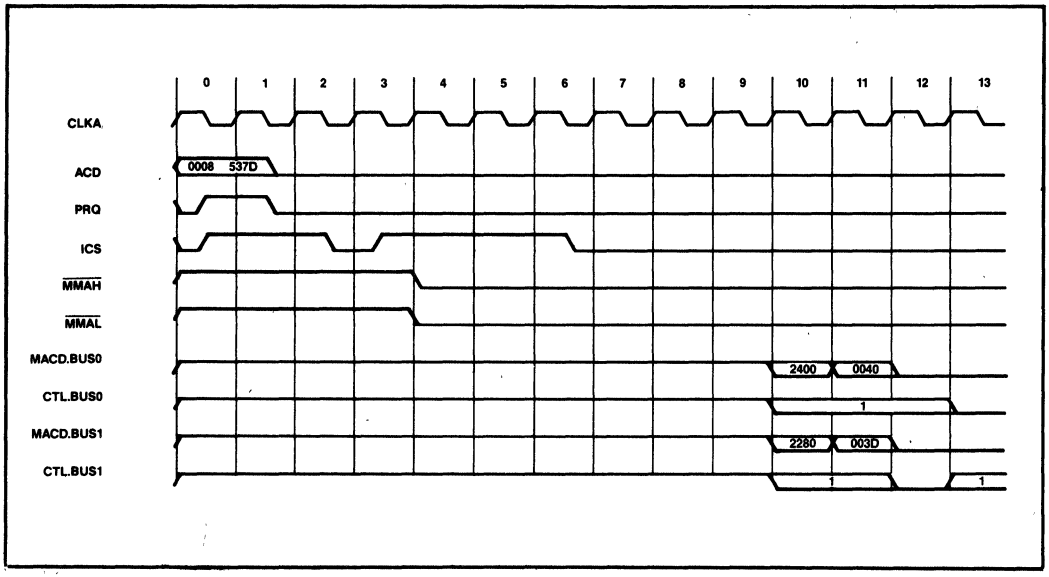


Figure 19. Multiple Module Access Write—2-Way Interleaving

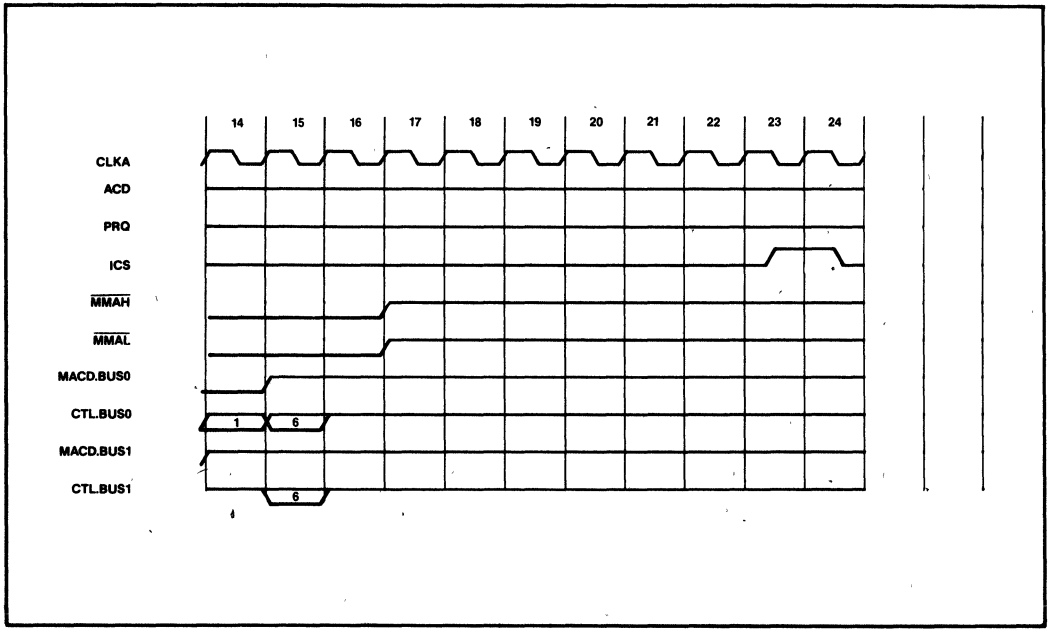


Figure 19. Multiple Module Access Write—2-Way Interleaving (Continued)

GLOBAL INTERPROCESSOR COMMUNICATION (IPC) BUS BLURB

Figure 20 illustrates the signaling of a global interprocessor communication message (IPC). The IPC is called a bus blurb since it is broadcast to all BIUs on the memory bus but does not require that any replies be generated. The delivery of the message is guaranteed by a memory-based communication object; the bus blurb only serves as the low-level notification. A programmer invokes a global IPC by the instruction BROADCAST TO PROCESSORS. The processor performs the instruction by writing a value (00H) to the IPC interconnect register (address 02H) in the associated BIU. In cycle 0, the processor emits a specification field (C7H). In cycles 0 and 1, the processor provides the interconnect address (000002H). In cycle 2, the processor provides the destination processor ID (0000H represents all processors, the global ID). Notice that the BIU stretches the processor (ICS=0) until the delivery of the global IPC is completed (cycle 10).

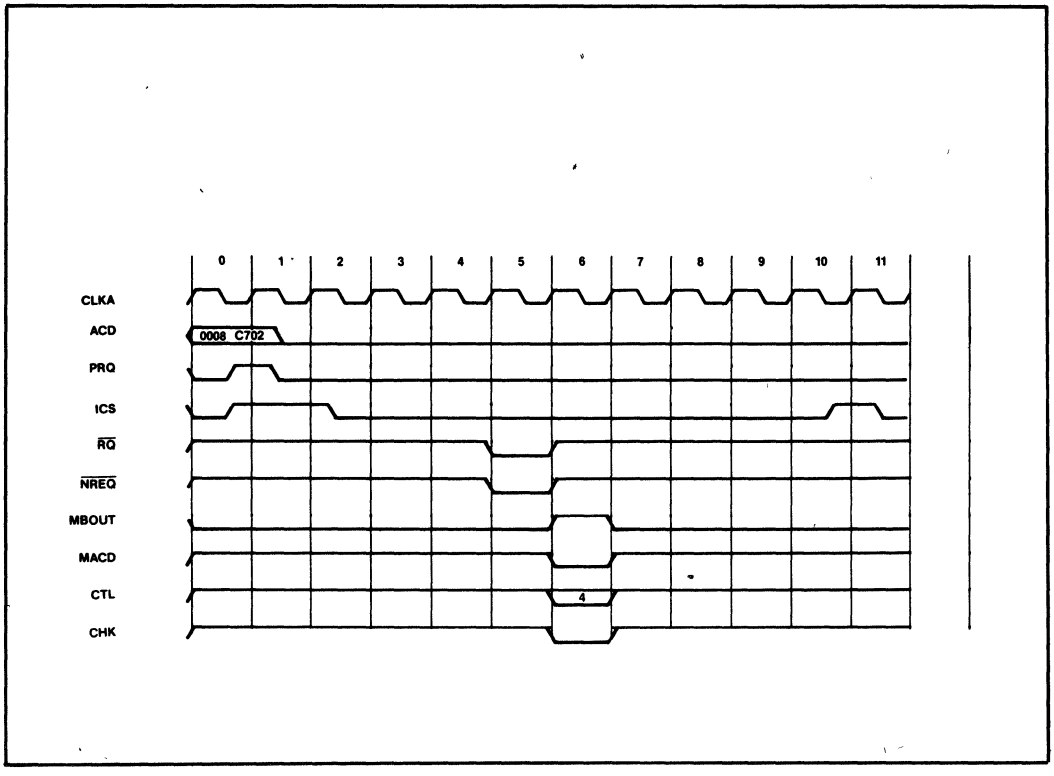


Figure 20. Global Interprocessor Communication (IPC) Bus Blurb

MACD CORRUPTION AND PARITY ERROR

Figure 22 illustrates the process of detecting and reporting a memory bus parity error to the serial error reporting network. This reporting method is common to all errors that are detected by the interconnect system. The error reporting paths for this example are illustrated in Figure 21.

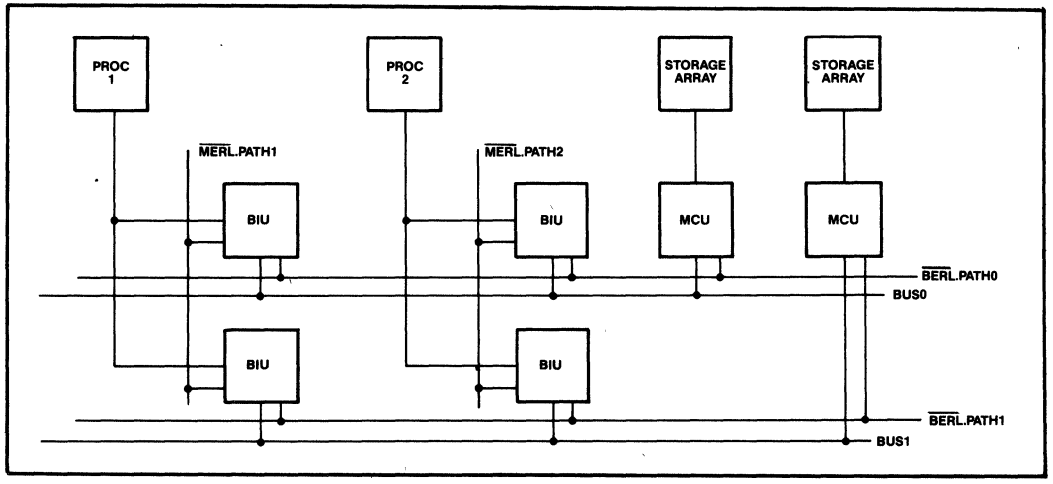


Figure 21. Hardware Configuration for Error Reporting

PHASE 1—DETECTING AND REPORTING THE PARITY ERROR ON BERL

In cycle 3, bit 1 of the memory bus (MACD signals) is corrupted with a zero causing the MACD data to change from FFFFH to FFFDH. This causes a parity error on the memory bus that must be reported to the interconnect system. The reporting process begins in cycles 5 and 6 when the BERL1 signal (driven by BERLOUT) carries the start code of the serial error report. In cycles 7 through 21, the message is propagated along the bus error report path associated with bus 0. The message sent is (complement of the levels on the BERL1 signals):

```
                2
7               1                    Cycle numbers
010000000000101                      Binary serial error message on BERL1.PATH0
 s  eeee  bbb  mmmmmm  p
    3210  210   543210
 0  1000  000   000010  1             Reformatted serial error report message
```

Parity bit
Module number
Bus number
Error code
Sequence

The error report codes are described in the iAPX 432 Interconnect Architecture Reference Manual, order no. 172487 (see BIU Register 00—Error Report Log).

PHASE 2—BROADCASTING THE ERROR ON MERL

Once the full serial message has propagated along the memory bus, each BIU that received it repeats the message along the module error report line path (MERL1 and MERL2). This time, the error report message is:

```
2               3
5               9                    Cycle numbers
110000000000100                      Reformatted serial error report message
 s  eeee  bbb  mmmmmm  p
    3210  210   543210
 0  1000  000   000010  0             Reformatted serial error report message
```

Parity bit
Module number
Bus number
Error code
Sequence

PHASE 3—REBROADCASTING THE ERROR ON ALL BERL PATHS

In cycle 42, the message is again rebroadcast, this time along all the bus error report line paths. The message sent is:

```
4          5
4          8
110000000000100
s  eeee   bbb  mmmmmm  p
   3210   210   543210
1  1000   000   000010  0
```

Cycle numbers

Binary serial error message on BERL1.PATH0

Reformatted serial error report message

Parity bit
Module number
Bus number
Error code
Sequence

Once this three-step reporting process is completed, all modules have been informed of the error.

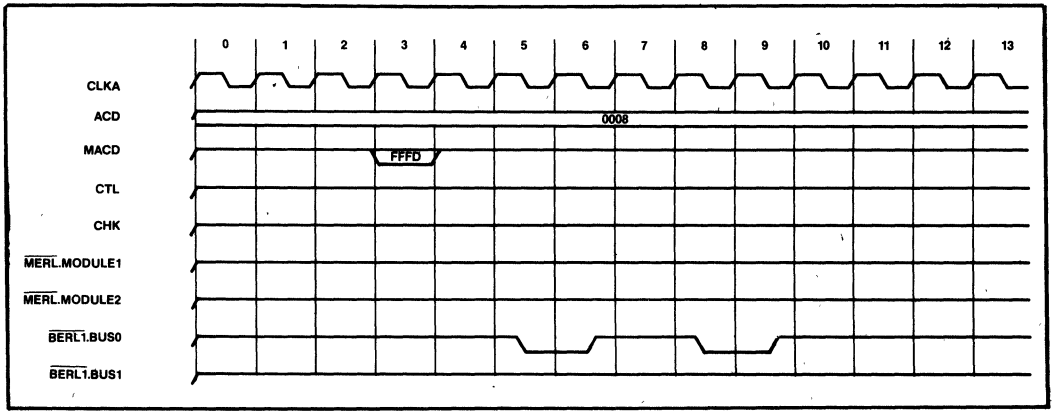


Figure 22. MACD Corruption Causing Parity Error

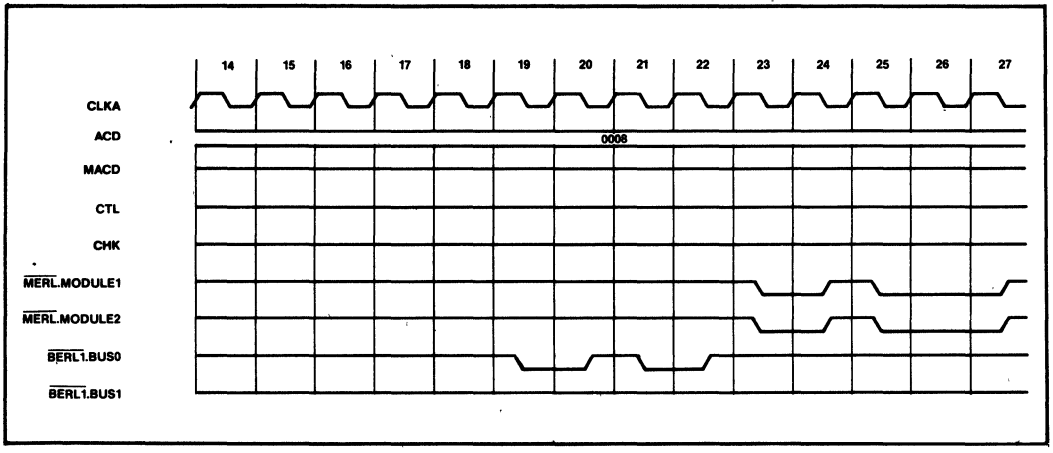


Figure 22. MACD Corruption Causing Parity Error (continued)

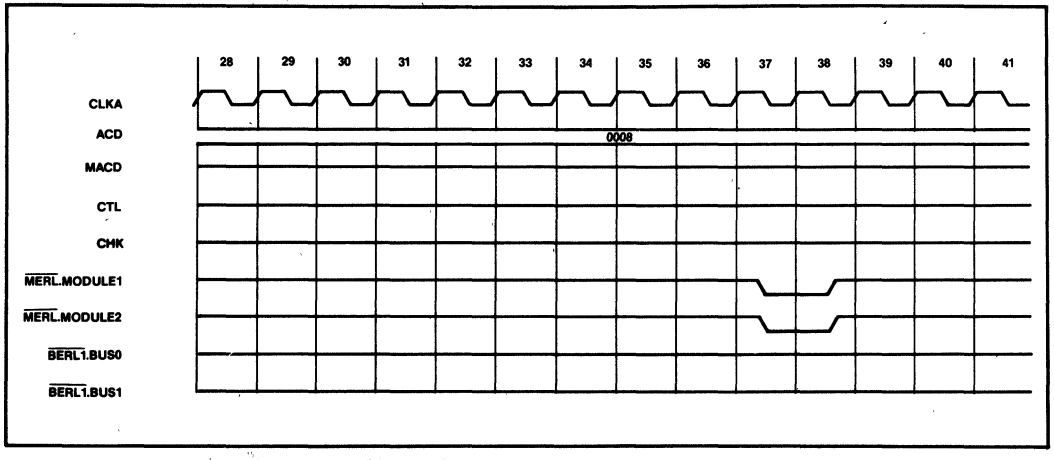


Figure 22. MACD Corruption Causing Parity Error (continued)

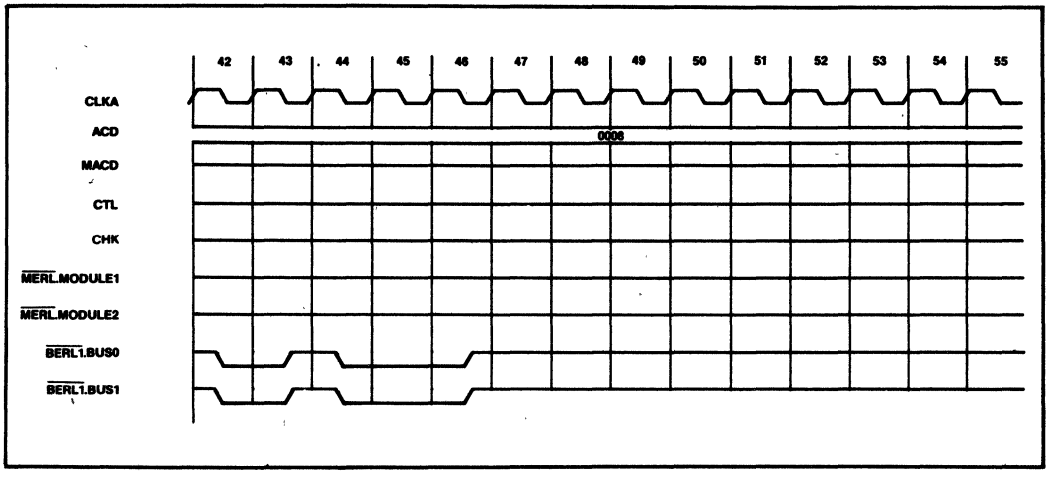


Figure 22. MACD Corruption Causing Parity Error (continued)

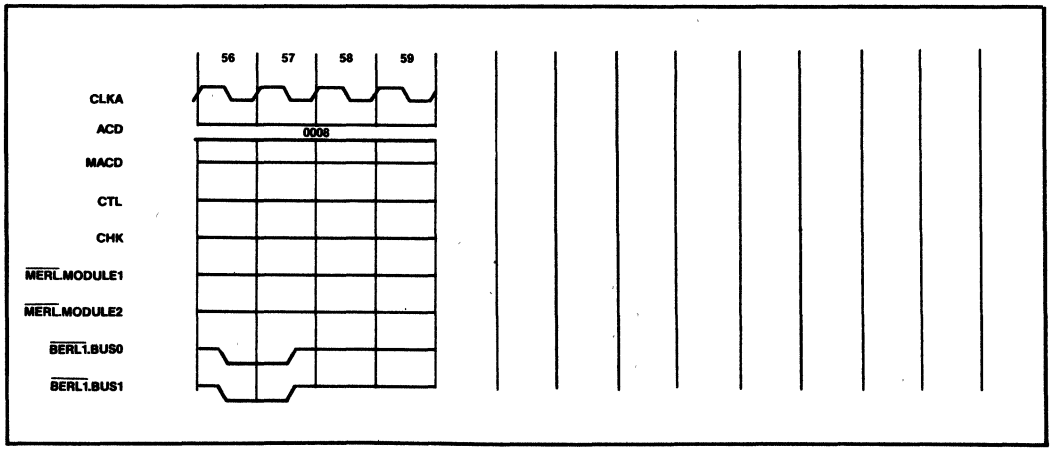


Figure 22. MACD Corruption Causing Parity Error (continued)

TEST DETECTION COMMAND

The Test Detection command is a special command that requests a BIU to test the detection hardware contained inside the component. It checks all the FRC circuitry, memory bus parity generator/checkers, and buffer checking logic. The hardware configuration for this example is the same as that for the previous example (see Figure 21). When the command has been performed an error report message is generated, just as in the previous example. When the test is successful, the message indicates "no error." When any of the detection circuitry has failed, the message indicates "module error." In this example, the detection circuitry is operating properly and phase 1 of the serial error reporting sequence contains the "no error" message starting in cycle 5 (see Figure 23).

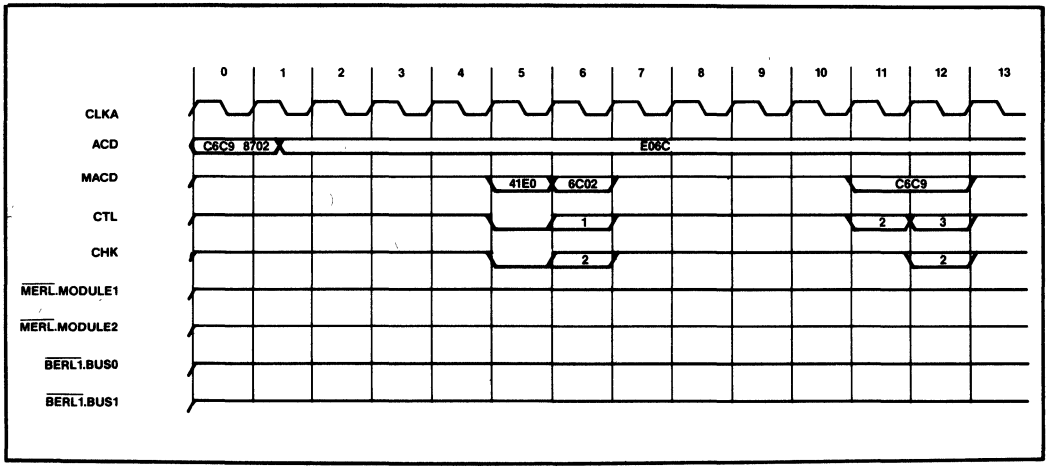


Figure 23. Test Detection Command

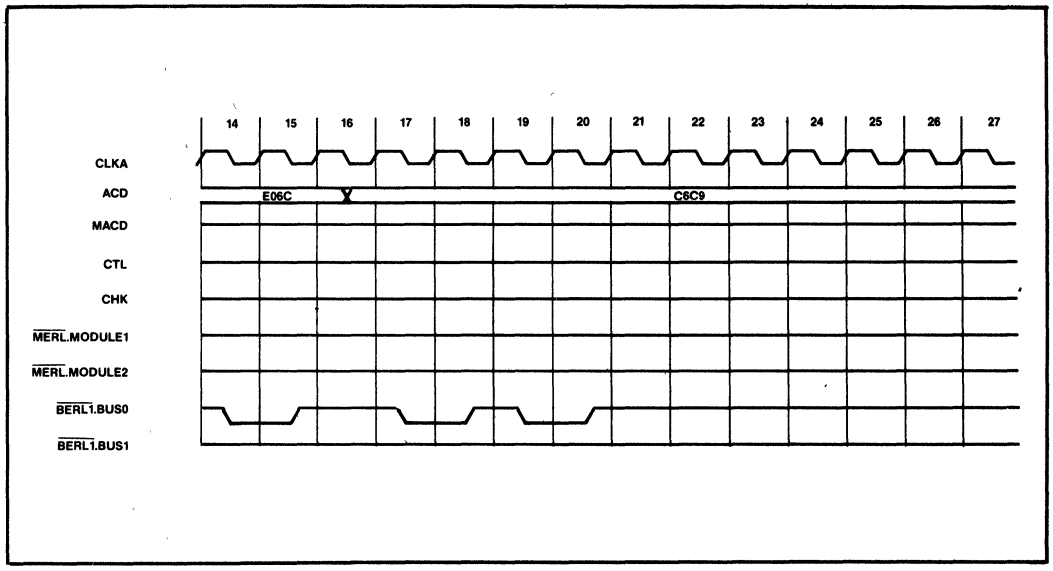


Figure 23. Test Detection Command (continued)

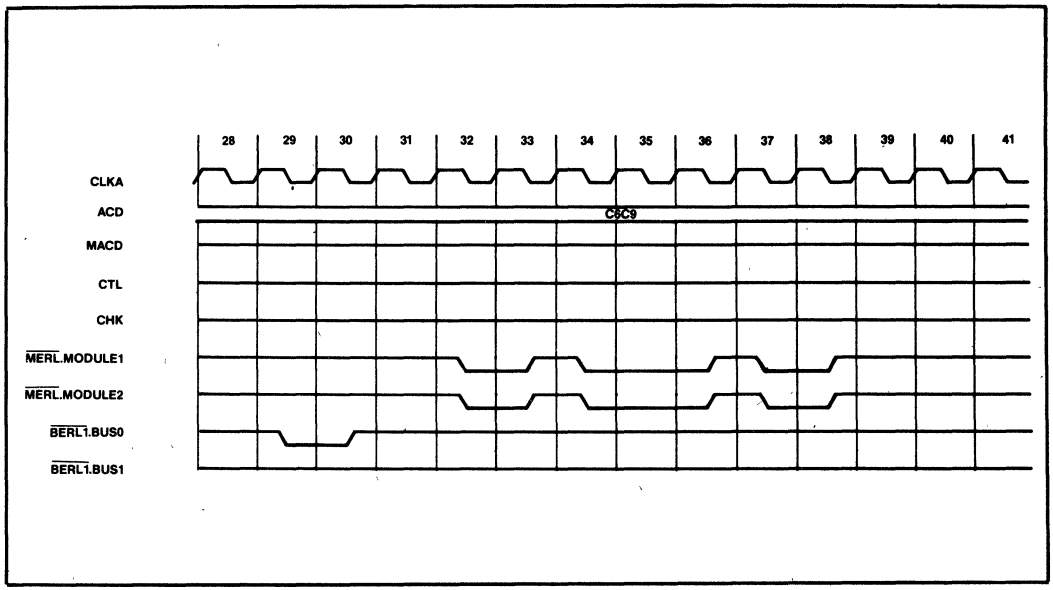


Figure 23. Test Detection Command (continued)

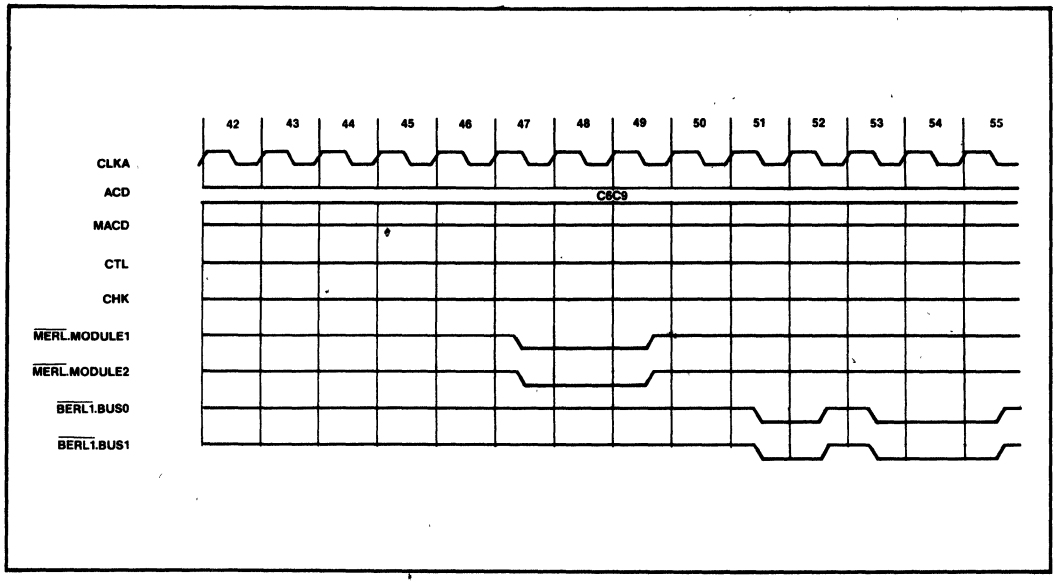


Figure 23. Test Detection Command (continued)

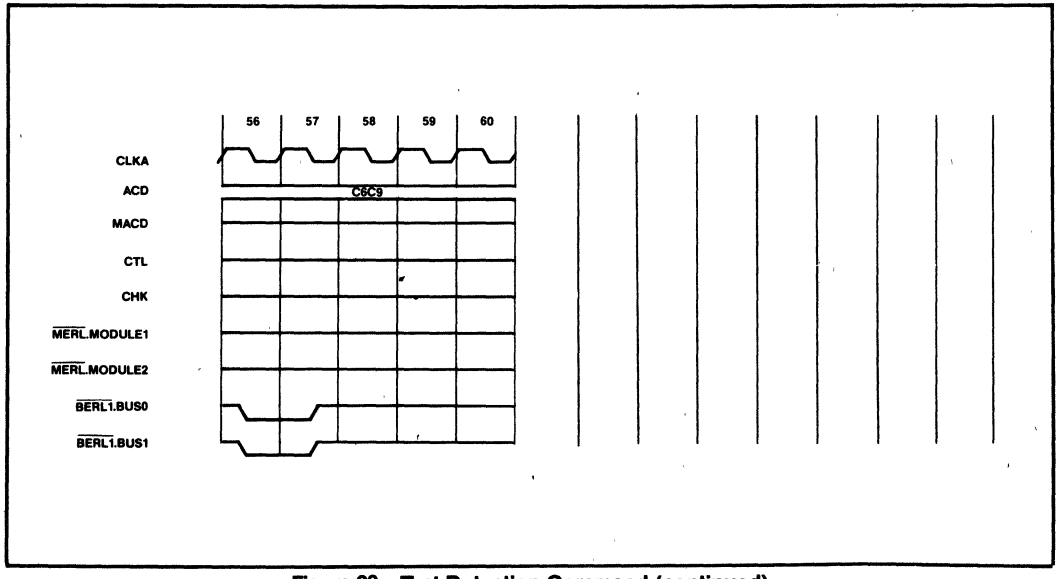


Figure 23. Test Detection Command (continued)

MACD BUS ARBITRATION

Figure 24 illustrates two processors as they simultaneously issue memory requests. Since the requests are both serviced by the same memory bus and MCU, arbitration is performed to select the first request that should be presented. The hardware configuration consists of two processors (processor modules 2 and 6), two BIUs, one memory bus, one MCU, and one storage array (see Figure 25).

In cycle 0, both processors issue a request indicated by the respective processor request signals, PRQ.PROC2 and PRQ.PROC6. In cycle 4, the associated BIUs present their requests to the memory bus (NREQ=0, RQ=0) and each notes that another BIU is also contending (CONT=0) for the use of the memory bus. The BIUs resolve the contention by examining the individual bits of their respective logical IDs, beginning with the most significant bit. In this case, the default logical ID is a bit-reversed version of the module ID. In cycle 4, each BIU notes contention. In cycle 5, each BIU still notes contention. In cycle 6, the BIU serving module 2 wins the arbitration and issues its request to the memory bus. Overlapped with the first request, the BIU in module 6 reissues its request for the memory bus (RQ=0 in cycle 9) and detects no contention. However, each BIU monitors the memory bus and the BIU in module 6 waits for the current bus activity to complete before its request can be placed on the bus.

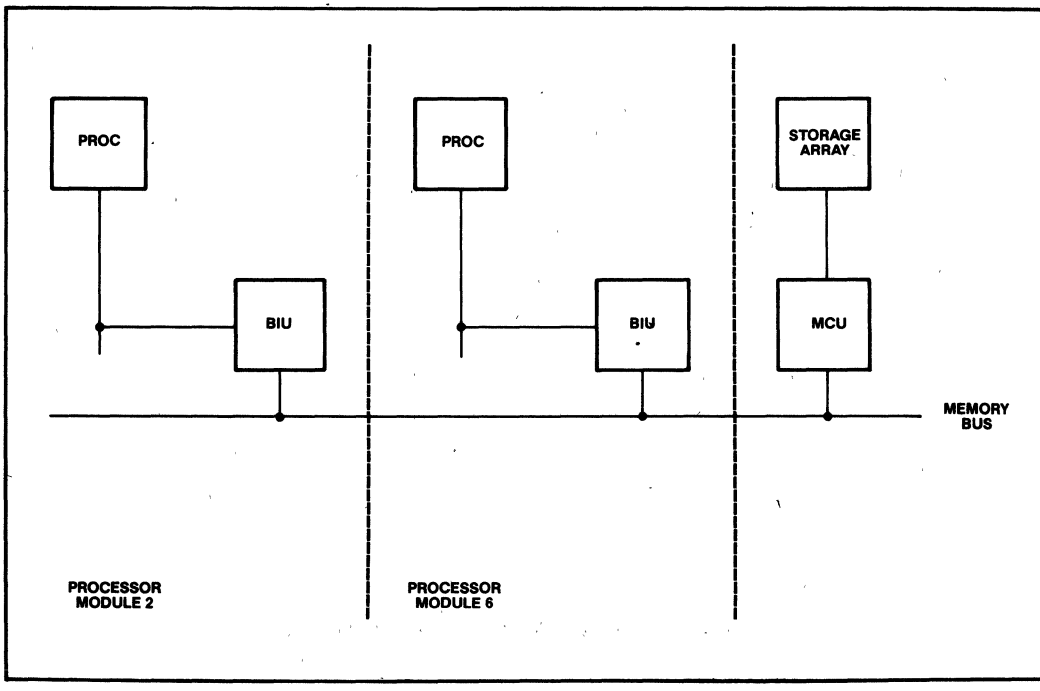


Figure 25. Hardware Configuration for Memory Bus Arbitration

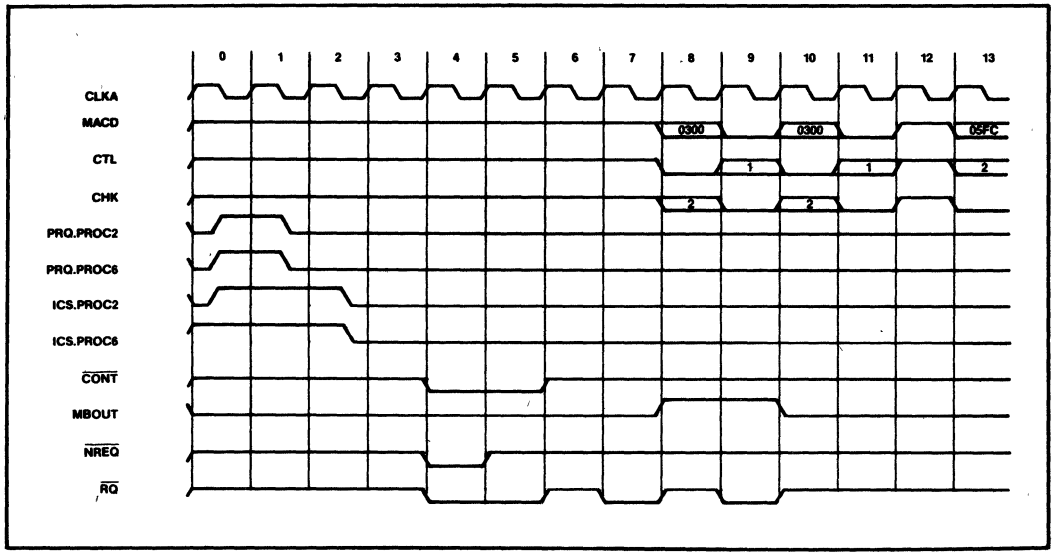


Figure 24. MACD Bus Arbitration

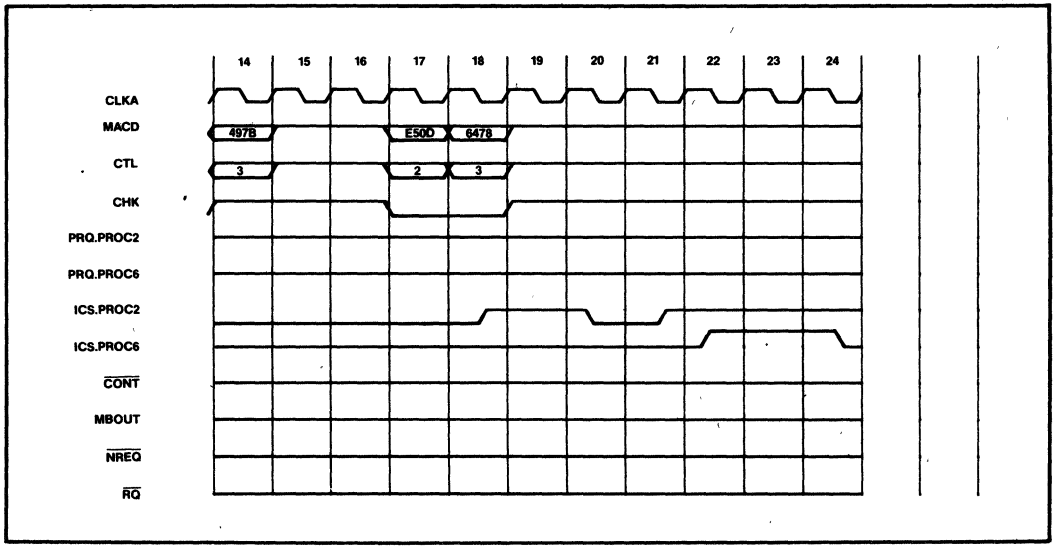


Figure 24. MACD Bus Arbitration (continued)

Table 3. iAPX 43204 Clock Edge Table

Signal	Pin(s)	Input Sample		Output Drive	
		Clock	Edge	Clock	Edge
Inputs					
MERL	44	CLKA	Rising	n/a	
BERL1	38	CLKA	Rising	n/a	
BERL2	39	CLKA	Rising	n/a	
ICS	48	CLKA	Rising	n/a	
INIT	37	CLKA	Rising	n/a	
RQ	9	CLKA	Falling	n/a	
CONT	10	CLKA	Falling	n/a	
NREQ	5	CLKA	Falling	n/a	
PRQ	49	CLKB	Rising	n/a	
Outputs					
MERLOUT	46	n/a		CLKA	Rising
BERLOUT	40	n/a		CLKA	Rising
ICSOUT	47	n/a		CLKB	Rising
CLRPUOUT	45	n/a		CLKB	Rising
Input/Output					
MMAH	50	CLKA	Rising	CLKA	Rising
MMAL	51	CLKA	Rising	CLKA	Rising
ACD15 . . . 0	1, 54-68	CLKB	Rising	CLKB	Falling
MACD15 . . . 0	13-28	CLKA	Rising	CLKB	Rising
CTL2 . . . 0	29-31	CLKA	Rising	CLKB	Rising
CHK1 . . . 0	11, 12	CLKA	Rising	CLKB	Rising
MBOUT	32	CLKA	Rising	CLKA	Rising
NREQOUT	4	CLKA	Falling	CLKA	Falling
RQOUT	6	CLKA	Falling	CLKA	Falling
BCHK	7	CLKA	Rising	CLKB	Rising

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Range ..... 0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 2.2 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

iAPX 43204 DC Characteristics

Symbol	Description	Min	Max	Units
Vilc	Clock input low voltage	−0.5	+0.8	V.
Vihc	Clock input high voltage	3.2	Vcc+0.5	V.
Vil	Input low voltage	−0.5	+0.8	V.
Vih	Input high voltage	2.0	VCC+0.5	V.
Vol	Output low voltage	—	0.45	V.
Voh	Output high voltage	2.4	VCC	V.
Ili	Input leakage current (measured at Vin=VCC)	—	±10	μA.
Ilo	Output leakage current (measured at 0.45 V. ≤ Vout ≤ VCC)	—	±10	μA.
Ioh	Output high current (measured at 2.6 V.)	−2	—	mA.
Iol	Output low current (measured at 0.45 V.)	4	—	mA.
Icc	Power supply current (sum of VCC0, VCC1, VCC2)	—	400	mA.

All DC parameters are guaranteed over the following conditions:

VSS2..VSS0 = 0 Volts
VCC2..VCC0 = 5.0 Volts ± 5%
The absolute value of the differential DC voltage between any of the VCC pins (VCC2..VCC0) must be less than 0.1 Volts. This is normally guaranteed by connecting the three VCC pins to the same printed circuit power trace.

iAPX 43204 AC Characteristics

Ambient temperatrue range of 0°C to 70°C

Symbol	Description	5 MHz		7 MHz		8 MHz		Unit
		Min	Max	Min	Max	Min	Max	
t_r, t_f	Clock rise and fall times	-	13	-	11	0	10	nsec
t_1, t_2, t_3, t_4	Clock pulse width	37	250	25	250	24	250	nsec
t_{cy}	Clock cycle time	200	1000	143	1000	125	1000	nsec
t_{cd}	Clock to signal delay time	-	70	-	60	-	55	nsec
t_{dh}	Clock to signal hold time	20	-	17	-	15	-	nsec
t_{en}	Clock to signal output enable time	20	-	17	-	15	-	nsec
t_{df}	Clock to signal data float time	-	50	-	44	-	40	nsec
t_{dc}	Signal to clock setup time	30	-	26	-	24	-	nsec
t_{ie}	Initialization period	20	100	20	100	20	100	t_{cy}

All AC parameters are guaranteed over the following conditions:

Ambient temperature range of 0 degrees Centigrade to 70 degrees Centigrade

VSS2 . . . VSS0 = 0 Volts
VCC2 . . . VCC0 = 5.0 Volts ± 10%

100 picoFarad external load capacitor on all output pins

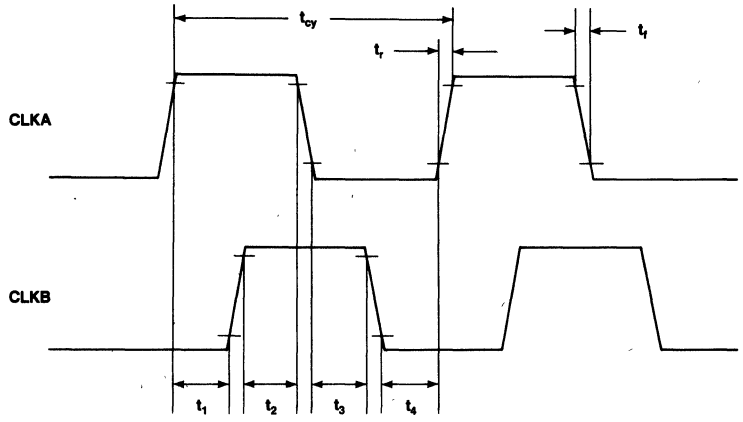
iAPX 43204 Capacitance Data

Conditions: Ta = 25°C
VCC = 5.0 Volts, GND = 0.0 Volts
f(test) = 1.0 MHz
Inputs held at 0.0 Volts
All outputs in high impedance state
All input/output pins are classified as outputs

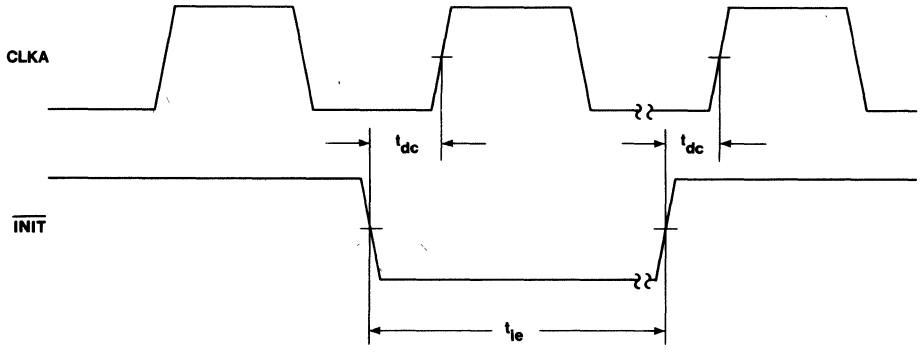
Symbol	Description	Max	Units
Cin	Input Capacitance	6	pF
Cout	Output Capacitance	12	pF

WAVEFORMS

iAPX 43204 Clock Input Specification

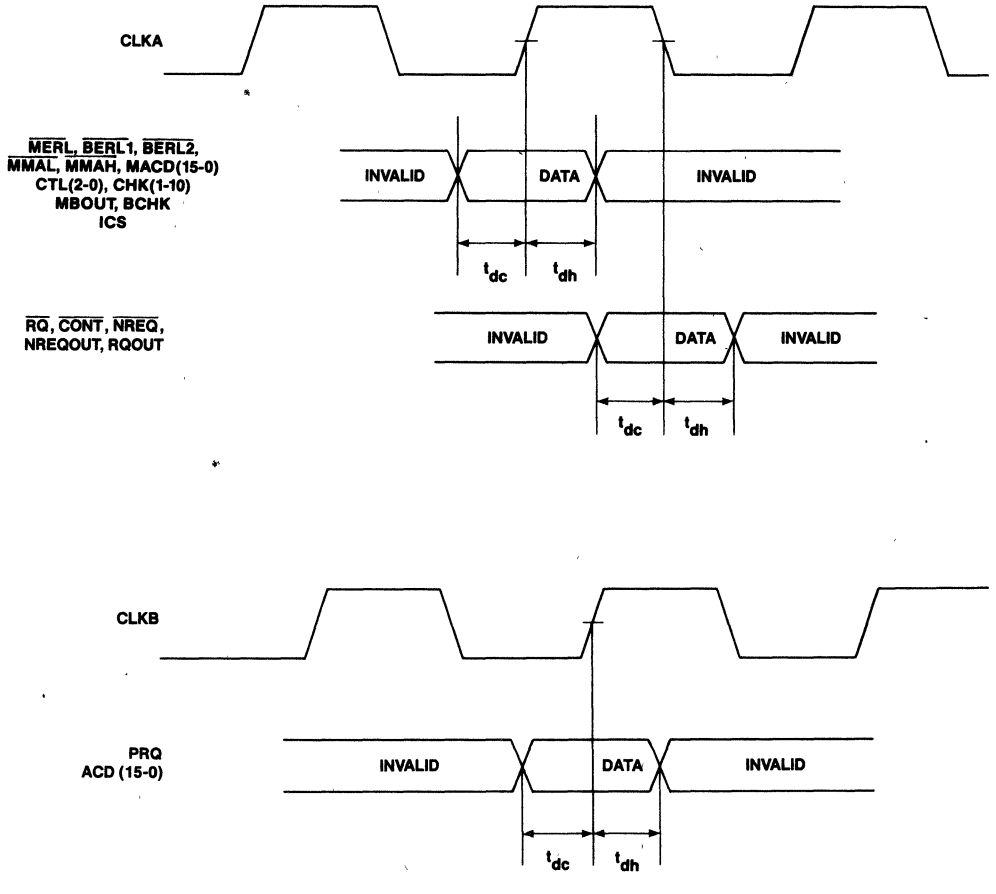


iAPX 43204 Initialization Timing



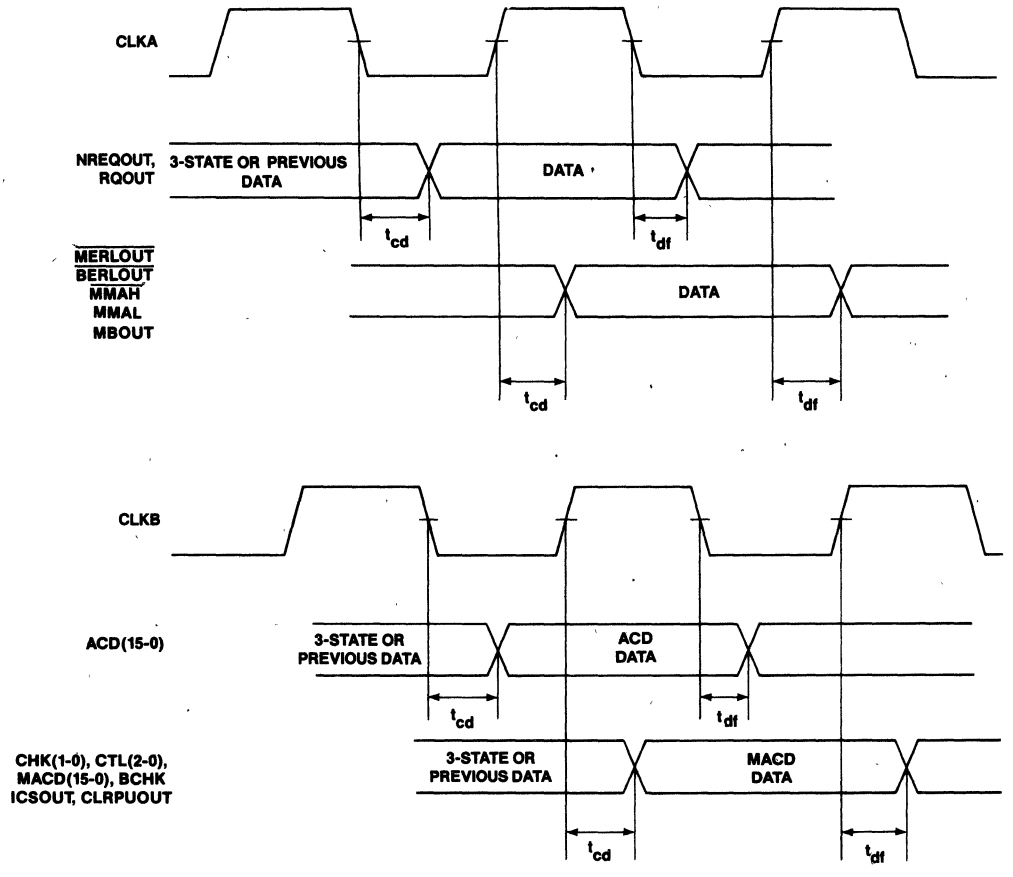
WAVEFORMS (Continued)

iAPX 43204 Input Timing Specification



WAVEFORMS (Continued)

iAPX 43204 Output Timing Specification



iAPX 43205 FUNCTIONAL DESCRIPTION

This section describes how the iAPX 43205 Memory Control Unit (MCU) operates. It contains a set of diagrams that present the clock-by-clock activity of the pins on an MCU component. To understand the notation on the waveforms, refer to Figure 26. It illustrates a 32-bit memory read operation of physical memory array address 00000H.

The cycle numbers (0, 1, 2, . . .) at the top of each waveform enumerate the clock cycles. When a common group of signals is plotted, its value is displayed inside a data waveform. For example, the MACD signals (MACD15 . . . MACD0) are all high in cycle 0, carry the hexadecimal value 0300H in cycle 1, and are all low in cycle 3. The MACD signals (MACD15 . . . MACD0) carry the values 0300H in cycle 1 and 0000H in cycle 2. The first double byte of the access contains the specification field (03H—the high byte of the first double byte) and the low-order address byte (00H—the low byte of the first double byte) of the memory address. The second double byte of the access contains the high- (00H) and mid-order (00H) address bytes. This example illustrates a double byte read performed at physical memory address 00000H. Also, notice that the SLAD pins (SLAD19 . . . SLAD0) are represented with a 5-digit hexadecimal value.

Consider the SLAD group in the example. In the last half of cycle 3 and during cycle 4, the SLAD group carries the value 00000H, the storage array address for the access. In cycle 5 the array returns read data of 42200H, and in cycle 6 the array responds with DC664H. In this example, the storage array is physically 20-bits wide, and the least significant physical bit is a spare bit. Thus, to interpret the actual data present on the SLAD wires, a logical shift is necessary to align the data as it will be presented on the MACD signals.

Original SLAD Information

Cycle 5	4	2	2	0	0	Hexadecimal
	0100	0010	0010	0000	0000	Binary
					↓	Spare Bit
Cycle 6	D	C	6	6	4	Hexadecimal
	1101	1100	0110	0110	0100	Binary

Cycle 5 contains the least significant information, and cycle 6 contains the most significant information. The least significant bit of cycle 5, the spare bit, is not used in this case. Thus, the actual data and ECC information transferred on the SLAD signals is formed by concatenating the low- and high-order SLAD data groups and deleting the low-order spare bit. The 7 ECC bits occupy the high-order 7 bits of the result, and the 32 data bits occupy the low-order portion.

Original SLAD Information

D	C	6	6	4	4	2	2	0	0	
1101	1100	0110	0110	0100	0100	0010	0010	0000	0000	
									↓	
6	0	31							0	Spare bit
ECC								Data		

Spare Bit Removed

6	E	3	3	2	2	1	1	0	0
110	1110	0011	0011	0010	0010	0001	0001	0000	0000
6	0	31							0
ECC								Data	

The 32-bit value 33221100H is returned to the MACD bus with the low-order 16 bits in cycle 7 and the high-order 16 bits in cycle 8. The 7-bit ECC value is checked by the MCU and is not transferred to the MACD signals.

These waveforms are accurate cycle-to-cycle representations of MCU function. They are not meant to serve as diagrams of exact component timing. For example, the diagrams depict MACD data being issued from the MCU at the functional clock boundaries for the rising edge of CLKA. The MCU actually sources MACD information on the rising edge of CLKB. Always refer to the precise electrical specifications, AC specifications, and timing diagrams when detailed timing information is required. The functional diagrams are intended to succinctly summarize the functions of the MCU.

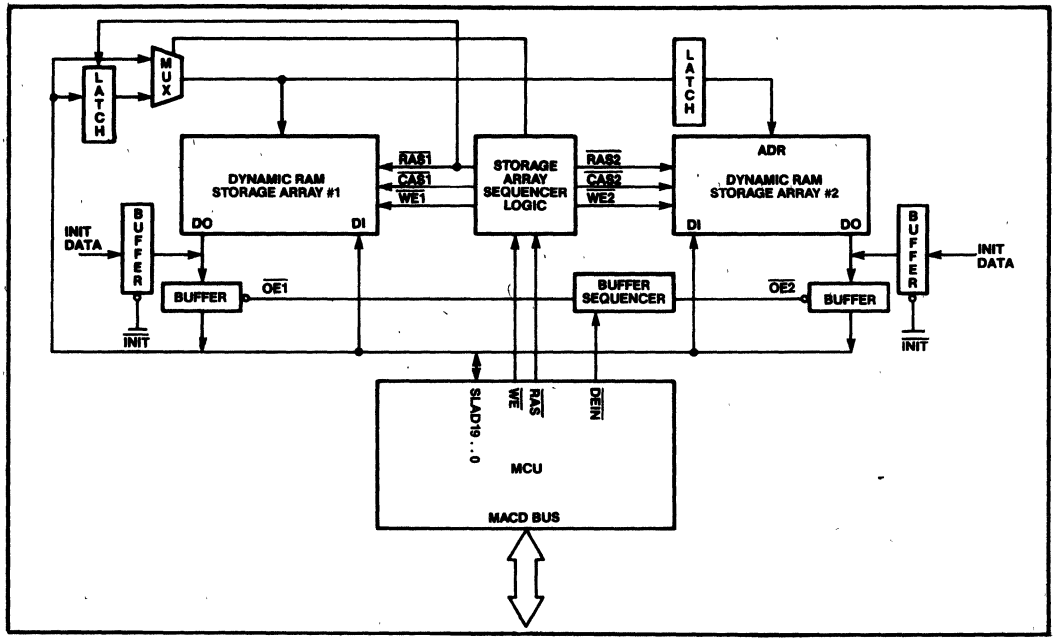


Figure 27. MCU and External Hardware Configuration

HARDWARE CONFIGURATION FOR MEMORY READ AND WRITE OPERATIONS

Figure 27 is the basis for demonstrating MCU memory read and write operations. In each memory read and memory write example, the appropriate command is presented to the MCU from the MACD (memory) bus and the MCU performs the operation to completion. Each memory operation assumes the following hardware configuration for the logic external to the MCU. Notice that external logic accepts the RAS, WE, and DEIN signals and produces the precise timing signals which are required by the associated dynamic RAM storage array. In the diagrams which follow, only signals which are present on the MCU signal pins are displayed.

MEMORY READ OPERATIONS

Figure 26 (Normal Aligned Read) illustrates a memory read operation that returns two double bytes (4 bytes) from the memory subsystem, typical for iAPX 432 GDP instruction fetch cycles.

Figure 28 (Normal 2-byte Read) illustrates a memory read operation that returns a single double byte operand.

Figure 29 (Normal Nonaligned Read) illustrates a 4-byte memory read operation that is not aligned to a 4-byte boundary, the natural boundary for instruction fetches and 32-bit operands. In this case, the MCU must perform two separate storage array accesses to acquire the 4 bytes.

Figure 30 (4-byte Read with Extended Access) illustrates a programmable option on the MCU that accommodates storage arrays that require longer memory access cycles. With the extended access option, the MCU extends RAS one cycle and delays DEIN one cycle to allow the external array sequencing logic to extend the storage array cycle.

Figure 31 (Staged Read with Correctable Error) demonstrates how an MCU can stage (hold) memory data until all ECC detection *and* correction is performed. Without selecting the staging option, a MCU would normally present the memory data it acquired to the MACD bus as in other memory read cycles. However, should an error be detected, the MCU would signal (BERLOUT) to inform a BIU that data it had been given was invalid, and the BIU would retry the access.

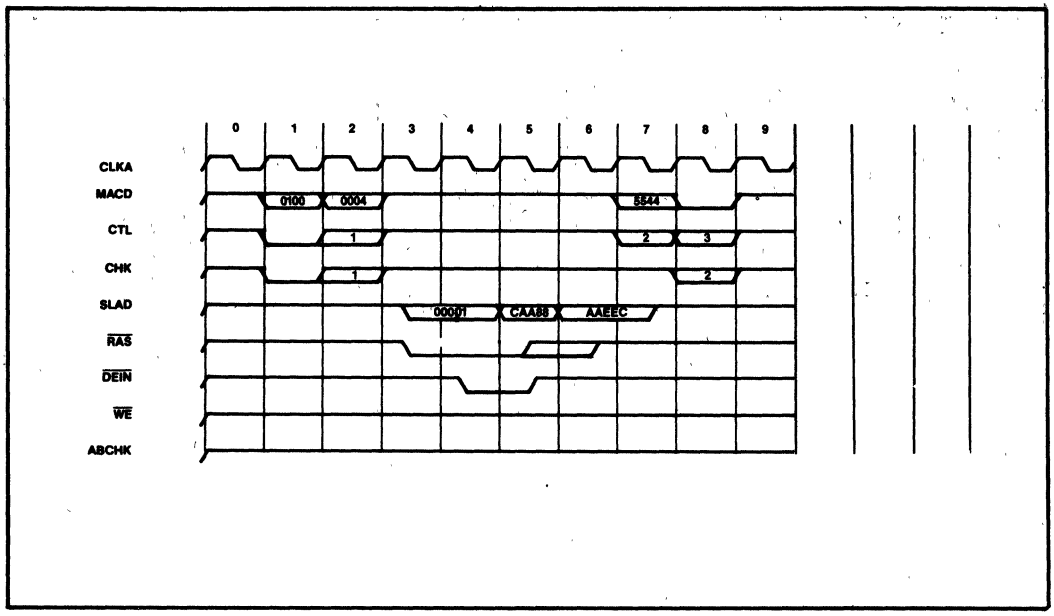


Figure 28. Normal 2-byte Read

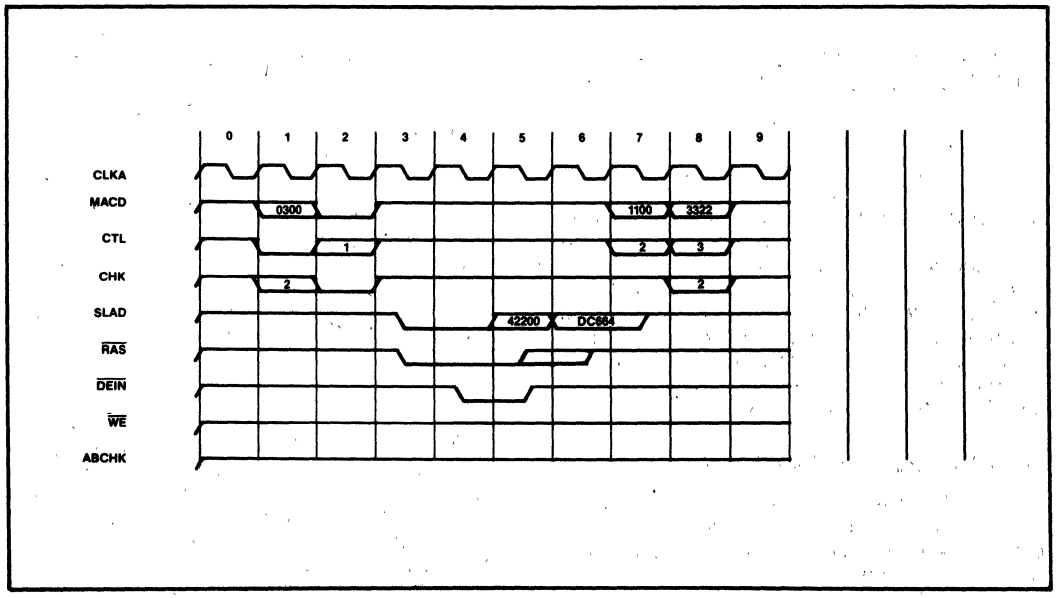


Figure 26. Normal Aligned Read

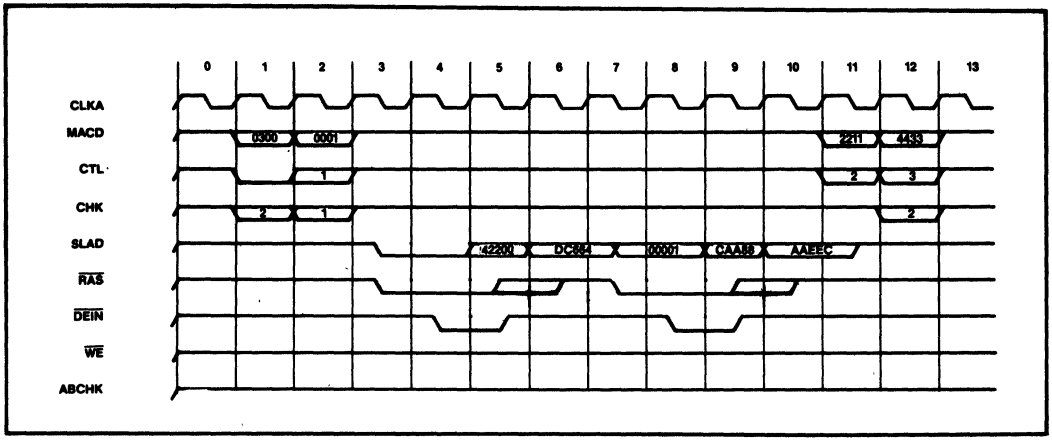


Figure 29. Normal Nonaligned Read

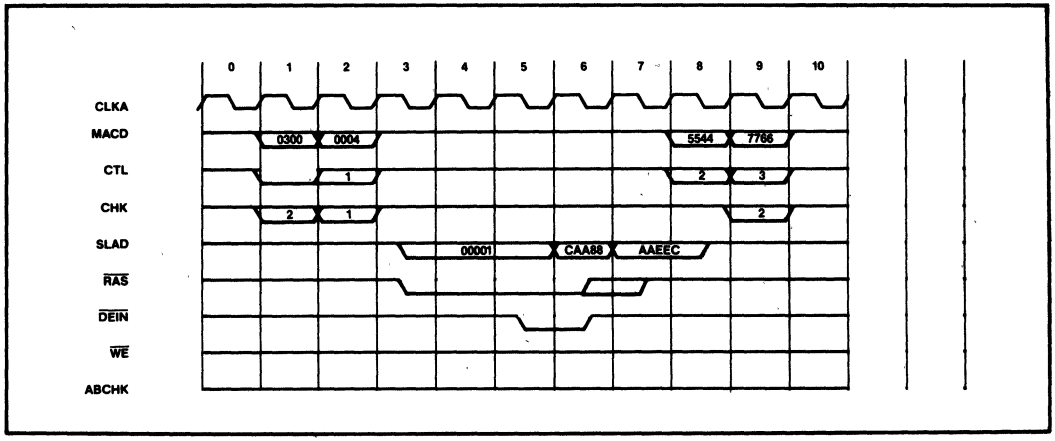


Figure 30. 4-Byte Read with Extended Access

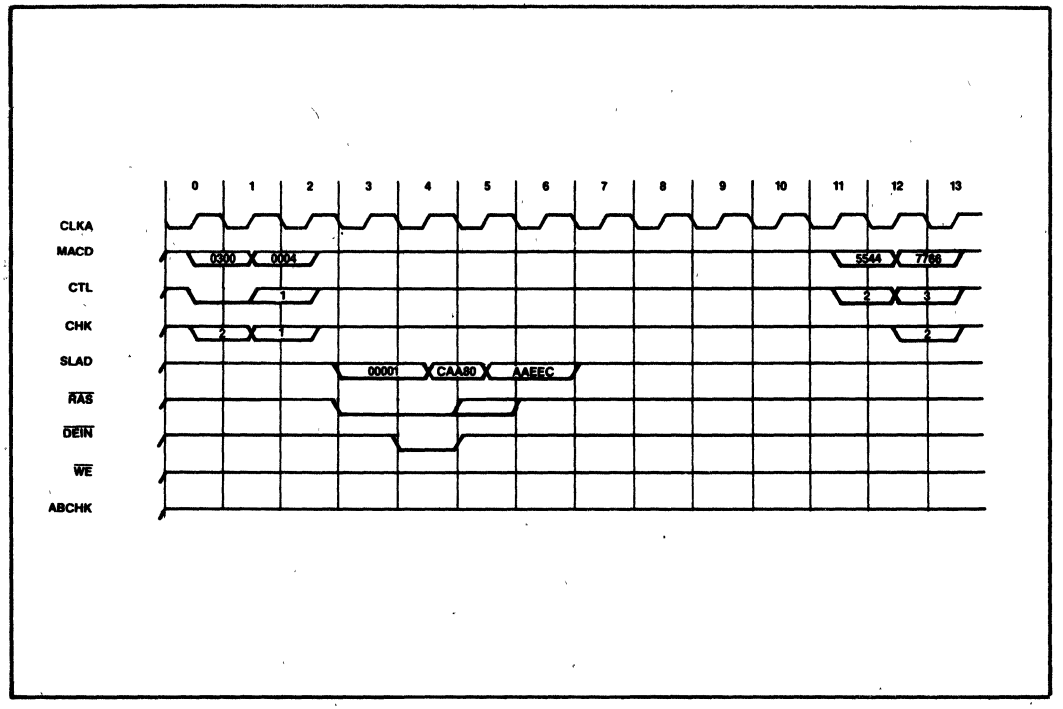


Figure 31. Staged Read with Correctable Error

MEMORY WRITE OPERATIONS

Figure 32 (Normal 2-byte Write) illustrates a memory write operation that stores a single double byte operand.

Figure 33 (Normal Aligned Write) illustrates a memory write operation that stores a 4-byte operand aligned to a 4-byte boundary.

Figure 34 (Normal Nonaligned Write) illustrates a 4-byte memory write operation that is not aligned to a 4-byte boundary. In this case, the MCU must perform two separate storage array accesses to store the 4 bytes.

Figure 35 (4-byte Write with Extended Access) illustrates a programmable option on the MCU that accommodates storage arrays which require longer memory access cycles. With the extended access option, the MCU extends RAS one clock cycle and delays DEIN one clock cycle to allow the external array sequencing logic to extend the storage array cycle.

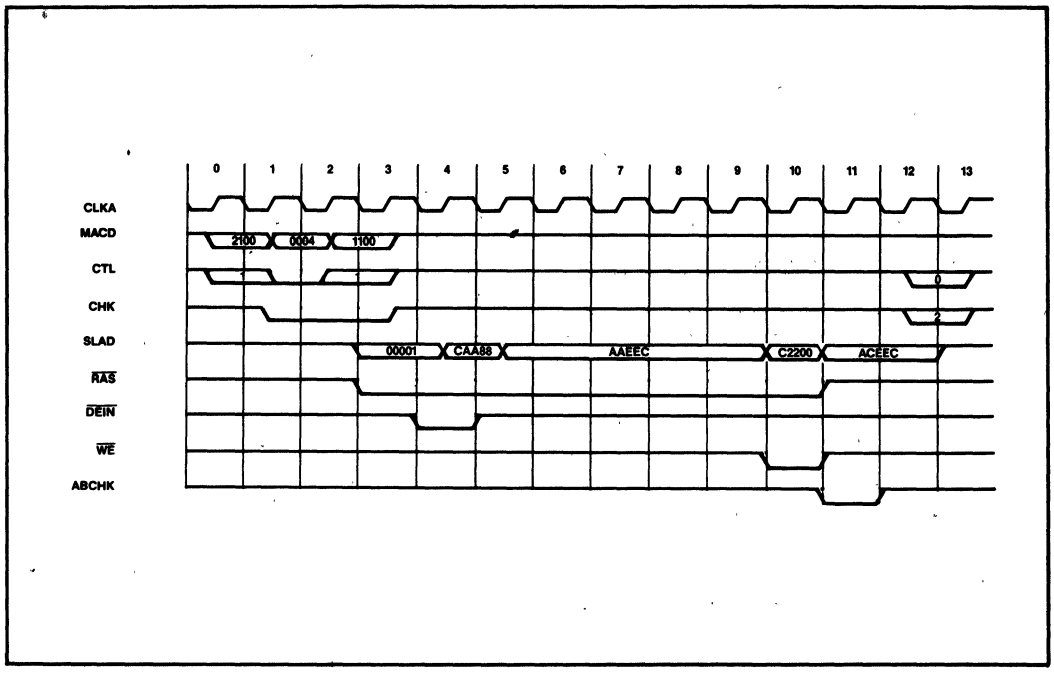


Figure 32. Normal 2-byte Write

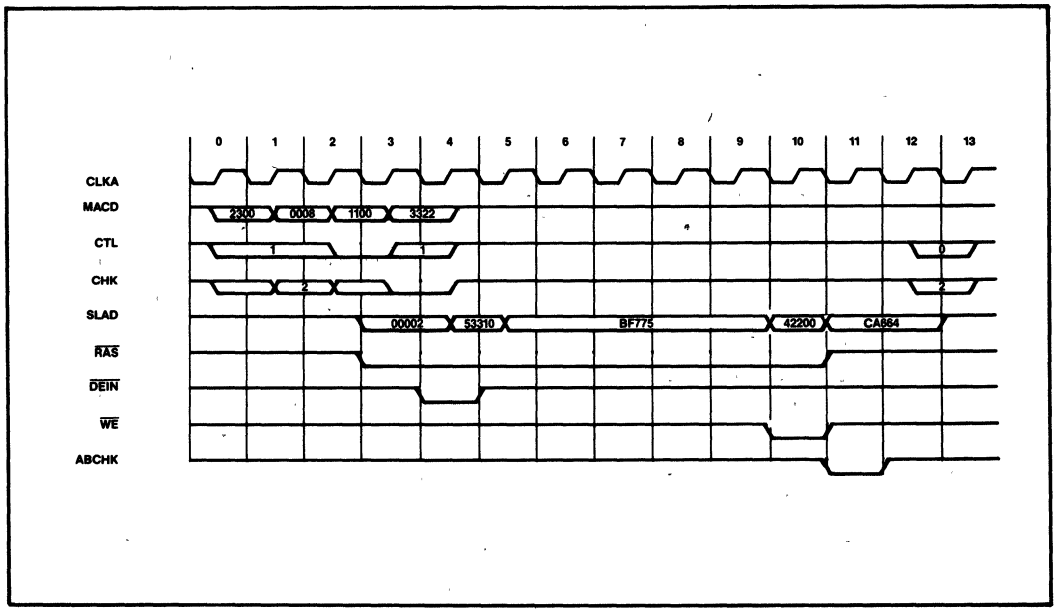


Figure 33. Normal Aligned Write

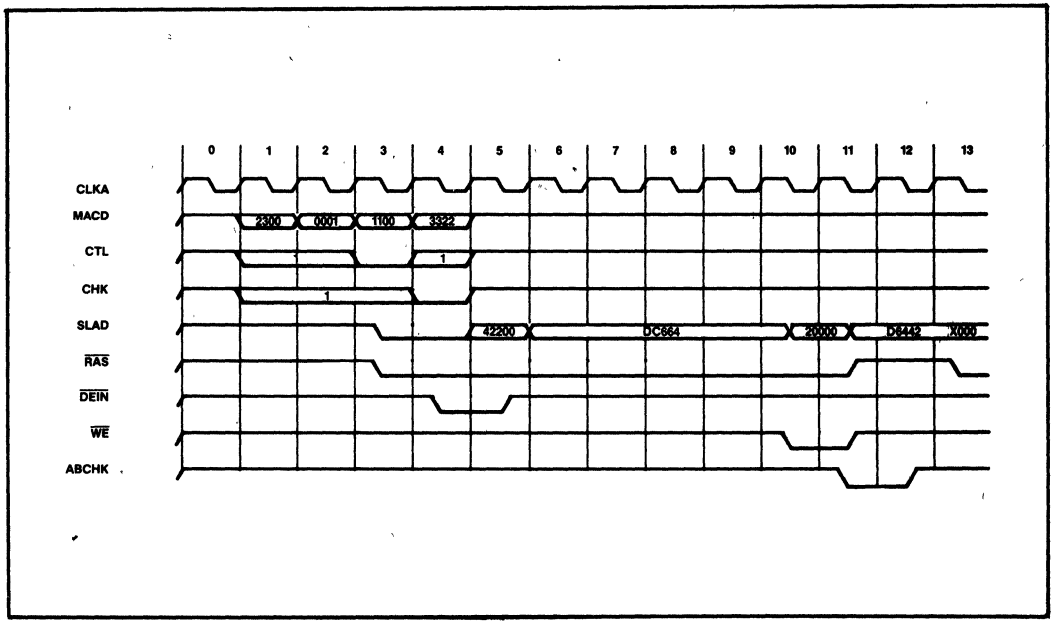


Figure 34. Normal Nonaligned Write

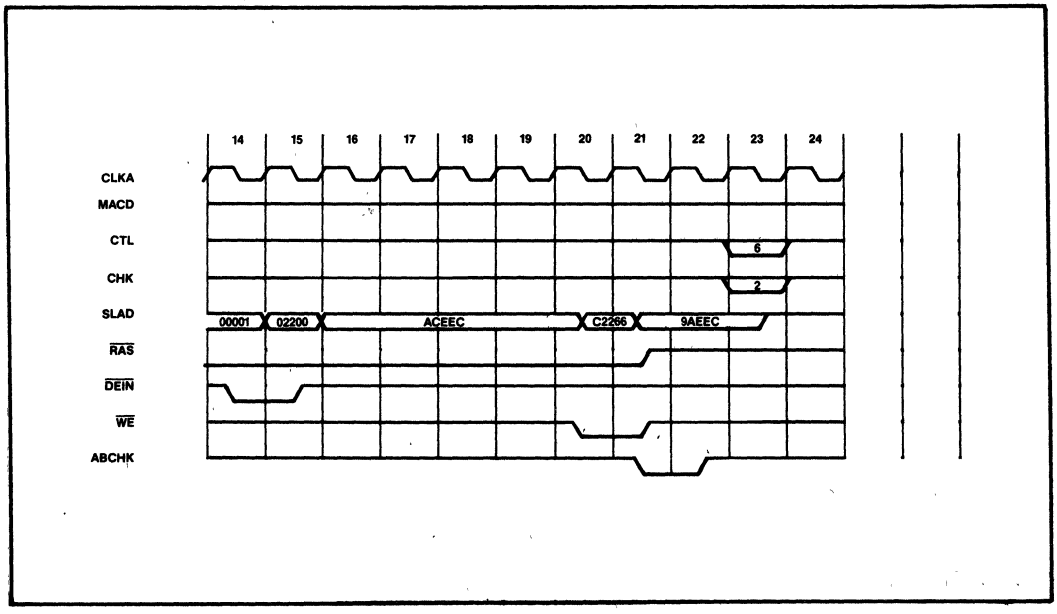


Figure 34. Normal Nonaligned Write (continued)

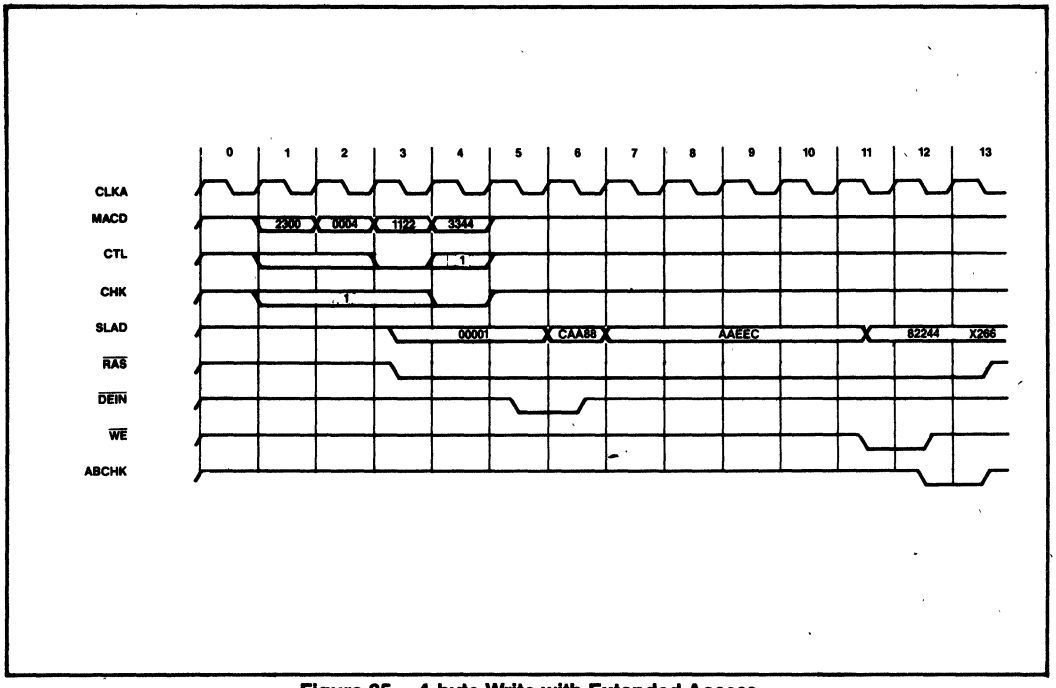


Figure 35. 4-byte Write with Extended Access

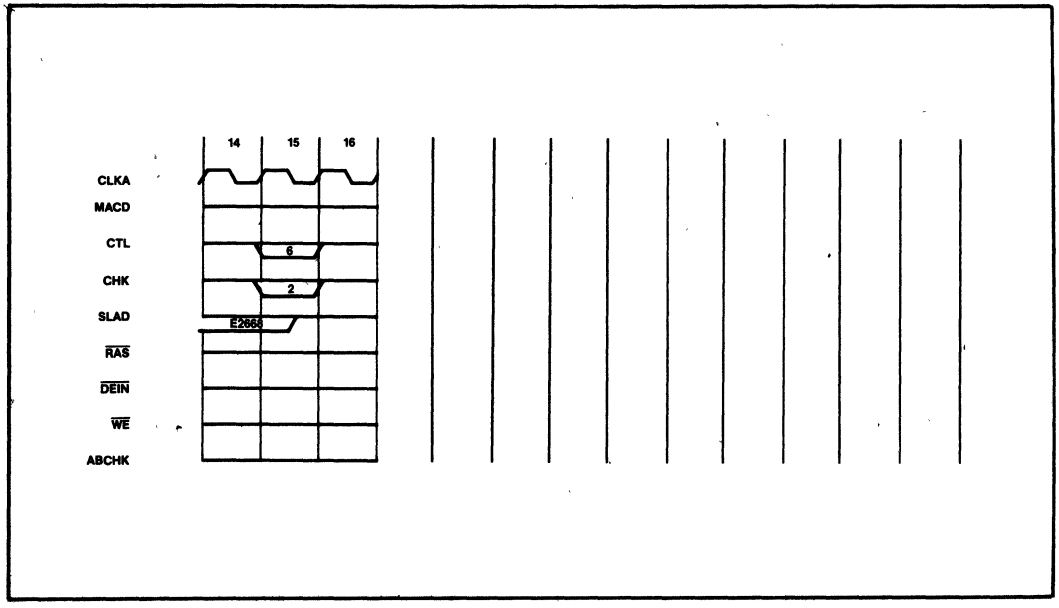


Figure 35. 4-byte Write with Extended Access (continued)

MEMORY REFRESHING OPERATIONS

Figures 36 and 37 illustrate an MCU performing two types of storage array refreshing operations. The MCU performs a *Normal Refresh Cycle* to satisfy standard dynamic RAM refresh requirements. External logic must use the REFRESH signal to generate the appropriate storage control signals (e.g., RAS-only refresh). The MCU performs a *Scrub Refresh Cycle* to cleanse the memory array of latent ECC errors by periodically reading the storage array and writing back a corrected version of data for any correctable errors it detects. Each of these diagrams utilize the same hardware configuration as the memory read and write operations described earlier. However, the MCU performs the refreshing operations independently of any commands from the MACD bus.

INTERCONNECT REGISTER OPERATIONS

Figures 38 and 39 illustrate the reading and writing of interconnect registers that are located on the MCU. Refer to the iAPX 432 Interconnect Architecture Reference Manual for a detailed description of specific interconnect registers. These diagrams illustrate general interconnect register operations. Later, specific commands are demonstrated that are invoked by interconnect register operations.

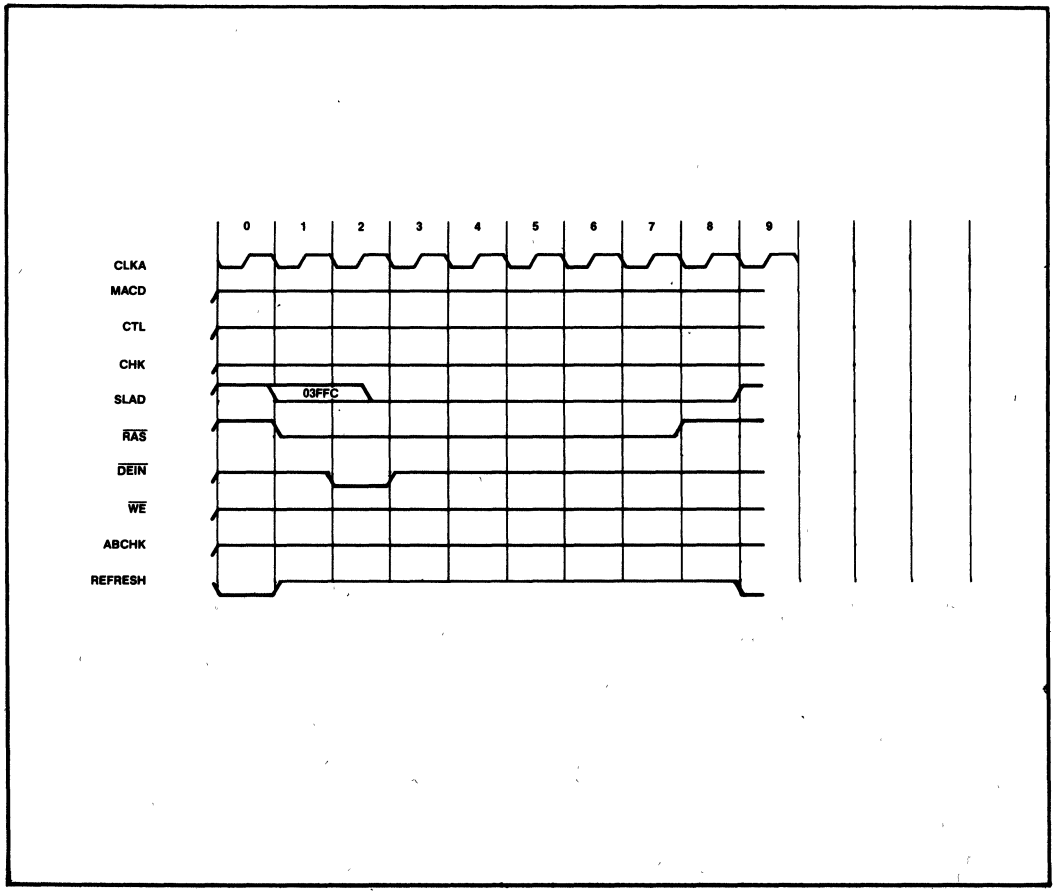


Figure 36. Normal Refresh Cycle

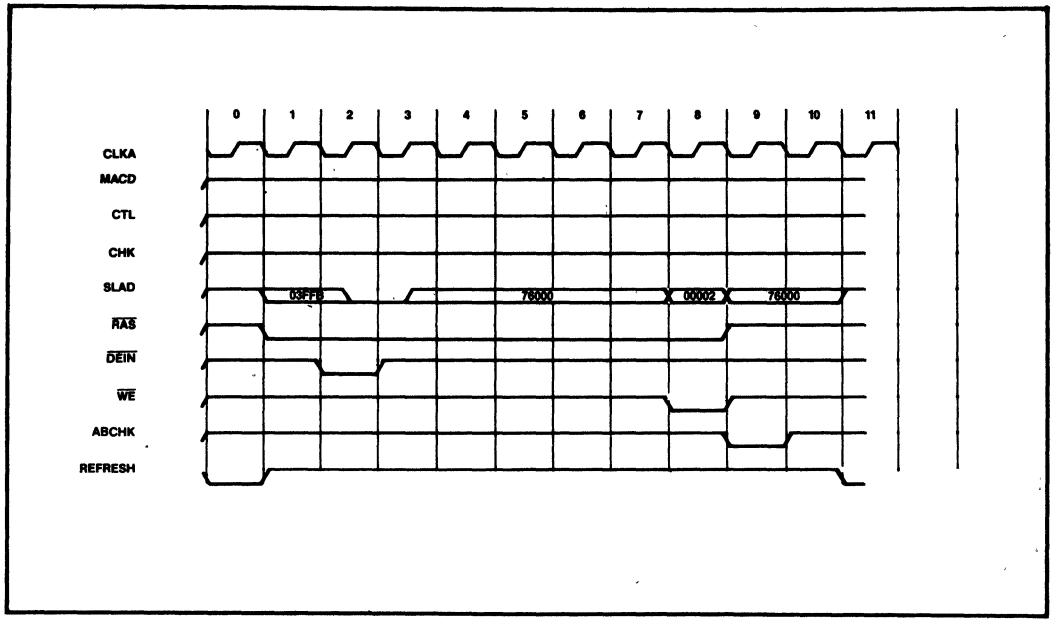


Figure 37. Scrub Refresh Cycle

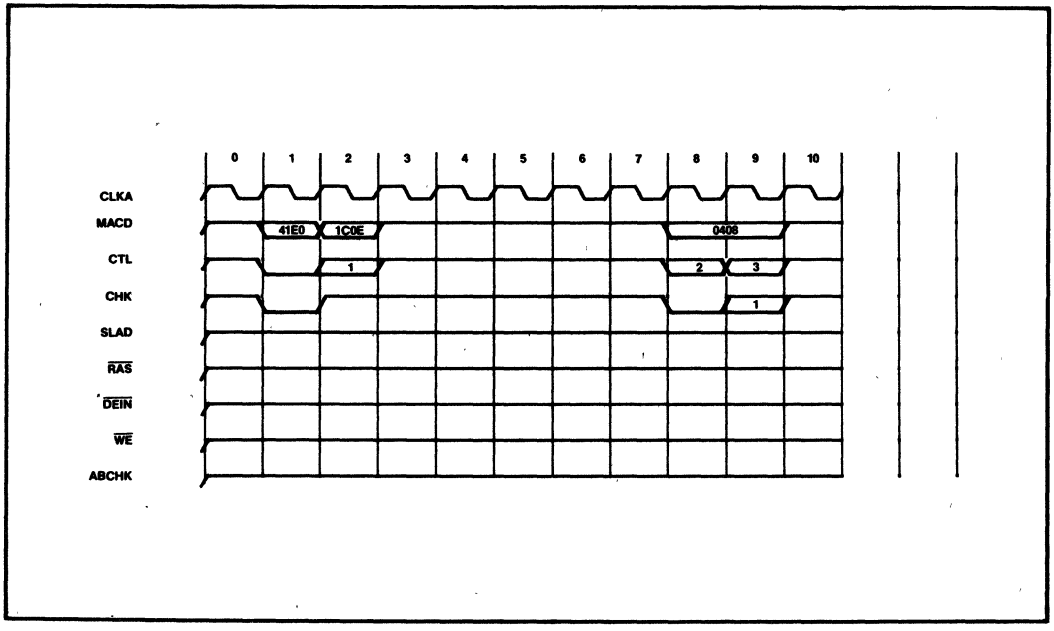


Figure 38. Interconnect Register Read

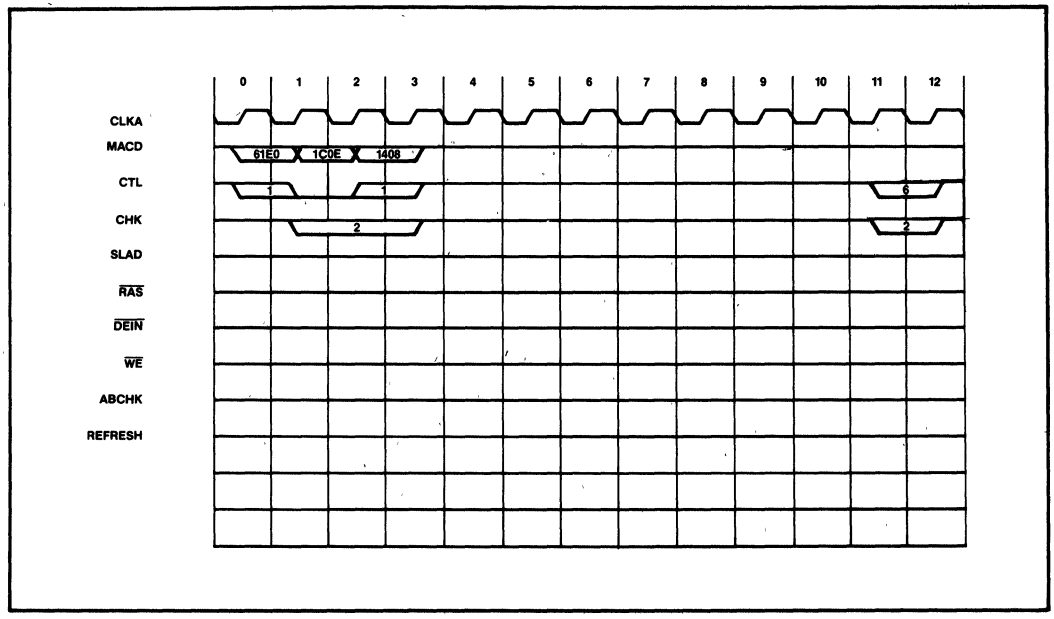


Figure 39. Interconnect Register Write

TEST DETECTION COMMAND

The Test Detection command is used to check that the detection circuits in the MCU are operating correctly. The command is invoked by an interconnect register write to the Test Detection register address (see the iAPX 432 Interconnect Architecture Reference Manual). The iAPX 432 instruction MOVE TO INTERCONNECT, executed on a GDP, causes the interconnect write cycle. The MCU reports the status of the test via the three-phase error report method described earlier. (See BERLOUT pin description.) If the test found no errors, the "no error" message is sent. If the test encountered an error, the "module error" message is sent.

Two Test Detection executions are shown in Figures 40 and 41, one each for the master and checker components in a FRC pair. Each of the diagrams consist of five parts. The two components operate identically except for the manner in which they check the BUSSEL signal (see cycles 10 and 11). The master operates the memory bus detection circuits with BUSSEL=0. The checker issues the complement, BUSSEL=1. External logic checks the two BUSSEL signals and reports the disagreement on the BCHKIN/M input.

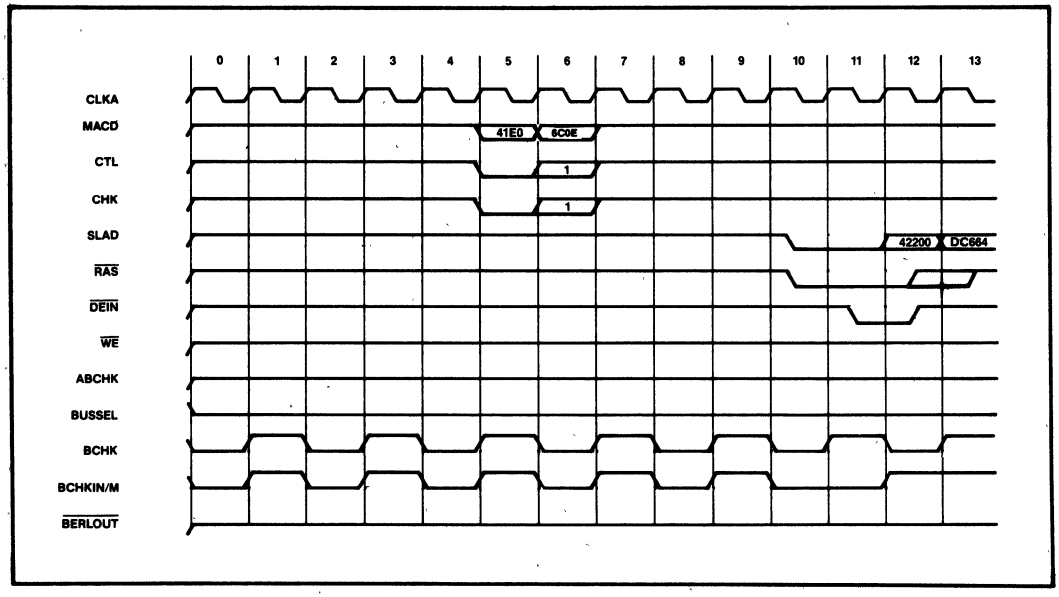


Figure 40. Test Detection—Master

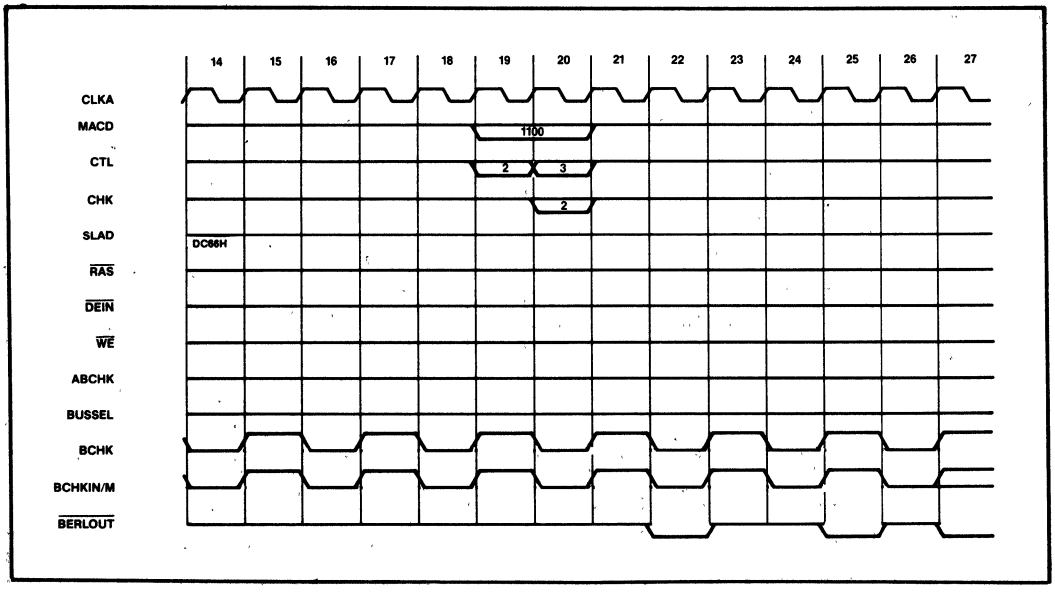


Figure 40. Test Detection—Master (continued)

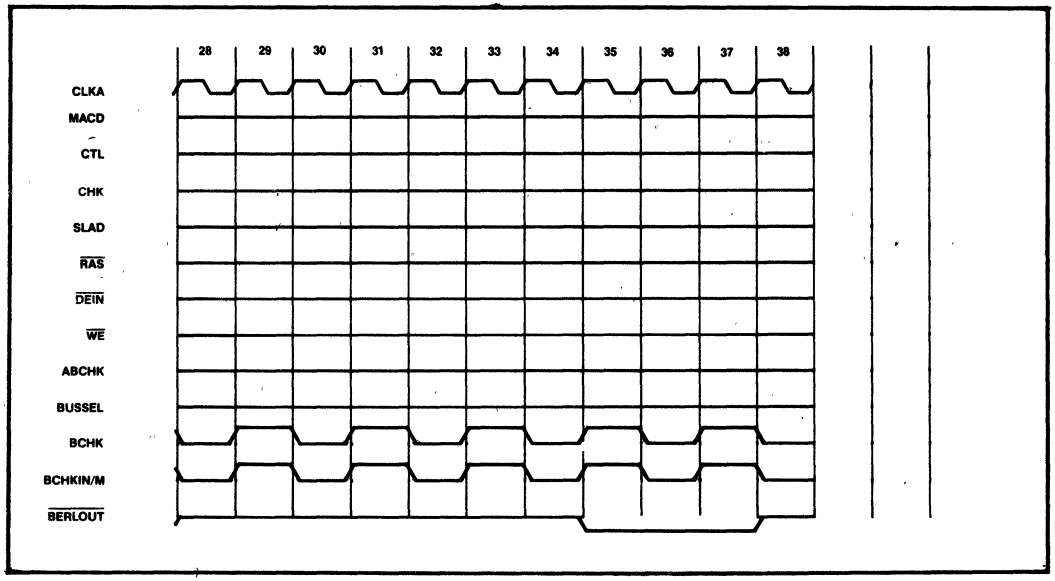


Figure 40. Test Detection—Master (continued)

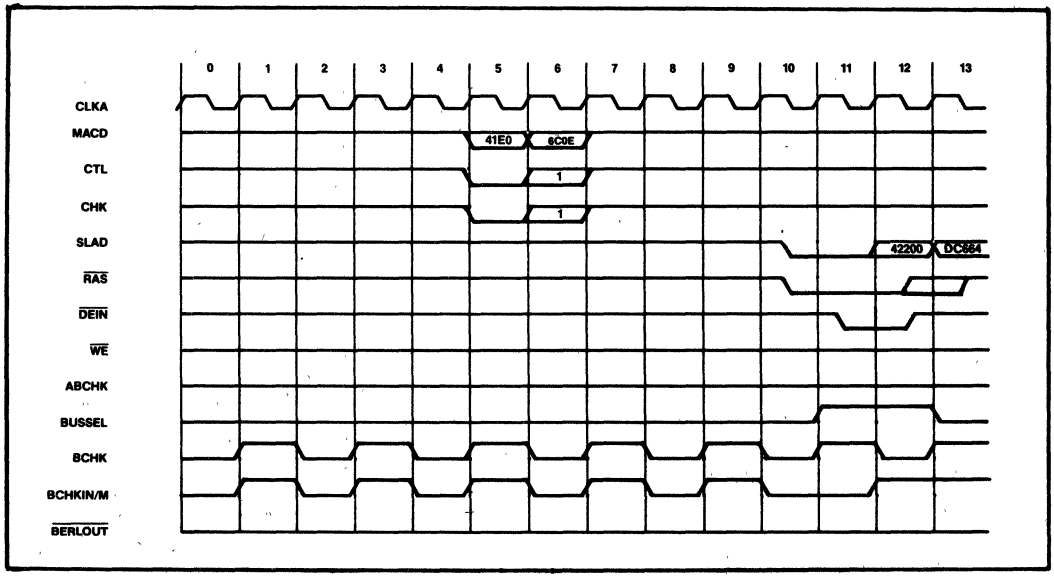


Figure 41. Test Detection—Checker

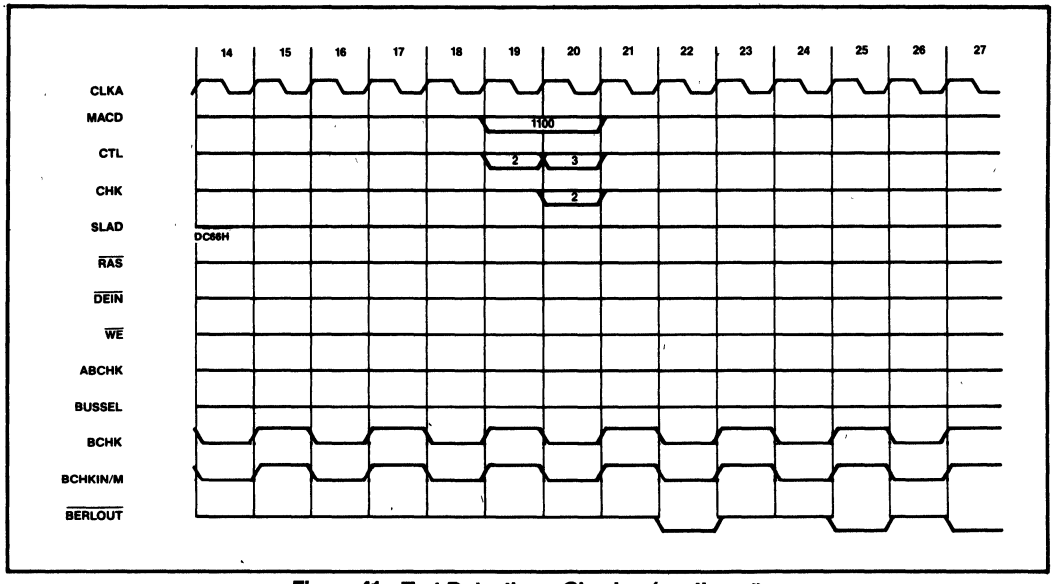


Figure 41. Test Detection—Checker (continued)

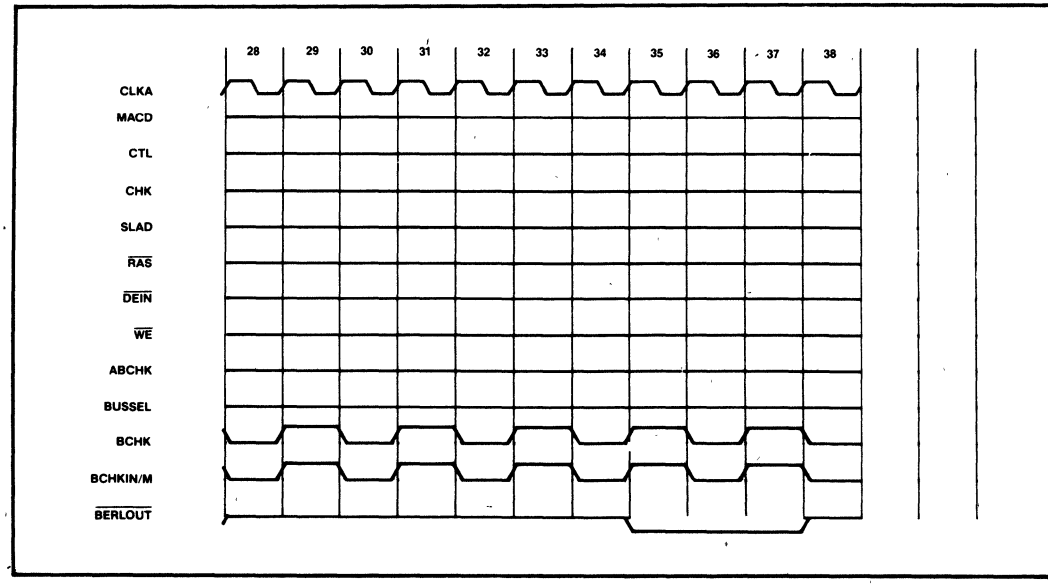


Figure 41. Test Detection—Checker (continued)

CLEAR MEMORY COMMAND

The Clear Memory command is invoked by an interconnect register write to the Clear Memory register. The MCU performs the command by writing zero data and appropriate ECC information to all storage array locations. The same hardware configuration that was used for memory read and write cycles is assumed. The MCU returns a reply to the requester as soon as it accepts the command but will not perform any further requests until it has cleared the memory. As seen in Figure 42, the MCU issues storage array write cycles beginning at the highest storage array address (in this case, 03FFFH) and decrements through remaining addresses (03FFEH, 03FFDH, . . .). Notice that the ECC and zero data information changes for each successive cycle (B000000000H, D200000000H, A600000000H, etc.). This occurs because ECC is computed across data *and* address information. This process continues until all storage locations have been cleared. Only the first few cycles of the process are demonstrated.

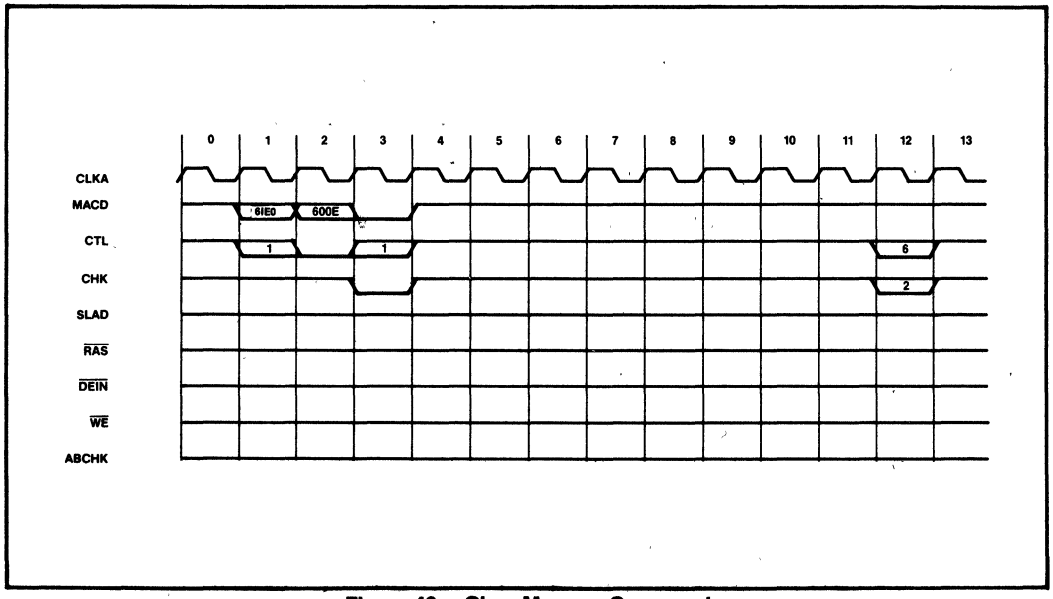


Figure 42. Clear Memory Command

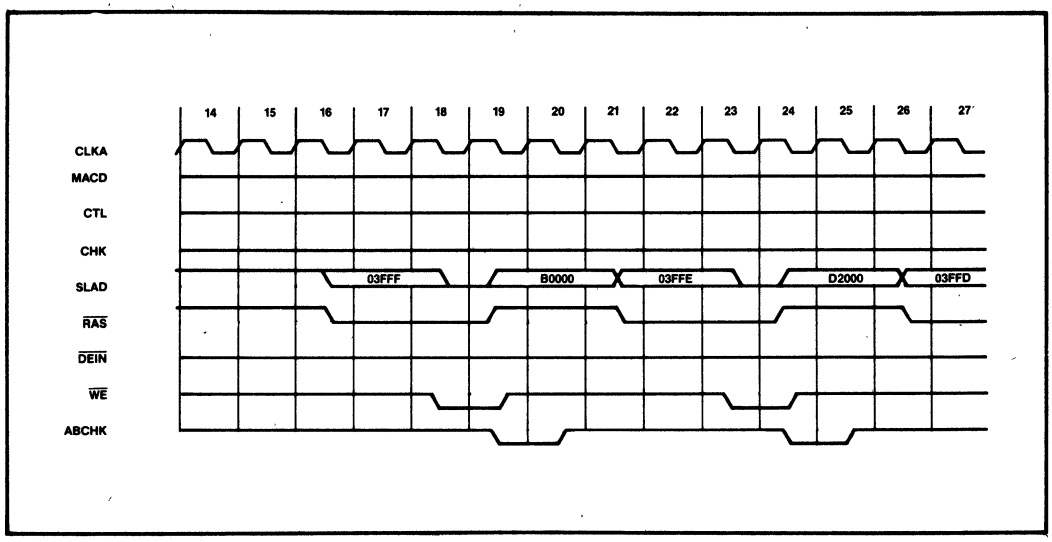


Figure 42. Clear Memory Command (continued)

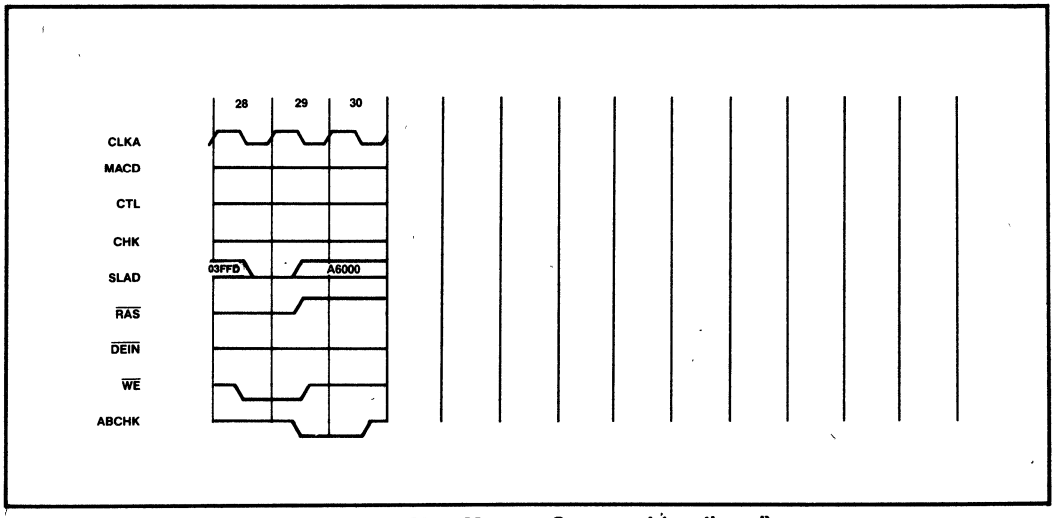


Figure 42. Clear Memory Command (continued)

Table 4. iAPX 43205 Clock Edge Table

Signal	Pin(s)	Input Sample		Output Drive	
		Clock	Edge	Clock	Edge
Inputs					
INIT	37	CLKA	Rising	n/a	
BERL1	38	CLKA	Rising	n/a	
BERL2	39	CLKA	Rising	n/a	
RQ	9	CLKA	Falling	n/a	
CONT	10	CLKA	Falling	n/a	
BCHKIN/M	4	CLKA	Rising	n/a	
ABCHK	45	CLKA	Rising	n/a	
Outputs					
RAS	48	n/a		CLKA	Falling
DEIN	47	n/a		CLKA	Falling
WE	46	n/a		CLKA	Falling
REFRESH	44	n/a		CLKA	Falling
BERLOUT	40	n/a		CLKA	Rising
BCHK	6	n/a		CLKB	Rising
BUSSEL	5	n/a		CLKA	Rising
Input/Output					
MACD15 . . . 0	13-28	CLKA	Rising	CLKB	Rising
CTL2 . . . 0	29-31	CLKA	Rising	CLKB	Rising
CHK1 . . . 0	11-12	CLKA	Rising	CLKB	Rising
MBOUT	32	CLKA	Rising	CLKA	Rising
SLAD19 . . . 0	1-2, 51-68	CLKA	Rising	CLKA	Falling

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Range ..... 0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −1.0V to +7V
Power Dissipation ..... 2.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

iAPX 43205 DC Characteristics

Symbol	Description	Min	Max	Units
Vilc	Clock input low voltage	−0.5	+0.8	V.
Vihc	Clock input high voltage	3.2	Vcc+0.5	V.
Vil	Input low voltage	−0.5	+0.8	V.
Vih	Input high voltage	2.0	VCC+0.5	V.
Vol	Output low voltage	—	0.45	V.
Voh	Output high voltage SLAD19..0 Other outputs	 2.6 2.4	 VCC VCC	 V. V.
Ili	Input leakage current (measured at Vin=VCC Volts)	—	±10	μA.
Ilo	Output leakage current (measured at 0.45 Volts ≤ Vout ≤ VCC Volts)	—	±10	μA.
Ioh	Output high current (measured at Vout=2.6 V.)	−2	—	mA.
Iol	Output low current (measured at Vout=0.45 V.)	4	—	mA.
Icc	Power supply current (sum of VCC0, VCC1, VCC2)	—	450	mA.

All DC parameters are guaranteed over the following conditions:

VSS2..VSS0 = 0 Volts
VCC2..VCC0 = 5.0 Volts ±10%
The absolute value of the differential DC voltage between any of the VCC pins (VCC2..VCC0) must be less than 0.1 Volts. This is normally guaranteed by connecting the three VCC pins to the same printed circuit power trace.

iAPX 43205 AC Characteristics

Ambient temperature range of 0°C to 70°C

Symbol	Description	5 MHz		7 MHz		8 MHz		Unit
		Min	Max	Min	Max	Min	Max	
t_r, t_f	Clock rise and fall times	-	13	-	11	-	10	nsec
t_1, t_2, t_3, t_4	Clock pulse width	37	250	25	250	24	250	nsec
t_{cy}	Clock cycle time	200	1000	143	1000	125	1000	nsec
t_{cd}	Clock to signal delay time	-	70	-	60	-	55	nsec
t_{dh}	Clock to signal hold time	20	-	17	-	15	-	nsec
t_{en}	Clock to signal output enable time	20	-	17	-	15	-	nsec
t_{df}	Clock to signal data float time	-	50	-	44	-	40	nsec
t_{dc}	Signal to clock setup time	30	-	26	-	24	-	nsec
t_{mc}	MACD input setup time	30	-	26	-	24	-	nsec
t_{ie}	Initialization period	40	100	40	100	40	100	t_{cy}

All AC parameters are guaranteed over the following conditions:

Ambient temperature range of 0 degrees Centigrade to 70 degrees Centigrade

VSS2 . . . VSS0 = 0 Volts
VCC2 . . . VCC0 = 5.0 Volts ± 10%

100 picoFarad external load capacitor on all output pins

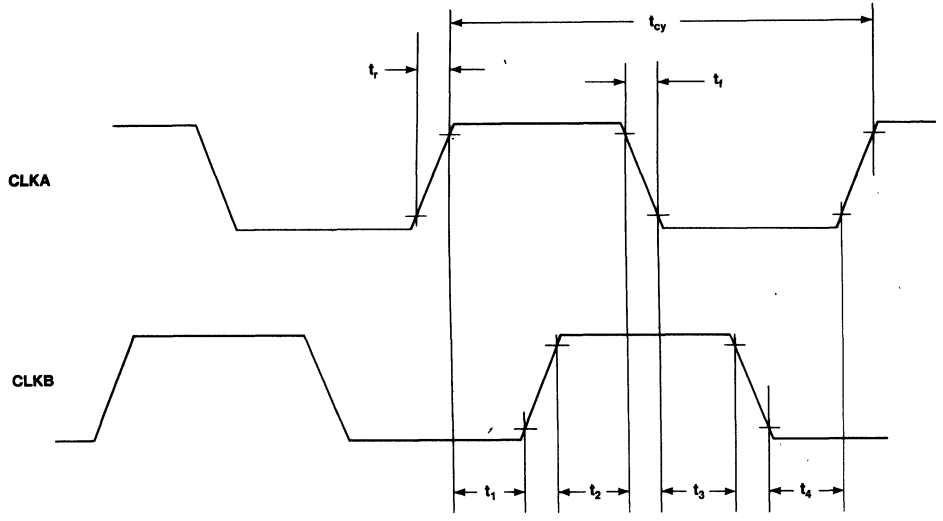
iAPX 43205 Capacitance Data

Conditions: Ta = 25°C
VCC = 5.0 Volts, GND = 0.0 Volts
f(test) = 1.0 MHz
Inputs held at 0.0 Volts
All outputs in high impedance state
All input/output pins are classified as outputs

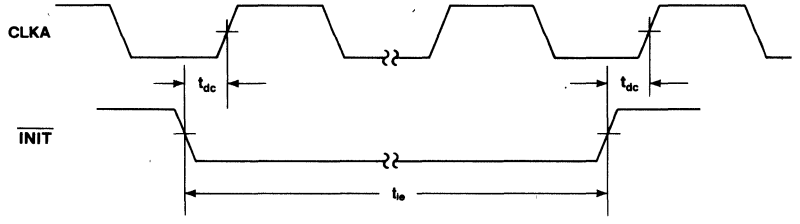
Symbol	Description	Max	Units
Cin	Input Capacitance	6	pF
Cout	Output Capacitance	12	pF

WAVEFORMS

Clock Input Timing Specification

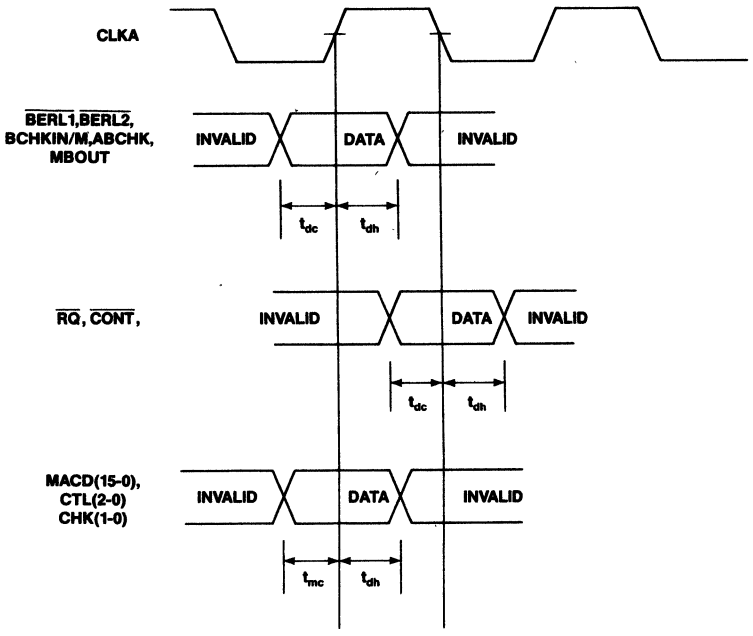


Initialization Timing Specification



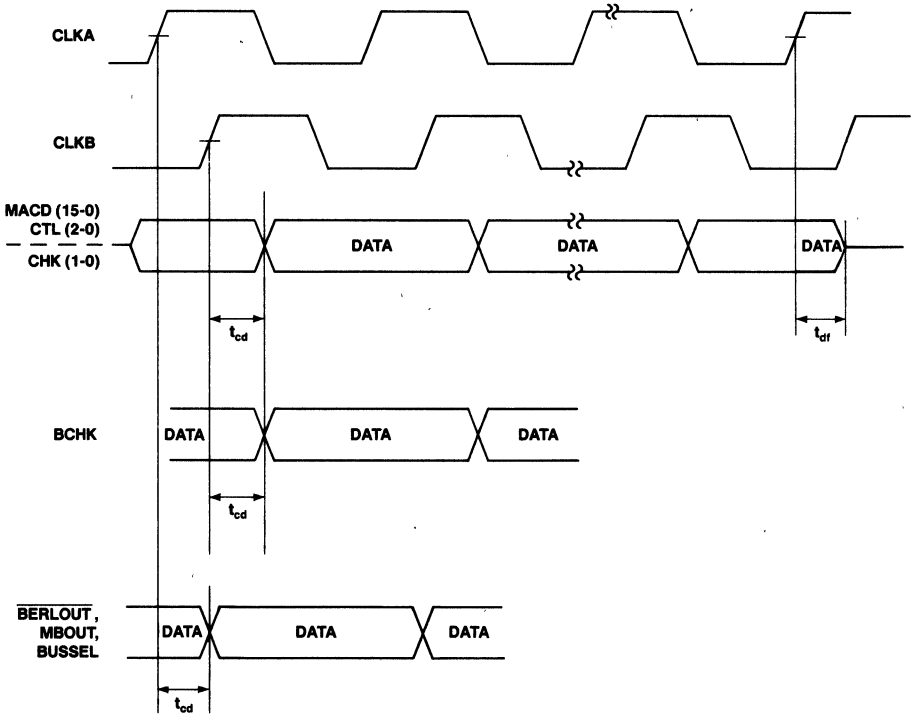
WAVEFORMS (Continued)

iAPX 43205 Input Timing Specification



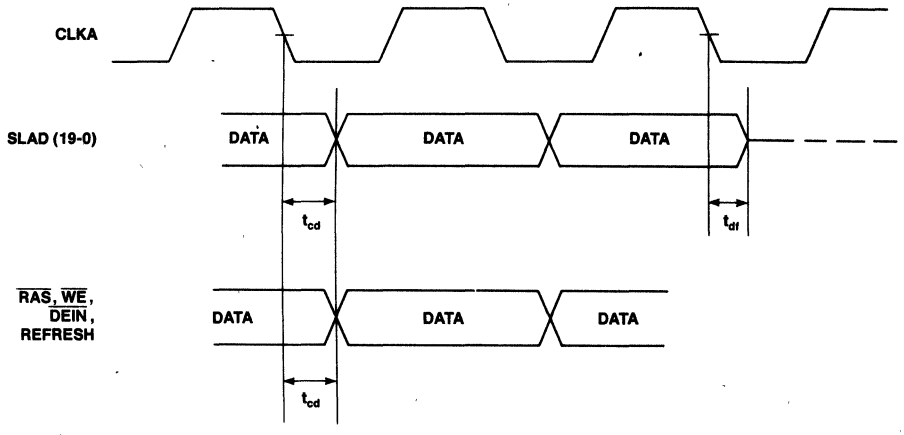
WAVEFORMS (Continued)

iAPX 43205 Output Timing Specification

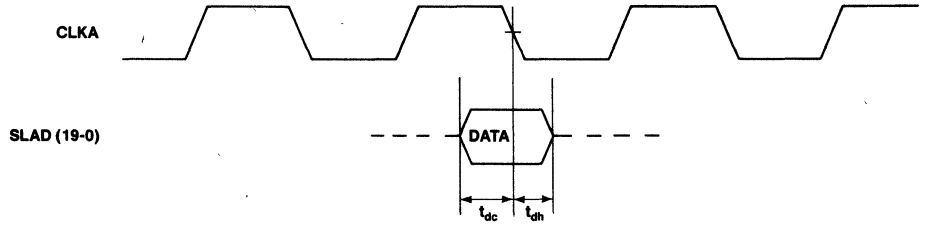


WAVEFORMS (Continued)

Storage Array Bus Output Timing Specification



Storage Array Bus Input Timing Specification



HARDWARE INTERFACING

This section presents examples of hardware that interfaces iAPX 432 processors (the iAPX 43201/43202 General Data Processor (GDP) and the iAPX 43203 Interface Processor (IP), the iAPX 43204 Bus Interface Unit (BIU), and the iAPX 43205 Memory Control Unit (MCU) to one another (see Figure 44). These examples present some alternatives for building iAPX 432 systems using the interconnect components. A wide variety of systems may be built with the interconnect components and this list of examples explores only a few dimensions of the design space.

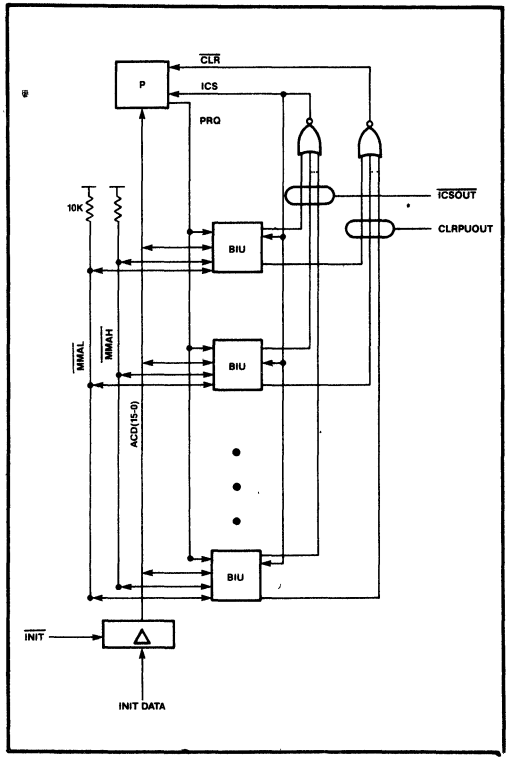


Figure 44. Interfacing iAPX 432 Processors to the BIU

Figure 45 illustrates the connection of a BIU to the serial error reporting networks. Here, the BIU connects to the module error report line (MERL) and has a duplicated bus error report line (BERL1 and BERL2). Each error report line is driven by open collector inverters so that the BIUs along either the module or bus axes may contribute error messages in wired-OR fashion. Two inverters are required to drive each error report line, one is a standard inverter and the other is an open collector version. The example shown also duplicates the bus error report lines so that bus error messages may be delivered even if one of the inverter paths fails. Should duplicated bus error report lines not be required, one of the two inverter groups could be deleted and the BERL1 and BERL2 signals connected together.

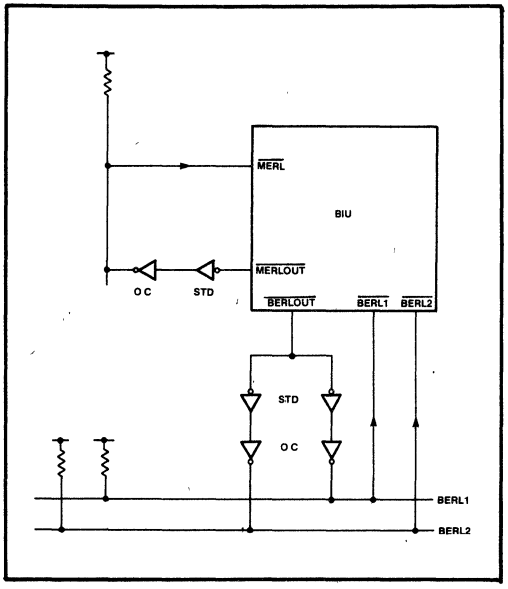


Figure 45. Interfacing a BIU to Error Reporting Network

The 432 hardware system designer must provide an external arbitration network which examines the BIU's arbitration signals, along with other BIUs in the system, to determine when it is permissible to use the memory bus. Two alternatives for this function are presented in Figures 46 and 47. The first method requires the fewest backplane signal lines since it utilizes multilevel *analog* signalling to arbitrate for the bus. The second method requires more backplane signals and unique wiring for each arbitrating unit but is fully *digital*.

With either method, each BIU produces two output signals, RQOUT and NREQOUT, which are activated to indicate that the BIU requires the use of the memory bus. Each BIU requires that an external logic network examine all the NREQOUTs and RQOUTs to identify when a request is being made by one or more BIUs. Three inputs to the BIU (NREQ, RQ and CONT) are generated by the external logic. NREQ (New Request) is activated to signal that a new time-ordering cycle has occurred. RQ (request) signifies that one or more RQOUTs are active and CONT (contention) signifies that more than one RQOUT is active.

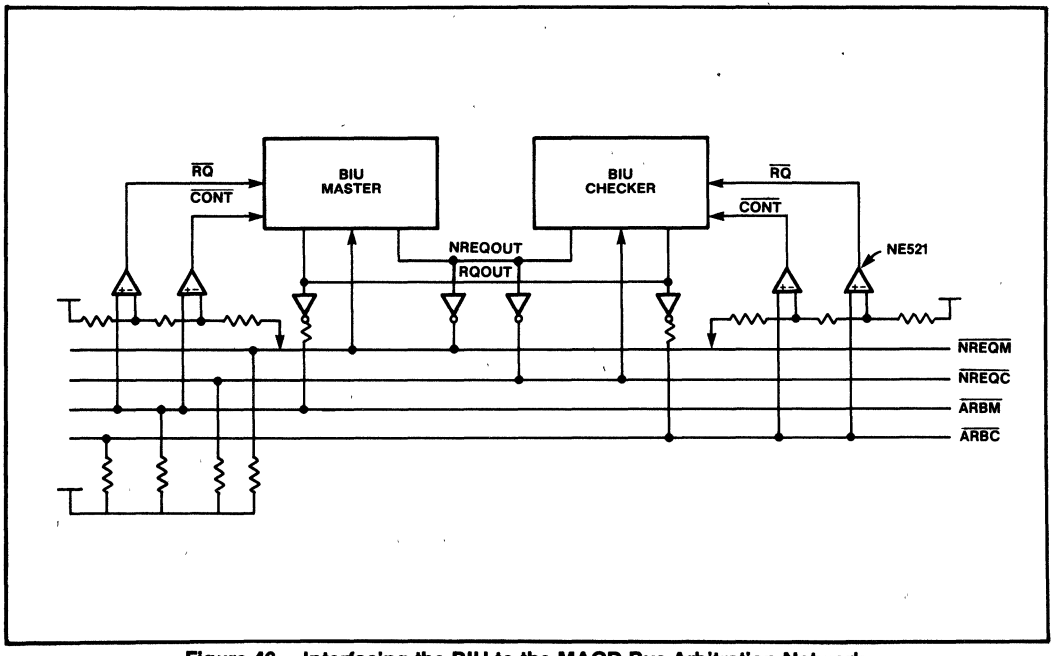


Figure 46. Interfacing the BIU to the MACD Bus Arbitration Network (Analog Method)

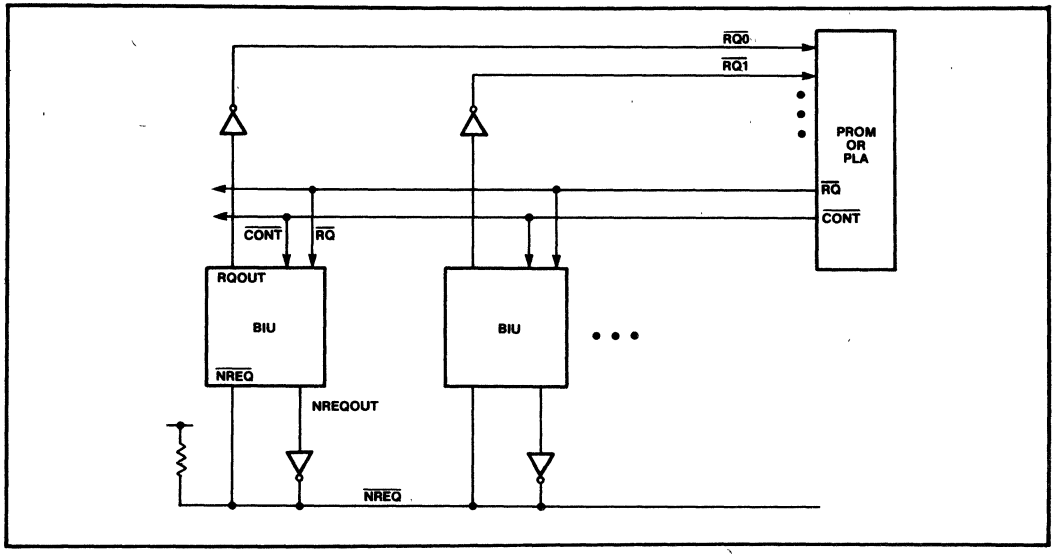


Figure 47. Interfacing the BIU to the MACD Bus Arbitration Network (Digital Method)

Figure 48 shows the required bus transceivers which connect a BIU to a memory bus. In addition, the drawing suggests how the oscillating BCHK signal may be used to check that the external buffers are operating correctly.

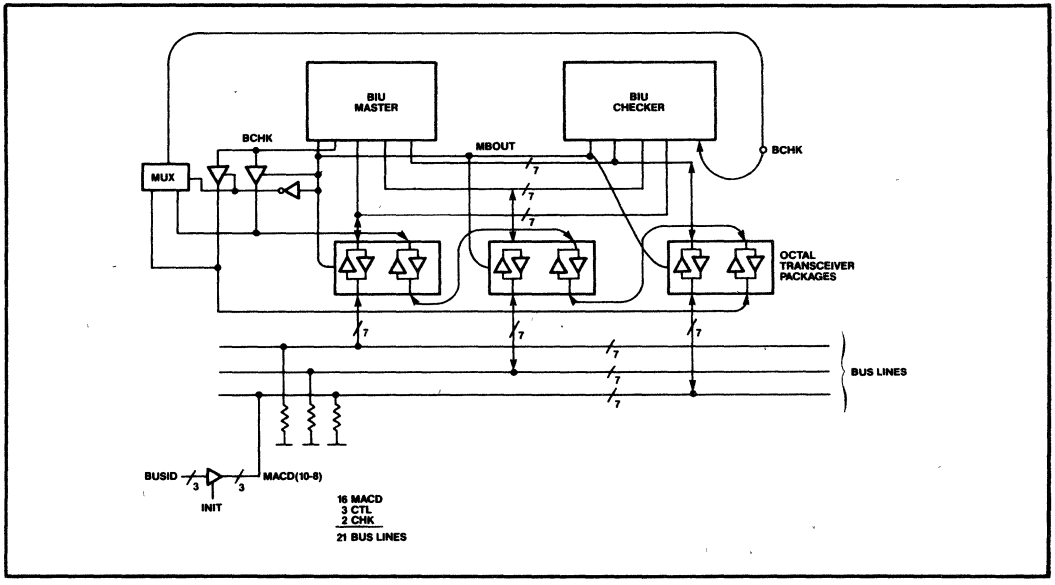


Figure 48. Interfacing the BIU to the MACD BUS

Figure 49 illustrates how an MCU, an external RAM storage array, and external sequencing logic form a memory subsystem. In this example, economical dynamic RAM components form the storage arrays and the array sequencing logic, under control of the MCU, provides the precise signals required to manage the arrays. Figures 50 and 51 detail the array sequencing logic and the timing of signals which coordinate the actions of the storage array.

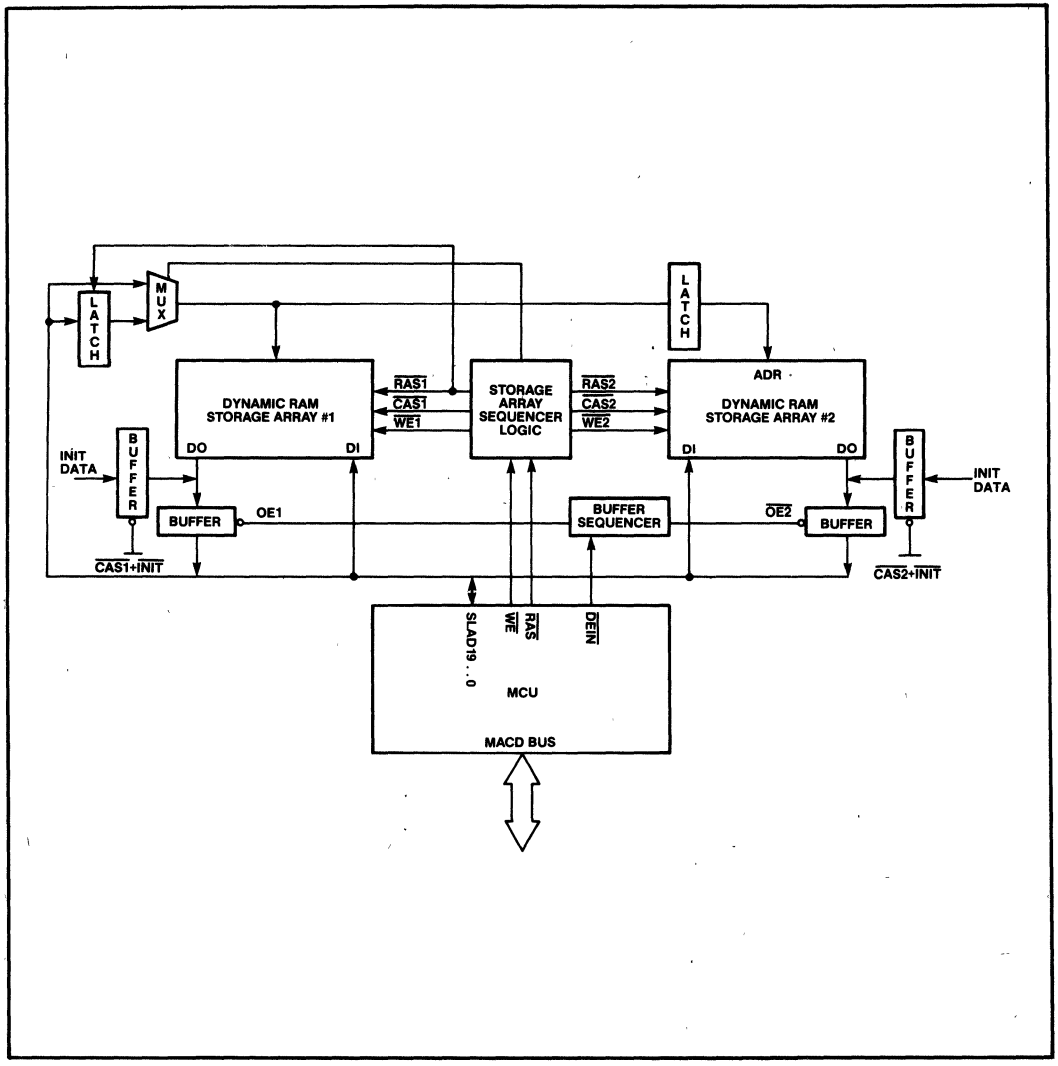


Figure 49. Interfacing the MCU to the Storage Array

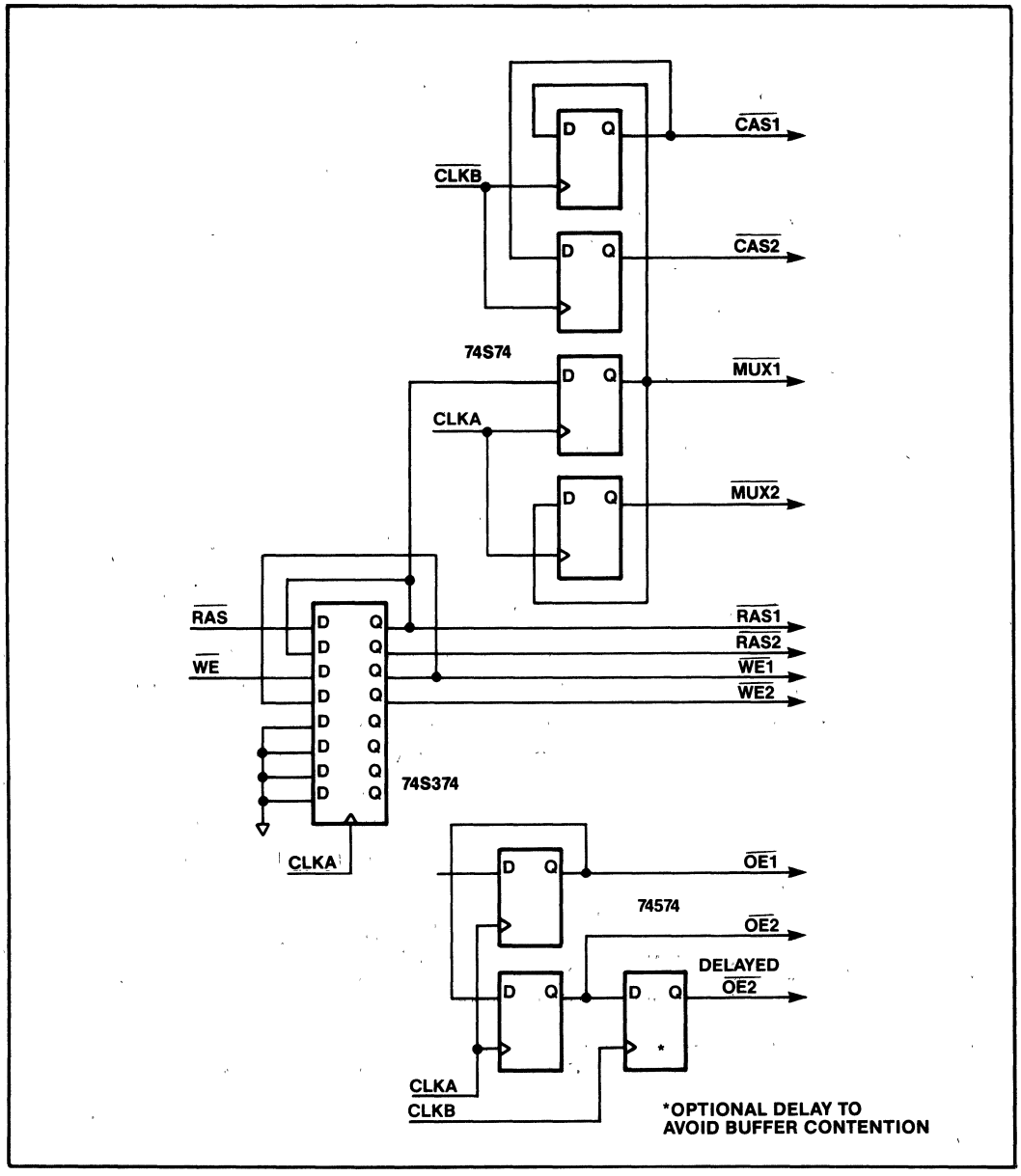


Figure 50. Sequencing Logic for the Storage Array

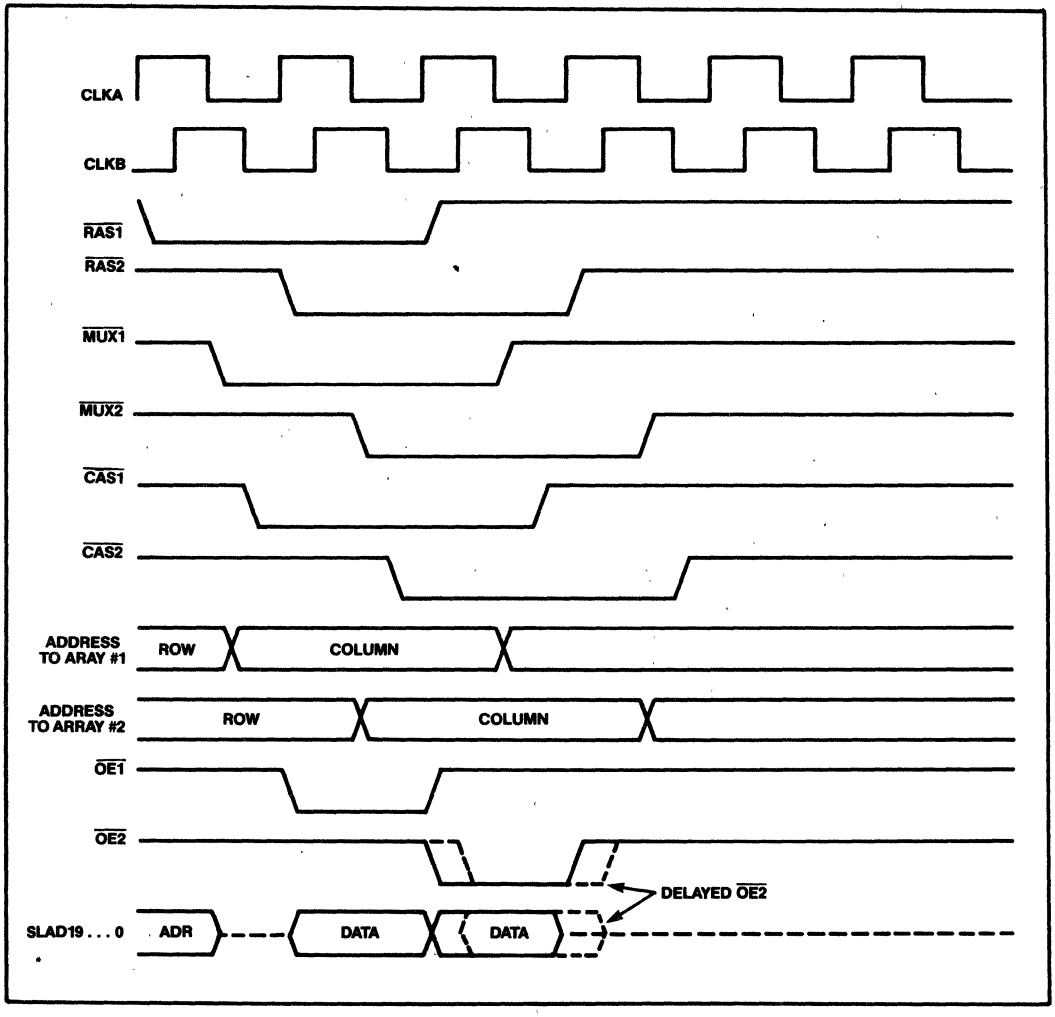


Figure 51. Timing Diagram for the Storage Array Interface

External hardware must be employed to permit an MCU to attach to its normal or backup memory bus. There are several facets to this requirement. Naturally, individual *bus transceivers* must be used to allow the MCU, through its BUSSEL (bus select) output signal, to choose which memory bus will carry its address, control, data, and check information. In addition, the MCU must access *error report* information and *arbitration* information from the correct memory bus. These requirements are met by the sort of network illustrated in Figure 52. Notice that these requirements are unique to the MCU. One BIU is required for *each* bus that a processor wishes to attach to, no such steering is required for a BIU. The next example highlights some of the considerations when extending these requirements to fault tolerant systems.

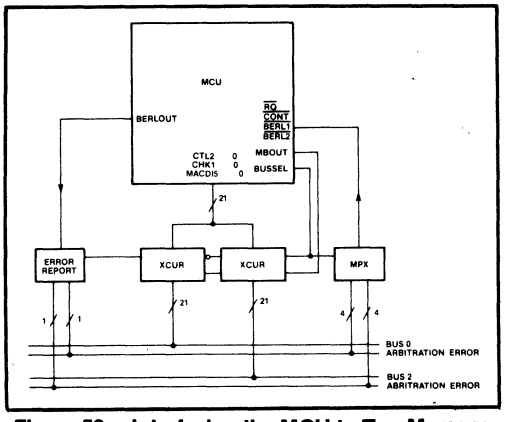


Figure 52. Interfacing the MCU to Two Memory Busses

Fault tolerant MCU configurations require that special external logic enable the transceivers which connect the MCU to its assigned memory bus. This may be done with a scheme illustrated in Figure 53. Two MCUs are employed in a FRC configuration. At their FRC interface, all the MACD, CTL, CHK signals as well as the MBOUT direction signal are compared for error. The master and checker MCUs each develop a version of the BUSSEL signal. Since it is not possible to FRC the BUSSEL signal and guarantee a correctly operating version, external hardware must be employed to develop a fault tolerant version of the bus selection function. The individual BUSSELs must be checked by external fault tolerant

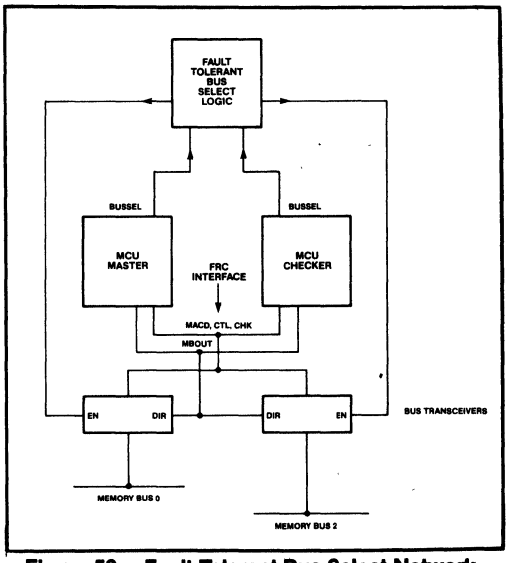


Figure 53. Fault Tolerant Bus Select Network

logic which enables only one of the two sets of memory bus transceivers when the BUSSELs are both active. If there is any discrepancy in the BUSSELs, the external logic must disable *both* of the bus transceiver sets so that the malfunctioning MCU cannot corrupt either memory bus.

This same technique must also be applied to other signals which must be routed to/from the currently assigned memory bus. The BERLOUT error report signal must only be output to the correctly selected bus. The $\overline{RQ}$, $\overline{CONT}$, $\overline{BERL1}$, and $\overline{BERL2}$ signals must only be received from the correctly selected bus.

Special logic is also required to check those signals which are not FRCed in a fault tolerant configuration. The BCHK output pins and the BCHKIN/M input pins of the master and checker MCUs may be used to detect errors in the external logic. In Figure 54, a PROM is used as the error detector. The inputs to the PROM may come from a variety of sources, depending on the particular hardware configuration. In this example, the PROM observes two sets of signals, one set from the master and another from

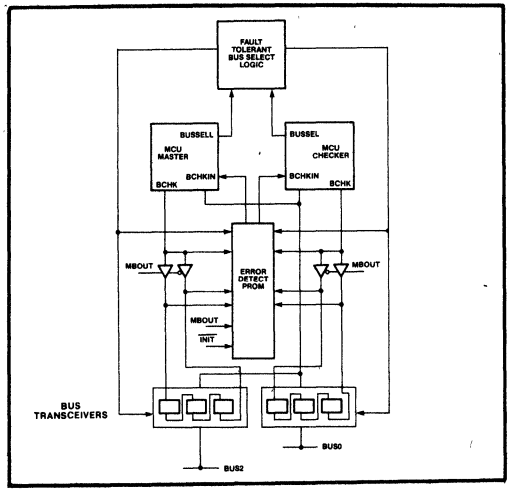


Figure 54. Detecting Errors in External MCU Logic

the checker. Notice that the oscillating BCHK signal is routed through the bus transceivers in different directions depending on the value of the MBOUT signal. The fault tolerant versions of the BUSSEL signal select collection of signals to be checked. The $\overline{\text{INIT}}$ input to the PROM provides a convenient way to establish master/checker roles during initialization since BCHKIN/M carries mastership information at that time.

PACKAGE

The 43204 and 43205 are packaged in 68-pin, leadless JEDEC type A hermetic chip carriers. Figure 55 illustrates the package, and Figures 9 and 11 show the pinouts.

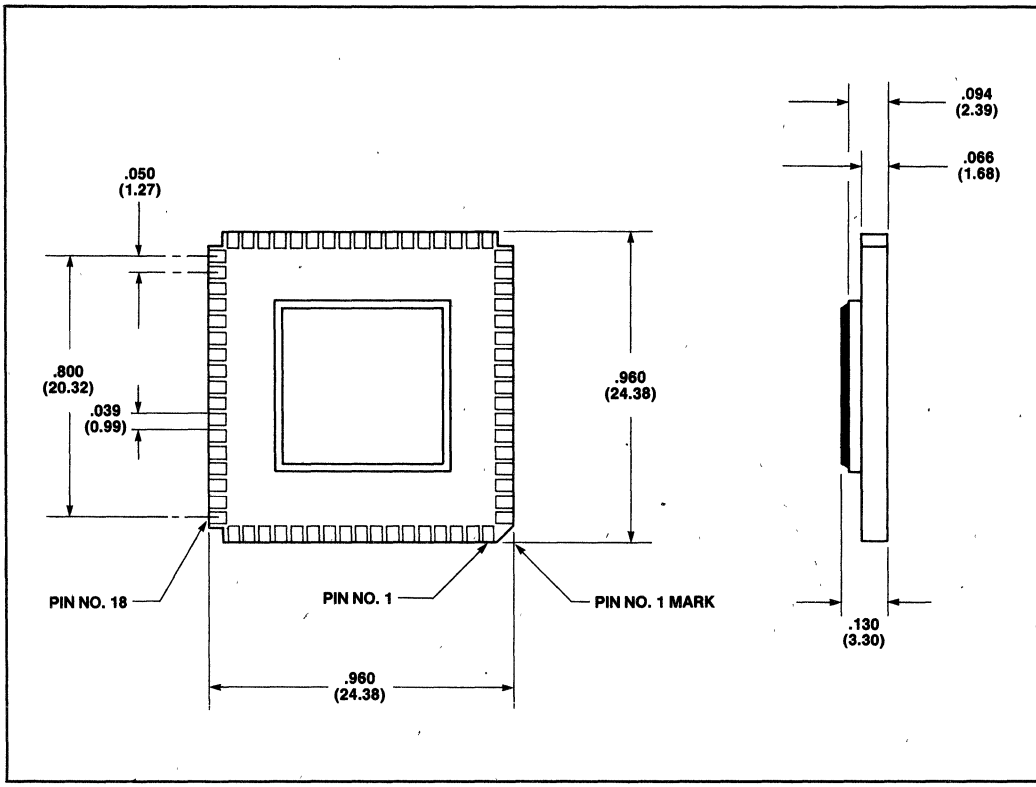


Figure 55. 43204 and 43205 JEDEC Type A Package

Peripherals 6

Peripherals Section

April 1982

Interfacing Dynamic RAM to iAPX 86, 88 Systems Using the Intel 8202A and 8203

Brad May
Peripheral Component
Applications Engineering

ORDER NUMBER: 210398-001

INTRODUCTION

The designer of a microprocessor-based system has two basic types of devices available to implement a random access read/write memory — static or dynamic RAM. Dynamic RAMs offer many advantages. First, dynamic RAMs have four times the density (number of bits per device) of static RAMs, and are packaged in a 16-pin DIP package, as opposed to the 20-pin or larger DIPs used by static RAMs; this allows four times as many bytes of memory to be put on a board, or alternatively, a given amount of memory takes much less board space. Second, the cost per bit of dynamic RAMs is roughly one-fourth that of statics. Third, static RAMs use about one-sixth the power of static RAMs, so power supplies may be smaller and less expensive. These advantages are summarized in Table 1.

On the other hand, dynamic RAMS require complex support functions which static RAMs don't, including

- address multiplexing
- timing of addresses and control strobes
- refreshing, to prevent loss of data
- arbitration, to decide when refresh cycles will be performed.

Table 1. Comparison of Intel Static and Dynamic RAMs Introduced during 1981

	2164-15 (Dynamic)	2167-70 (Static)
Density (No. of bits)	64K	16K
No. of pins	16	20
Access time (ns)	150	70
Cycle time (ns)	300	70
Active power (ma)	60	125
Standby power (ma)	5	40
Approx. cost per bit (millicents/bit)	45	250

In addition, dynamic RAMs may not always be able to transfer data as fast as high-performance microprocessors require; wait states must be generated in this case. The circuitry required to perform these functions takes up board space, costs money, and consumes power, and so detracts from the advantages that make dynamic RAMs so appealing. Obviously, the amount of support circuitry should be minimized.

The Intel 8202A and 8203 are LSI dynamic RAM controller components. Either of these 40-pin devices alone does all of the support functions required by dynamic RAMs. This results in a minimum of board space, cost, and power consumption, allowing maximum advantage from the use of dynamic RAMs.

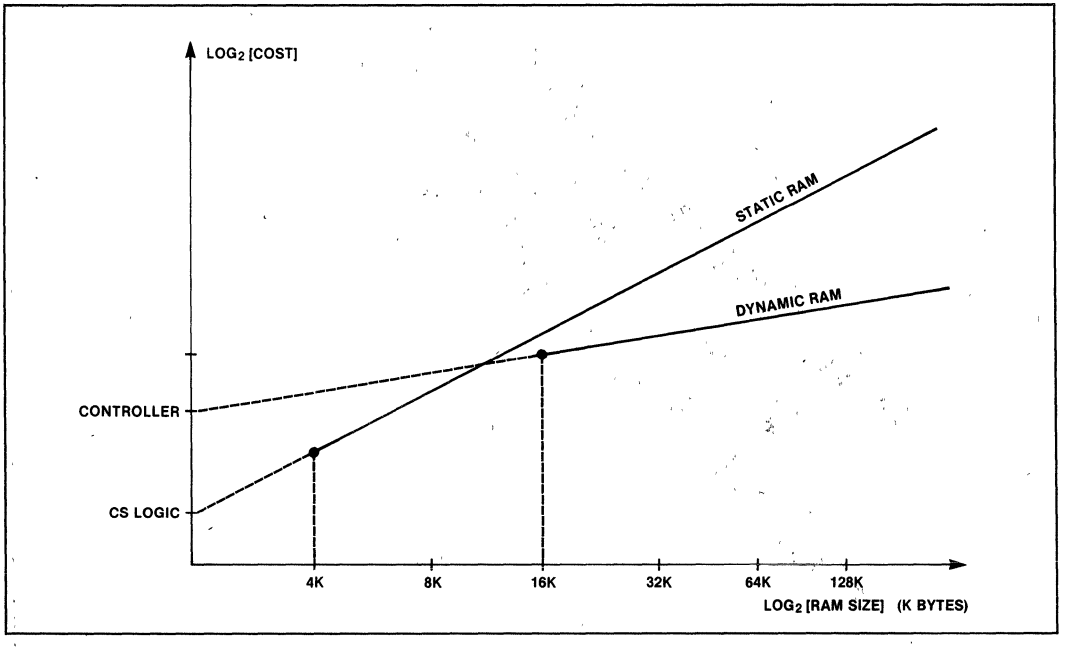


Figure 1. Implemented Cost of Static vs. Dynamic RAM

Figure 1 shows the relative cost of static and dynamic RAM, including support circuitry, as a function of memory size, using the Intel 8202A or 8203. For any memory larger than 16KBytes, the dynamic RAM is less expensive. Since the cost of the dynamic RAM controller is relatively independent of memory size, the cost advantge for dynamic RAM increases with increasing memory size.

This Application Note will describe the techniques of interfacing a dynamic RAM memory to an iAPX-86 or iAPX-88 system using either the 8202A or 8203 dynamic RAM controller. Various configurations of the 8086 and 8088 microprocessors, and those timings which they satisfy, are described. The Note concludes with examples of particular system implementations.

DYNAMIC RAMS

This section gives a brief introduction to the interfacing requirements for Dynamic RAMs. Later sections will describe the operation of the Intel 8202A and 8203 Dynamic RAM Controllers.

Device Description

The pinout of two popular families of dynamic RAMs, the Intel 2118 and 2164A, are shown in Figure 2. The 2118 is a 16,384 word by 1-bit dynamic MOS RAM. The 2164 is a 65,536 word by 1-bit dynamic MOS RAM. Both parts operate from a single +5v supply with a ± 10% tolerance, and both use the industry standard 16-lead pinout.

The two parts are pinout-compatible with the exception of the 2164 having one extra address input (A_7, pin 9); this pin is a no-connect in the 2118. Both parts are also compatible with the next generation of 256K dynamic RAMs (262,144 word by 1-bit), which will use pin 1 (presently a no-connect on both the 2118 and 2164A) for the required one extra address input (A_8). This makes it possible to use a single printed circuit board layout with any of these three types of RAM.

Addressing

Each bit of a dynamic RAM is individually addressable. Thus, a 2164A, which contains 2^{16} (or 65,536) bits of information, requires 16-bit addresses; similarly, the 2118, which contains 2^{14} (or 16,384) bits, requires 14-bit addresses.

In order to reduce the number of address pins required (and thus reduce device cost), dynamic RAMs time-multiplex addresses in two halves over the same pins. Thus a 2164A needs only 8 address pins to receive 16-bit addresses, and the 2118 needs only 7 for its 14-bit addresses. The first address is called the *row* address, and the second is called the *column* address. The row address is latched internal to the RAM by the falling edge of the $\overline{RAS}$ (*R*ow *A*ddress *S*trobe) control input; the column address is latched by the falling edge of the $\overline{CAS}$ (*C*olumn *A*ddress *S*trobe) control input. This operation is illustrated in Figure 3.

Dynamic RAMS may be visuallized as a two-dimensional array of single-bit storage cells arranged across the surface of the RAM's die. In the case of the 2164A, this array would consist of 2^8 (or 256) rows and 2^8 (or 256) columns, for a total of 2^{16} (or 65,526) total bit cells (Figure 4). This is the source of the "row address" and "column address" terminology. Bear in mind that any given RAM may not be physically implemented as described here; for instance, the 2164A actually contains four arrays, each one 2^7 rows by 2^7 columns.

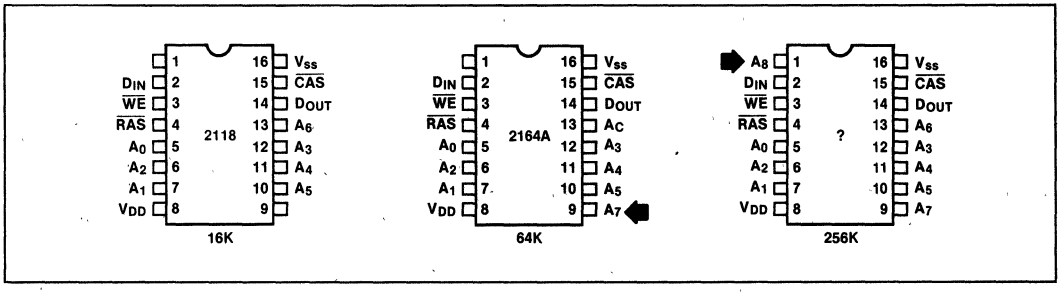


Figure 2. Dynamic RAM Pinout Compatibility

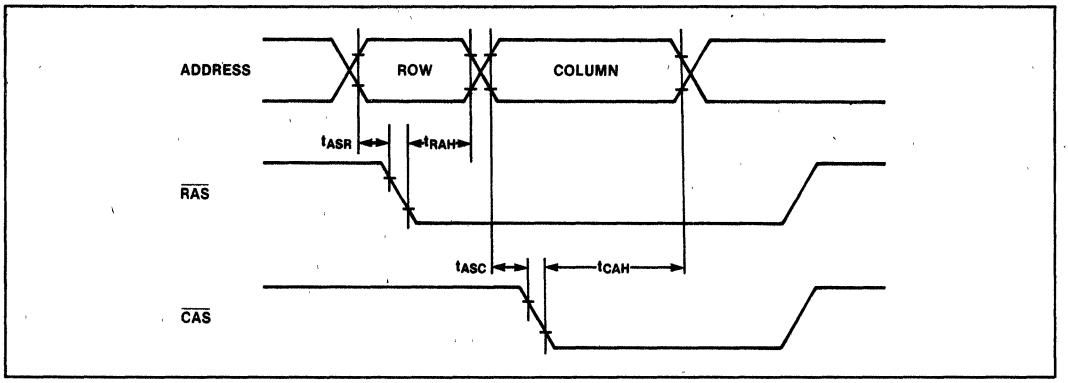


Figure 3. Dynamic RAM Addressing

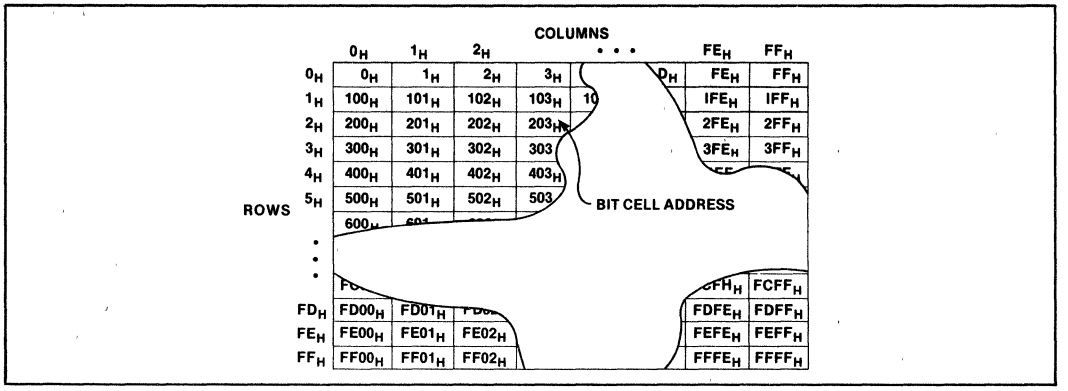


Figure 4. Bit Cell "Array"

Memory Cycles

In this Application Note, we will discuss three types of memory cycles — read, write, and RAS-only refresh. Dymanic RAMs may perform other types of cycles as well; these are described in the dynamic RAM's data sheet.

Whether data is read or written during a memory cycle is determined by the RAM's $\overline{WE}$ control input. Data is written only when $\overline{WE}$ is active.

During a read cycle, the $\overline{CAS}$ input has a second function, other than latching the column address. $\overline{CAS}$ also enables the RAM data output (pin 14) when active, assuming $\overline{RAS}$ is also active. Otherwise, the data output is 3-stated. This allows multiple dynamic RAMs to have their data outputs tied in common.

During write cycles, data on the RAM data input pin is latched internally to the RAM by the falling edge of $\overline{CAS}$ or $\overline{WE}$, *whichever occurs last*. If $\overline{WE}$ goes active before $\overline{CAS}$ (the usual case, called an "early write"), write data is latched by the falling edge of $\overline{CAS}$. If $\overline{WE}$ goes active after $\overline{CAS}$ (called a "late write"), data is latched by the falling edge of $\overline{WE}$ (see Figure 5).

Late writes are useful in some systems where it is desired to start the memory cycle as quickly as possible, to maximize performance, but the CPU cannot get the write data to the dynamic RAMs quickly enough to be latched by $\overline{CAS}$. By delaying $\overline{WE}$, more time is allowed for write data to arrive at the dynamic RAMs.

Note that when "late write" is performed, $\overline{CAS}$ goes active while $\overline{WE}$ is still inactive; this indicates a read cycle, so the RAM enables its data output. So, if "late write" cycles are performed by a system, the RAM data inputs and data outputs must be electically isolated from each other to prevent contention. If no "late writes" are performed, the RAM data inputs and data outputs may be tied together at the RAM to reduce the number of board traces.

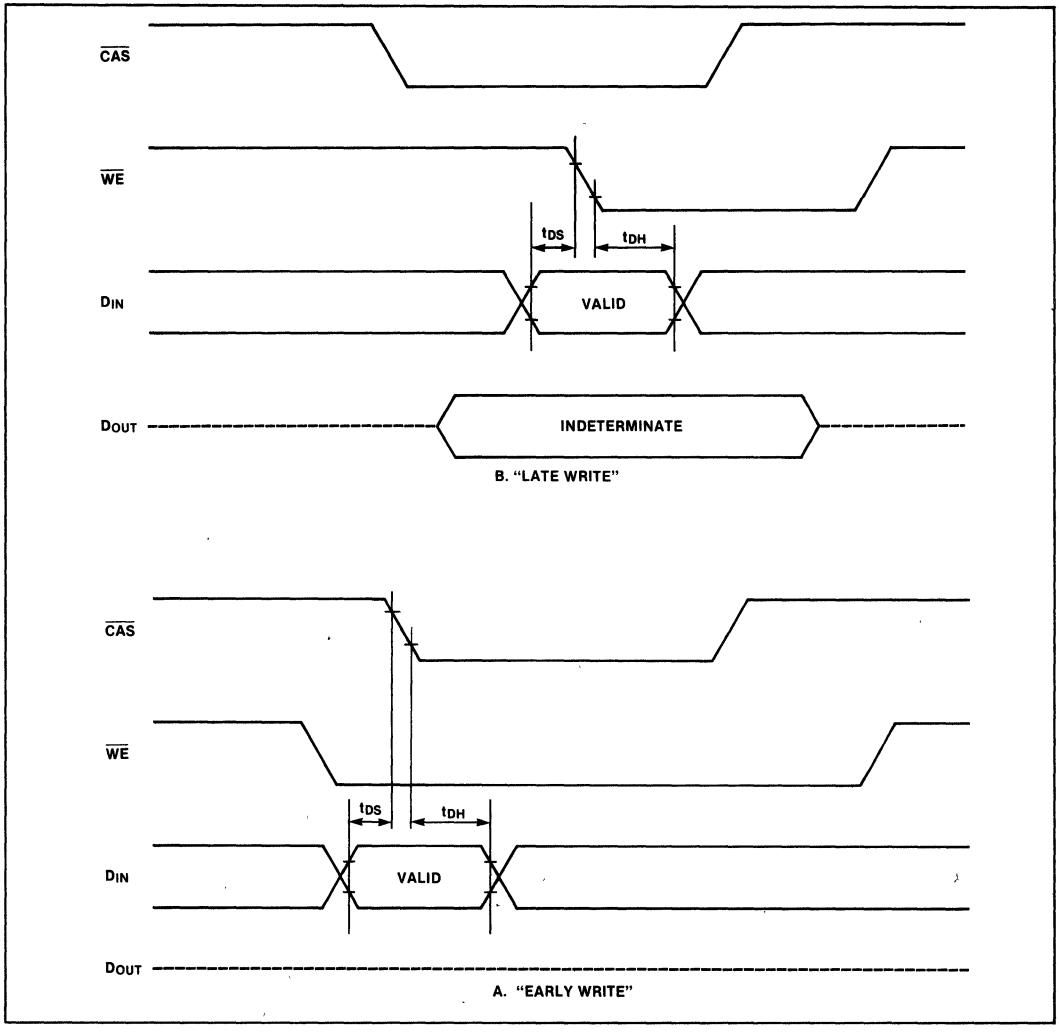


Figure 5. Dynamic RAM Write Cycles

Access Times

Each dynamic RAM has two different access times quoted for it — access time from $\overline{RAS}$ active (t_{RAC}) and access time from $\overline{CAS}$ active (t_{CAC}); these are illustrated in Figure 6. How do you know which to use? This depends on the timings of your RAM controller. First, the worst case delay from the memory read command active to $\overline{RAS}$ active (t_{CR}) and $\overline{CAS}$ active (t_{CC}) must be determined. Then the read data access time is the larger of the t_{CR}(Controller) + t_{RAC}(RAM) or t_{CC}(Controller) + t_{CAC}(RAM). An alternative way to determine whether to use t_{RAC} or t_{CAC} is to look at the dynamic RAM parameter for $\overline{RAS}$ active to $\overline{CAS}$ active delay, t_{RCD}. t_{RCD}*max* is a calculated value, and is shown on dynamic RAM data sheets as a reference point only. If the delay from $\overline{RAS}$ to $\overline{CAS}$ is less than or equal to t_{RCD}max, then t_{RAC} is the limiting access time parameter; if, on the other hand, the delay from $\overline{RAS}$ to $\overline{CAS}$ is greater than t_{RCD}max, then t_{CAC} is the limiting parameter. t_{RCD}max is not an operating limit, and this spec may be exceeded without affecting operation of the RAM. t_{RCD}*min*, on the other hand, is an operating limit, and the RAM will not operate properly if this spec is violated.

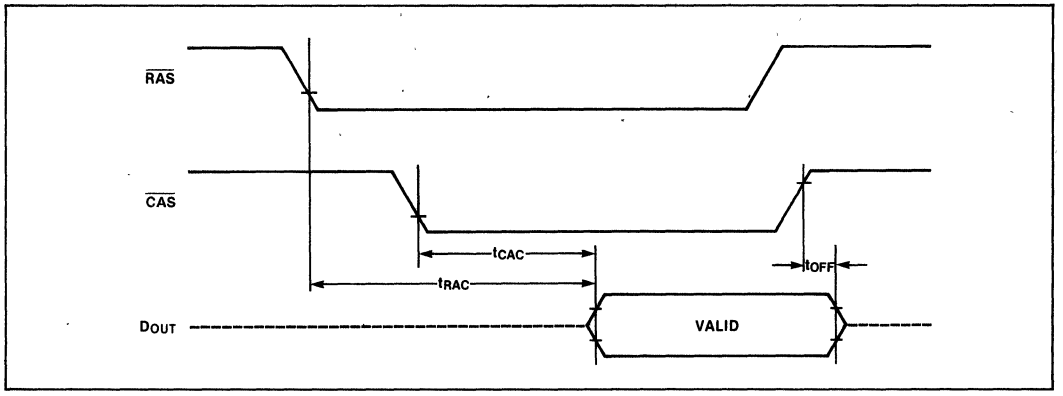


Figure 6. Dynamic RAM Access Times

Refresh

One unique requirement of dynamic RAMs is that they be *refreshed* in order to retain data. To see why this is so, we must look briefly at how a dynamic RAM is implemented.

Dynamic RAMs achieve their high density and low cost mostly because of the very simple bit-storage cell they use, which consists only of one transistor and a capacitor. The capacitor stores one bit as the presence (or absence) of charge. This capacitor is selectively accessed for reading and writing by enabling its associated transistor (see Figure 7).

Unfortunately, if left for very long, the charge will leak out of the capacitor, and the data will be lost. To prevent this, each bit-cell must be periodically read, the charge on the capacitor amplified, and the capacitor recharged to its initial state. The circuitry which does this amplification of charge is called a "sense amp". This must be done for every bit-cell every 2 ms or less to prevent loss of data.

Each column in a dynamic RAM has its own sense amp, so refresh can be performed on an entire row at a time. Thus, for the 2118, it is only necessary to refresh each of its 128 rows every 2 ms. Each row must be addressed via the RAM's address inputs to be refreshed. To simplify

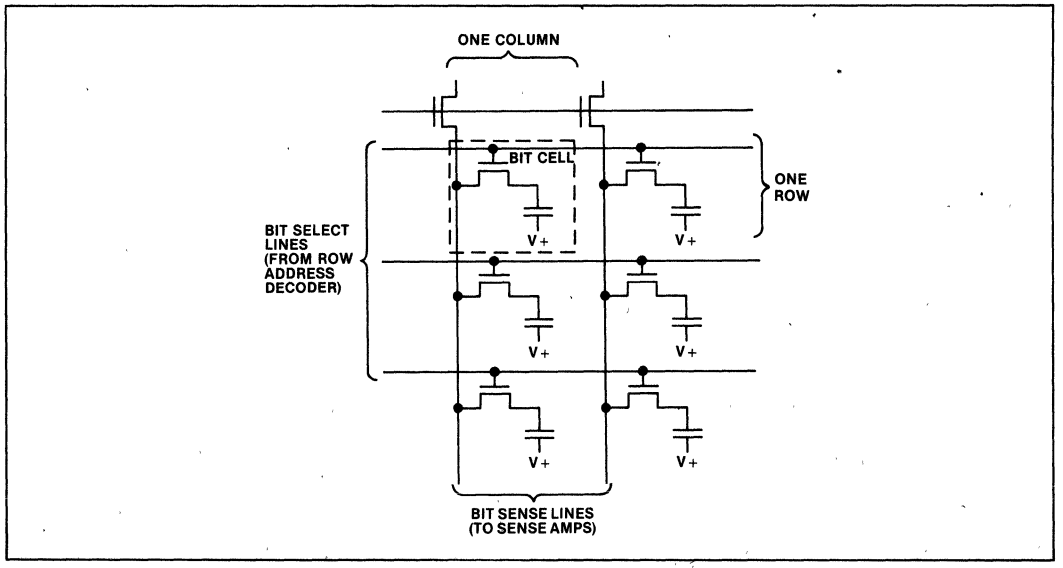


Figure 7. Dynamic RAM Cell

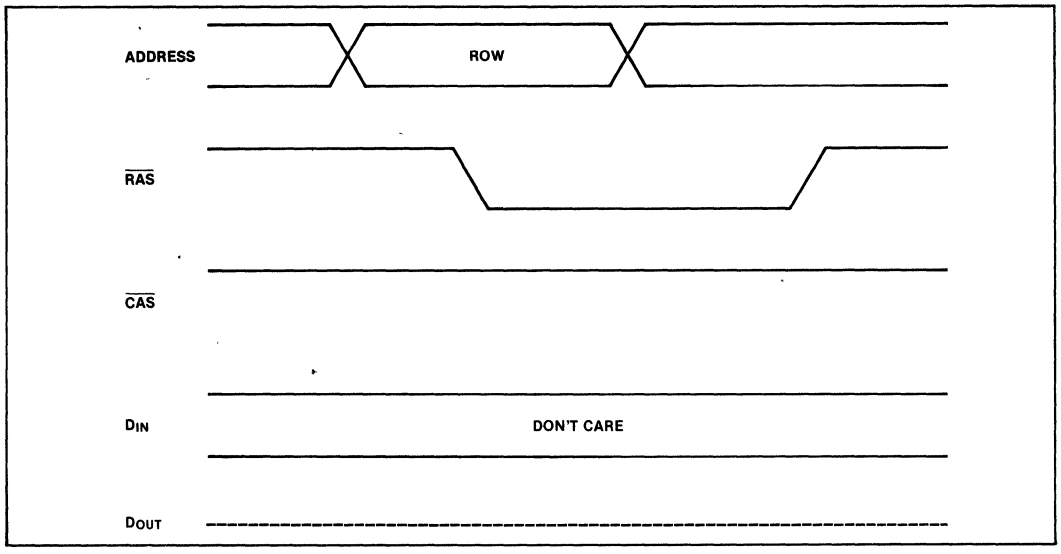


Figure 8. RAS-only Refresh

refresh, the 2164A is implemented in such a way that its refresh requirements are identical to the 2118; 128 rows every 2 ms. Some other 64K RAMs require 256 row refresh every 4 ms.

Refresh can be performed by a special cycle called a *RAS-only refresh*, shown in Figure 8. Only a row address is sent; that row is refreshed. No column address is sent, and no data is read or written during this cycle. Intel dynamic RAM controllers use this technique.

Any read, write, or read-modify-write cycle also refreshes the row addressed. This fact may be used to refresh the dynamic RAM without doing any special refresh cycles. Unfortunately, in general you cannot be sure that every row of every dynamic RAM in a system will be read from or written to every 2 ms, so refresh cannot be guaranteed by this method alone, except in special applications.

A third technique for refresh is called *hidden refresh*. This method is not popular in microprocessor systems, so it is not described here, but more information is available in the dynamic RAM's data sheet.

Three techniques for timing when refresh cycles are performed are in common use: burst refresh, distributed refresh, and transparent refresh.

Burst refresh means waiting almost 2 ms from the last time refresh was performed, then refreshing the entire memory with a "burst" of 128 refresh cycles. This method has the inherent disadvantage that during the time refresh is being performed (more than 40 microseconds for 128 rows) no read or write cycles can be performed. This severely limits the worst case response time to interrupts and makes this approach unsuitable for many systems.

As long as every row of the RAM is refreshed every 2 ms, the distribution of individual refresh cycles is unimportant. *Distributed refresh* takes advantage of this fact by performing a single refresh cycle every 2 ms/128, or about every 15 microseconds. In this way, the refresh requirements of the RAM are satisfied, but the longest time that read and write cycles are delayed because of refresh is minimized. Those few dynamic RAMs which use 256 row refresh allow 4 ms for the refresh to be completed, so the distributed refresh period is still 15 microseconds.

The third technique is called *transparent* (or "hidden" or "syncronous") *refresh*. This takes advantage of the fact that many microprocessors wait a fixed length of time after fetching the first opcode of an instruction to decode it. This time is necessary to determine what to do next (i.e. fetch more opcode bytes, fetch operands, operate on internal registers, etc.); this time may be longer than the time required for a RAM refresh cycle. If the status outputs of the CPU can be examined to determine which memory cycles are opcode fetches, a refresh cycle may be performed immediately afterward (Figure 9). In this way, refresh cycles will never interfere with read or write cycles, and so appear "transparent" to the microprocessor.

Transparent refresh has the disadvantage that if the microprocessor ever stops fetching opcodes for very

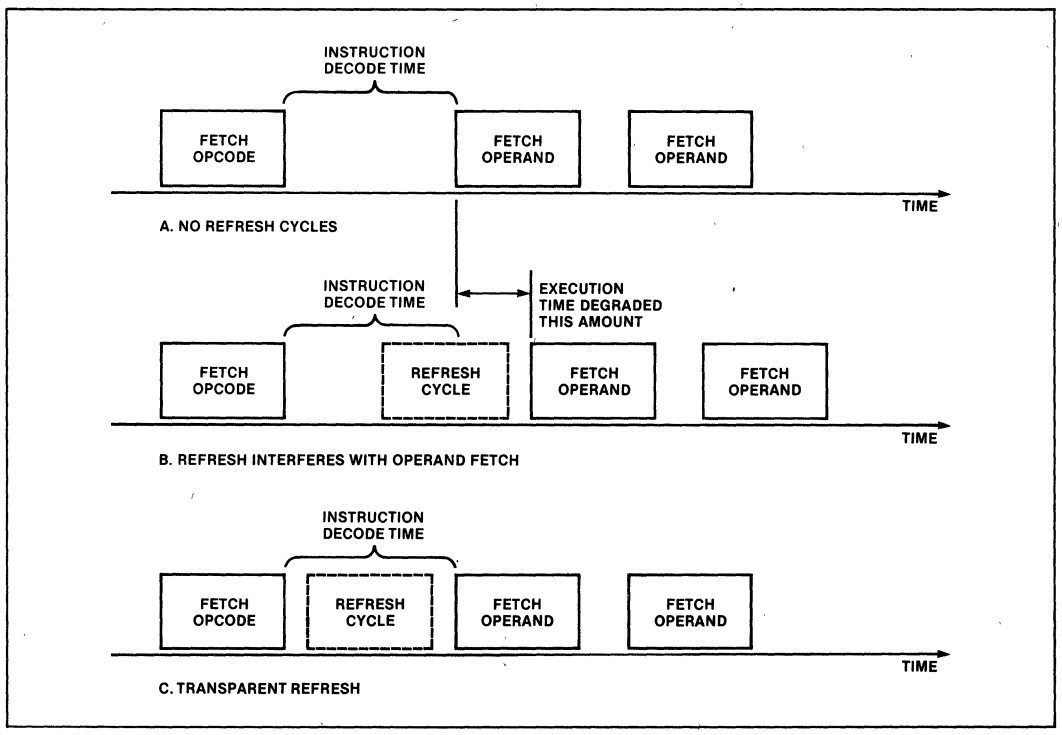


Figure 9. Transparent Refresh

long, due to a HOLD, extended DMA transfers, or when under hardware emulation, no refresh cycles will occur and RAM data will be lost. This puts restrictions on the system design. Also, high speed microprocessors do not allow sufficient time between opcode fetches and susequent bus cycles for a complete RAM refresh cycle to be performed, so they must wait for the refresh cycle to complete before they can do a subsequent bus cycle. These microprocessors cannot use transparent refresh to any advantage. Transparent refresh is useful for microprocessors like the Intel 8085 operating at low clock frequencies.

The 8086 and 8088, however, prefetch opcodes into a queue which is several bytes long. This prefetching is independent of the actual decoding and execution of the opcodes, and there is no time at which it can be guaranteed that the 8086 or 8088 will not request a memory cycle. So transparent refresh is not applicable to these microprocessors.

The 8202A and 8203 perform distributed and/or transparent refresh. Each device has an internal timer which automatically generates a distributed refresh cycle every 15.6 microseconds or less. In addition, an external refresh request input (REFRQ) allows the microprocessor's status to be decoded to generate a refresh cycle for transparent refresh. If, for whatever reason, no external REFRQ is generated for 15 microseconds, the internally generated refresh will take over, so memory integrity will be guaranteed.

Arbitration

Because RAMs cannot do a read or write cycle and a refresh cycle at the same time, some form of *arbitration* must be provided to determine when refresh cycles will be performed.

Arbitration may be done by the microprocessor or by the dynamic RAM controller. Microprocessor arbitration may be implemented as follows:

A counter, running from the microprocessor's clock, is used to time the period between refresh cycles. At terminal count, the arbitration logic asserts the bus request signal to prevent the microprocessor from performing any more memory cycles. When the microprocessor responds with a bus grant, the arbitration logic generates a refresh cycle (or cycles, if burst refresh is

used). After refresh is complete, the arbitration logic releases the bus. This method has several disadvantages: First, time is wasted in exchanging bus control, which would not be required if the RAM controller did arbitration. Second, while refresh is being performed, *all* bus activity is stopped; for instance, even if the microprocessor is executing out of ROM at the time, it must stop until refresh is over. Third, bursts of DMA transfers must be kept very short, as refresh cannot be performed while DMA is in progress.

Some microprocessors, such as the Zilog Z-80, generate refresh cycles themselves after instruction fetches. This removes the need for external arbitration logic, but still has several disadvantages: First, DMA bursts still must be kept short to allow the CPU to do refresh. Second, this method adds to the complexity of the microprocessor, without removing the need for the RAM controller which is still required to do address multiplexing and $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ timing. Microprocessor refresh can cause problems of RAM compatibility; for instance, the Z-80 only outputs a 7-bit refresh address, which means some 64K RAMs which use 256 row refresh cannot be used with the Z-80. Also, since the Z-80 refresh cycle is a fixed length (no wait states), faster speed selections of the Z-80 are not compatible with slower dynamic RAMs. Third, systems employing multiprocessing or DMA are harder to implement, because of the difficulty in insuring the microprocessor will be able to perform refresh.

It is preferable to have arbitration performed by the dynamic RAM controller itself. This method avoids all the problems described above, but introduces a complication. If the microprocessor issues a read or write command while the dynamic RAM is in the middle of a refresh cycle, the RAM controller must make the microprocessor wait until it is done with the refresh before it can complete the read or write cycle. This means that from when the microprocessor activates the read or write signal, the time until the cycle can be completed can vary over a range of roughly 200 to 700 ns. Because of this, an acknowledge signal from the dynamic RAM controller is required to tell the microprocessor the memory cycle it requested is complete. This signal goes to the microprocessor's READY logic.

Memory Organization

As each dynamic RAM operates on only one bit at a time, multiple RAMs must be operated in parallel to operate on a word at a time. RAMs operated in this way are called a *bank* of RAM. A bank consists of as many RAMs as there are bits in the memory word. When used in this way, all address and control lines are tied to all RAMs in the bank.

A single bank of RAM will provide 64K words of memory in the case of the 2164A, or 16K words in the case of the 2118. To provide more memory words, multiple banks of RAM are used. In this case, all address, CAS, and $\overline{WE}$ lines are tied to all RAMs, but each bank of RAM has *its own* $\overline{RAS}$. Each bank knows whether it is being addressed during a read or write operation by whether or not its $\overline{RAS}$ input was activated — if not, then all other inputs are ignored during that cycle.

Data outputs for RAMs in corresponding bit positions in each of the banks may be tied in common, since they are 3-state outputs; even though $\overline{CAS}$ is connected to all banks of RAM, only that bank whose $\overline{RAS}$ is active will enable its data outputs in response to $\overline{CAS}$ going active. Data inputs for RAMs in corresponding bit positions in each of the banks are also tied in common.

INTEL DYNAMIC RAM CONTROLLERS

The Intel 8202A and 8203 Dynamic RAM Controllers each provide all the interface logic needed to use dynamic RAMs in microprocessor systems, in a single chip. Either the 8202A or 8203 allow a dynamic RAM memory to be implemented using a minium of components, board space, and power, and in less design time than any other approach.

The following sections will describe each of these controllers in detail.

8202A

FUNCTIONAL DESCRIPTION

The 8202A provides total dynamic RAM control for 4K and 16K dynamic RAMs, including the Intel 2104A, 2117, and 2118. The pinout and simplified logic diagram of the 8202A are shown in Figures 10 and 11.

The 8202A is always in one of the following states:

a) IDLE
b) TEST cycle
c) REFRESH cycle
d) READ cycle
e) WRITE cycle

The 8202A is normally in the idle state. Whenever a cycle is requested, the 8202A will leave the idle state to perform the desired cycle; if no cycle requests are pending, the 8202A will return to the idle state. A refresh cycle request may originate internally or externally to

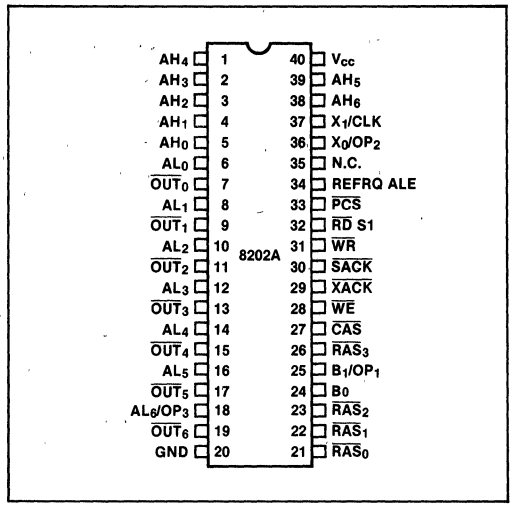


Figure 10. 8202A Pinout

the 8202A; all other requests come only from outside the 8202A.

A test cycle is requested by activating the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$ (Protected Chip Select). The test cycle will reset the refresh address counter to zero and perform a write cycle. A test cycle should not be allowed to occur in normal system operation, as it interferes with normal RAM refresh.

A refresh cycle performs a $\overline{RAS}$-only refresh cycle of the next lower consecutive row address after the one previously refreshed. A refresh cycle may be requested by activating the REFRQ input to the 8202A; this input is latched on the next 8202A clock. If no refresh cycles are requested for a period of about 13 microseconds, the 8202A will generate one internally. By refreshing one row every 15.6 microseconds or sooner, all 128 rows will be refreshed every 2 ms. Because refresh requests are generated by the 8202A itself, memory integrity is insured, even if the rest of the system should halt operation for an extended period of time.

The arbiter logic will allow the refresh cycle to take place only if there is not another cycle in progress at the time.

A read cycle may be requested by activating the $\overline{RD}$ input, with $\overline{PCS}$ (Protected Chip Select) active. In the Advanced Read mode, a read cycle is requested if the microprocessor's S1 status line is high at the falling edge of ALE (Address Latch Enable) and $\overline{PCS}$ is active. If a dynamic RAM cycle is terminated prematurely, data loss may result. The 8202A chip select is "protected" in that once a memory cycle is started, it will go to completion, even if the 8202A becomes de-selected.

A write cycle may be requested by activating the $\overline{WR}$ input, with $\overline{PCS}$ active; this is the same for the normal and Advanced Read modes.

BLOCK DIAGRAM

Let's look at the detailed block diagram in Figure 12 to see how the 8202A satisfies the interface requirements of the dynamic RAM.

Address Multiplexing

Address multiplexing is achieved by a 3-to-1 multiplexer

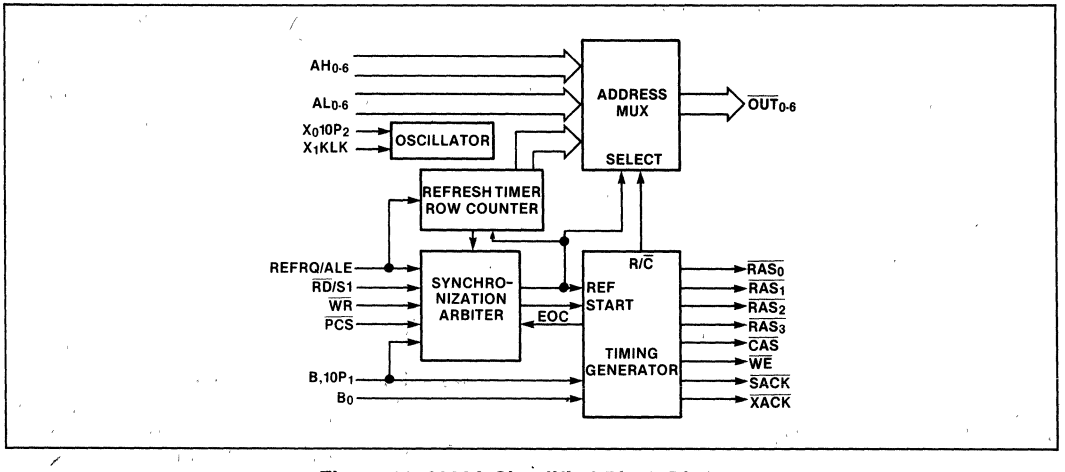


Figure 11. 8202A Simplified Block Diagram

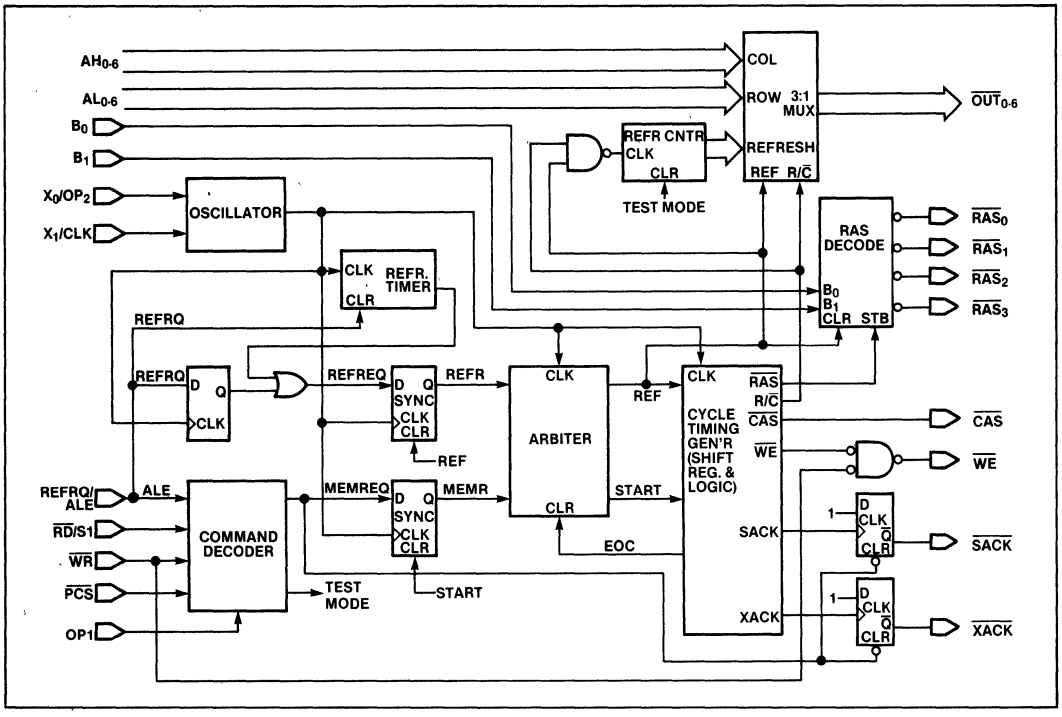


Figure 12. 8202A Detailed Block Diagram

internal to the 8202A; the three inputs are the row address (AL_{0-6}), column address (AH_{0-6}), and refresh row address (generated internally). When the 8202A is in the Idle state, the multiplexer selects the row address, so it is prepared to start a memory cycle. If a refresh cycle is requested either internally or externally, the address multiplexer will select the refresh row address long enough before $\overline{RAS}$ goes active to satisfy the RAM's t_{ASR} parameter.

To minimize propagation delays, the 8202A address outputs ($\overline{OUT}_{0-6}$) are inverted from the address inputs. This has no effect on RAM operation; inverters are not needed on the address outputs.

Doing this multiplexing internally minimizes timing skews between the address, $\overline{RAS}$, and $\overline{CAS}$, and allows higher performance than would otherwise be possible.

Refresh Counter

The next row to be refreshed is determined by the refresh counter, which is implemented as a 7-bit ripple-carry counter. During each refresh cycle, the counter is

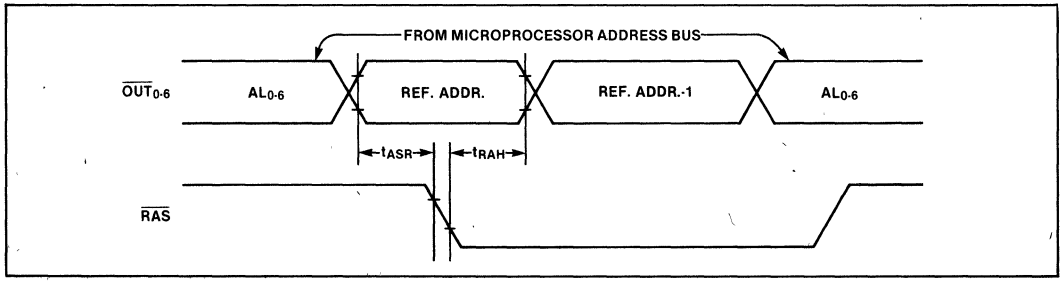


Figure 13. Detailed 8202A Refresh Cycle

incremented by one in preparation for the next refresh cycle (a refresh cycle is shown in detail in Figure 13).

When the 8202A enters TEST mode, the refresh counter is cleared. This feature is useful for automatic testing of the refresh counter function. Because the address outputs are inverted, the first refresh address after clearing the counter in test mode is $7F_H$, and the addresses *decrease* for subsequent refresh cycles.

$\overline{RAS}$ Decoding

Which bank of RAM is selected for a memory cycle is determined by the $\overline{RAS}$ decoder from the B_{0-1} inputs, which normally come from the microprocessor address bus. The 8202A Timing Generator produces an internal $\overline{RAS}$ pulse which strobes the $\overline{RAS}$ decoder, generating the appropriate external $\overline{RAS}$ pulse. The B_{0-1} inputs are *not* latched, so they must be held valid for the length of the memory cycle. During a refresh cycle, all the $\overline{RAS}$ outputs are activated, refreshing all banks at once.

Oscillator

The 8202A operates from a single reference clock with a frequency between 18.432 MHz and 25 MHz; this clock is used by the synchronization, arbitration, and timing generation logic. This clock may be generated by an on-board crystal oscillator, or by an external TTL-compatible clock source. When using the internal oscillator (available only on part number D8202A-1 or D8202A-3), a fundamental-mode crystal is attached to pins 36 and 37 (X_0 and X_1), as shown in Figure 14. The external TTL clock option is selected by pulling pin 36 (OP_2) to +12v through 1K ohm resistor, and attaching the clock input to pin 37 (CLK).

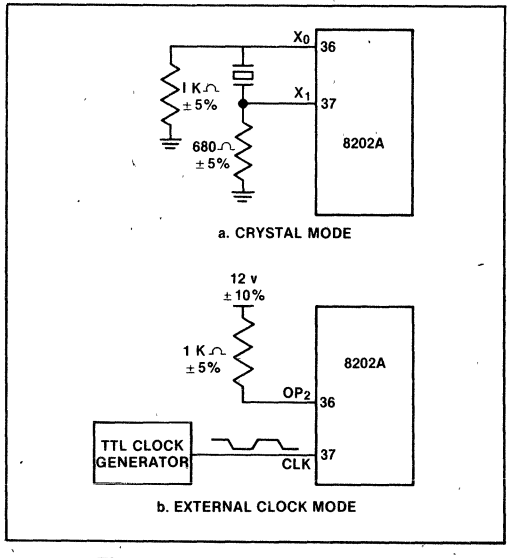


Figure 14. 8202A Clock Options

Command Decoder

The command decoder takes the commands from the bus and generates internal memory request (MEMR), and TEST signals.

The 8202A has two bus interface modes: the "normal" mode, and the "Advanced Read" mode. In the normal mode, the 8202A interfaces to the usual bus RD and WR signals.

In the Advanced Read mode, the 8202A interfaces to the Intel microprocessor bus signals ALE, S1, and $\overline{WR}$. S1 must be high on the falling edge of ALE for read cycles, and $\overline{WR}$ must be low for write cycles (write cycles are the same as for normal read mode). The 8085A S1 may be used directly by the 8202A; the 8086 and 8088 $\overline{S1}$ must be inverted. ALE and $\overline{WR}$ must be qualified by $\overline{PCS}$.

The Advanced Read mode is useful for reducing read data access time, and thus wait states. This mode is used mainly with 8085A systems.

If both $\overline{RD}$ and $\overline{WR}$ are active at once (regardless of the state of $\overline{PCS}$), the internal TEST signal is generated and the 8202A performs a test cycle as described above. One or both of $\overline{RD}$ and $\overline{WR}$ should have pull-up resistors to prevent the 8202A from inadvertantly being put into test mode, as the $\overline{RD}$ and $\overline{WR}$ signals are 3-stated by the microprocessor when RESET or HOLD are active. Since the test mode resets the refresh address counter, the refresh sequence will be interrupted, and data loss may result.

Refresh Timer and REFRQ

The 8202A contains a counter, operated from the internal clock to time the period from the last refresh cycle. When the counter times out, an internal refresh request is generated. This refresh period is proportional to the 8202A's clock period, and varies from 10.56 to 15.625 microseconds. Even at the lowest refresh rate, all the rows of the dynamic RAM will be refreshed every 2 ms.

The 8202A has an option of reducing the refresh rate by a factor of two, for use with 4K RAMS. These RAMs have only 64 rows to refresh every 2 ms, so need refresh cycles only half as often. This option is selected by pull-

ing pin 18 (AL_6/OP_3) to +12v through a 5.1K ohm resistor. This pin normally serves as the high-order row address input for the address multiplexer, but it is no longer needed for this function, as 4K RAMs have one less address input.

A refresh cycle may also be requested externally by activating the REFRQ input. This input is latched, so it only needs to be held active a maximum of 20 ns. If the 8202A is currently executing a memory cycle, it will complete that cycle, and then perform the refresh cycle. The internal and external refresh requests are ORed together before going to the arbiter.

The REFRQ input cannot be used in the Advanced Read mode, as the REFRQ pin is used for ALE in this mode.

REFRQ is most often used to implement transparent refresh, as explained in the section *Dynamic RAMS — Refresh*. This technique is not useful in iAPX 86 and iAPX 88 systems, so REFRQ is normally tied to ground.

The refresh timer is reset as soon as a refresh cycle is started (whether it was requested internally or externally). The time between refresh cycle (t_{REF}) is measured from when the first cycle is *started*, not when it was *requested*, which occurs sometime earlier. Of course, t_{REF}min does not apply if REFRQ is used — you may externally request refresh cycles as often as you wish.

Arbiter

This is the hardest section of a dynamic RAM controller to implement. If a read or write arrives at the same time as a refresh request, the arbiter must decide which one to service first. Also, if a read, write, or refresh request arrives when another cycle is already in progress, the arbiter must delay starting the new cycle until the current cycle is complete.

Both of the internal signals REFR (refresh request) and MEMR (memory cycle request) are synchronized by D-type master-slave flip-flops before reaching the arbiter. these circuits have been optimized to resolve a valid logic state in as short a time as possible. Of course, with any synchronizer, there is a probability that it will fail — not be able to settle in one logic state or the other in the allowed amount of time, resulting in a memory failure — but the 8202A has been designed to have less than one system memory failure every three years, based on operation in the worst case system timing environments.

Both synchronizers and the arbiter are operated from the 8202A's internal clock. Assuming the 8202A is initially in an idle state, one full clock period after the synchronizers sample the state of the MEMREQ and REFREQ signals, the arbiter examines the REFR and MEMR outputs of the synchronizers. If MEMR is active, the arbiter will activate START to begin the memory cycle (either read or write) on that clock. If REFR is active (regardless of the state of MEMR), the arbiter will activate START and REF to begin a refresh cycle on that clock. Once the cycle is complete, the Cycle Timing Generator will generate an end-of-cycle (EOC) signal to clear the arbiter and allow it to respond to any new or pending requests on the next clock.

Once a memory cycle is started, it cannot be stopped, regardless of the state of the $\overline{RD}$/S1, $\overline{WR}$, $\overline{ALE}$, or $\overline{PCS}$ inputs. This is necessary, as ending a dynamic RAM cycle prematurely may cause loss of data. Note, however, that the RAM $\overline{WE}$ output is directly gated by the $\overline{WR}$ input, so if $\overline{WR}$ is removed prematurely, the RAM $\overline{WE}$ pulse-width spec (t_{WP}) may be violated, causing a memory failure.

What happens if a memory request and refresh request occur simultaneously?

If the 8202A is in the idle state, the *memory* request will be honored first.

If the 8202A is *not* in the idle state (a memory or refresh cycle is in progress) then the memory cycle will lose priority and the refresh cycle will be honored first.

Remember, if the 8202A is performing a cycle, the arbiter doesn't arbitrate again until the end of that cycle. So the memory and refresh cycles are "simultaneous" if they both happen early enough to reach the arbiter before it finishes the current cycle. This arbitration arrangement gives memory cycles priority over refresh cycles, but insures that a refresh cycle will be delayed at most one RAM cycle.

Refresh Lock-Out

As a result of the 8202A operation, transparent refresh circuits like the one shown in Figure 15 should not be used. This circuit uses the RD input, with some qualifying logic, to activate REFRQ whenever the microprocessor does an opcode fetch. This circuit will work fine, as long as the 8202A never has to generate an internal refresh request, which is unlikely (if nothing else, the system RESET pulse is probably long enough that the 8202A will throw in a couple of refreshes while the microprocessor is reset). If the 8202A ever does generate its own refresh, there is a probability that the microprocessor will try to fetch an opcode while the

refresh is still in progress. If that happens, the 8202A will finish the refresh, see both the RD and REFRQ inputs active, honor the REFRQ first, and start a *second* refresh. In the meantime, the microprocessor is sitting in wait states, waiting for the 8202A to complete the opcode fetch. When the 8202A finishes the second refresh, it will see both RD and REFRQ active again, and will start a *third* refresh, etc. The system "locks up" with the microprocessor sitting in wait states *ad infinitum*, and the 8202A doing one refresh cycle after another.

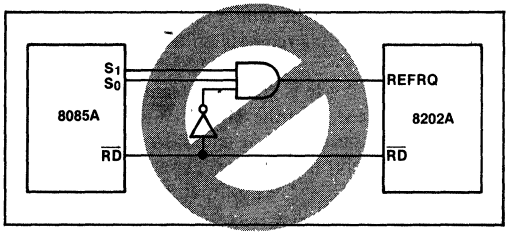


Figure 15. Improper Transparent Refresh Generation

To prevent this from happening, the transparent refresh circuit should be modified as shown in Figure 16. In this circuit, REFRQ cannot be activated until the opcode fetch is already in progress, as indicated by $\overline{SACK}$ being active (remember, $\overline{SACK}$ is never active during a refresh). If the microprocessor tries to do an opcode fetch while the 8202A is doing a refresh, REFRQ will not be active; the 8202A will finish the refresh and see only $\overline{RD}$ active, and will start the opcode fetch; only *then* will REFRQ be activated.

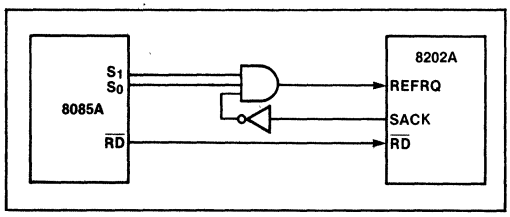


Figure 16. Generating Transparent Refresh For 8085A Systems

Cycle Timing Generator

The Cycle Timing Generator consists of a travelling-ones shift register and combinational logic required to generate all the RAM control signals and $\overline{SACK}$ and $\overline{XACK}$. All timings are generated from the 8202A's internal clock; no external delay lines are ever needed. The timing of these signals relative to CLK is illustrated in Figure 17.

When the cycle is complete, the Cycle Timing Generator sends an end-of-cycle (EOC) pulse to the arbiter to enable it to respond to new or pending cycle requests.

Minimum and maximum values for the 8202A parameters t_{CR} (Command to $\overline{RAS}$ active delay) and t_{CC} (Command to $\overline{CAS}$ active delay) differ by one 8202A clock period. This is because the commands ($\overline{RD}$, $\overline{WR}$, ALE) must be synchronized to the 8202A's clock; this introduces a ± one clock period (t_p) uncertainty due to the fact that the command may or may not be sampled on the first clock after it goes active, depending on the set-up time. If $\overline{RD}$ or ALE and $\overline{WR}$ are synchronous to the 8202A's clock, and the set-up time (t_{SC}) is met, the smaller number of clock periods will apply.

All 8202A output timings are specified for the capacitive loading in the data sheet. Typical output characteristics are shown in the data sheet for capacitive loads ranging from 0 to 660 pF, these can be used to calculate the effect of different loads than those specified in the data sheet on output timings. All address, $\overline{RAS}$, $\overline{CAS}$, and $\overline{WE}$ drivers are identical, so these characteristic curves apply to all outputs.

$\overline{SACK}$ AND $\overline{XACK}$

Because refresh cycles are performed asynchronously to the microprocessor's operation (except during transparent refresh), the microprocessor cannot know when it activates $\overline{RD}$ or $\overline{WR}$ if a refresh cycle is in progress, and therefore, it can't know how long it will take to complete the memory cycle.

This added consideration requires an acknowledge or "handshake" signal from the 8202A to tell the microprocessor when it may complete the memory cycle. This acknowledge would be used to generate the microprocessor's READY input — the microprocessor will sit in wait states until the 8202A acknowledges the memory cycle. Two signals are generated for this purpose by the 8202A; they are called *system acknowledge* ($\overline{SACK}$) and *transfer acknowledge* ($\overline{XACK}$). They serve the same purpose but differ in timing.

$\overline{XACK}$ is a Multibus-compatible signal, and is not activated until the read or write cycle has been completed by the RAMs. In a microprocessor system, however, there is a considerable delay from when the 8202A acknowledges the memory cycle until the microprocessor actually terminates the cycle. This delay is due to the time required to combine this acknowledge with other sources of READY in the system, synchronize READY to the microprocessor's clock, sample the state of READY, and respond to an active READY signal. As a result, more wait states than necessary may actual-

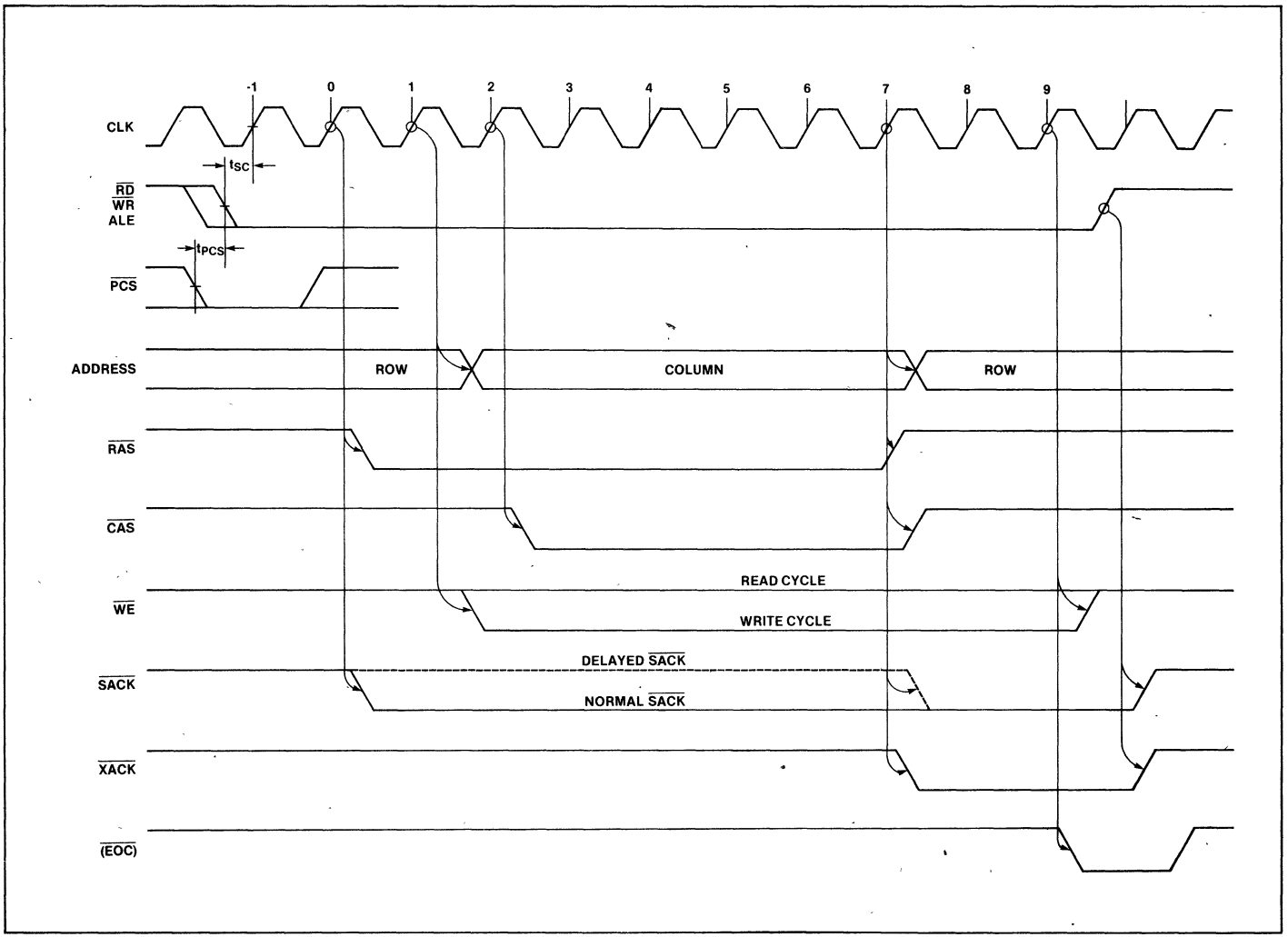


Figure 17. 8202A Timing Relative To CLK

AFN: 02200A

ly be generated by using $\overline{XACK}$. $\overline{SACK}$ is activated earlier in the cycle to improve performance of microprocessors by compensating for the delays in the microprocessor responding to $\overline{XACK}$, and thus eliminating unneeded wait states which might be generated as a result of $\overline{XACK}$ timing. The system designer may use one or the other acknowledge signal, or use both in different parts of the system, at his option.

$\overline{SACK}$ and $\overline{XACK}$ are activated by the Cycle Timing Generator, but they can be de-activated only by the microprocessor removing its $\overline{RD}$ or $\overline{WR}$ request, or by activating ALE when in the advanced read mode. As the $\overline{SACK}$ and $\overline{XACK}$ signals are used to generate READY for the microprocessor, this is necessary to give the microprocessor as much time as it needs to respond to its READY input.

Delayed SACK Mode

$\overline{SACK}$ may be activated at one of two different times in the memory cycle; the earlier case is called "normal $\overline{SACK}$" and the later is called "delayed $\overline{SACK}$" (Figure 18). Delayed $\overline{SACK}$ occurs if the memory request was received by the 8202A while it was doing a refresh cycle. In this case, the memory cycle will be delayed some length of time while the refresh cycle completes; $\overline{SACK}$ is delayed to ensure the microprocessor will generate enough wait states. This is a concern mostly for read cycles.

Because of the way the delayed $\overline{SACK}$ mode is implemented in the 8202A, if the $\overline{RD}$ or $\overline{WR}$ input is activated while a refresh cycle is in progress, regardless of whether or not the 8202A is chip-selected, the internal delayed $\overline{SACK}$ mode flip-flop will be set. The next 8202A memory cycle will have $\overline{SACK}$ delayed, even if that cycle was not actually delayed due to a refresh cycle in progress. The delayed SACK flip-flop will be reset at the end of that cycle, and the 8202A will return to normal $\overline{SACK}$ operation. The same thing happens in Advanced Read mode if S1 is high at the falling edge of ALE during a refresh cycle, once again regardless of the state of $\overline{PCS}$.

8203

The 8203 is an extension of the 8202A architecture which allows the use of 64K dynamic RAMs. It is pinout compatible with the 8202A and shares identical A.C. and D.C. parameters with that part. The description of the 8202A applies to this part also, with the modifications below.

ENHANCEMENTS

1. Supports 16K *or 64K* dynamic RAMs. 4K RAM mode, selected by pulling AL_6/OP_3 (pin 18) to +12v, is not supported.

2. Allows a single board design to use either 16K or 64K RAMs, without changing the controller, and only making between two and four jumper changes to reconfigure the board.

3. May operate from external TTL clock without the +12v pull-up which the 8202A requires (a +5v or +12v pull-up may be used).

The pinout of the 8203 is shown in Figure 19. This pinout is identical to the 8202A, with the exception of the five highlighted pins. The function of these is described below. The simplified block diagram is similar to the 8202A's, in Figure 11.

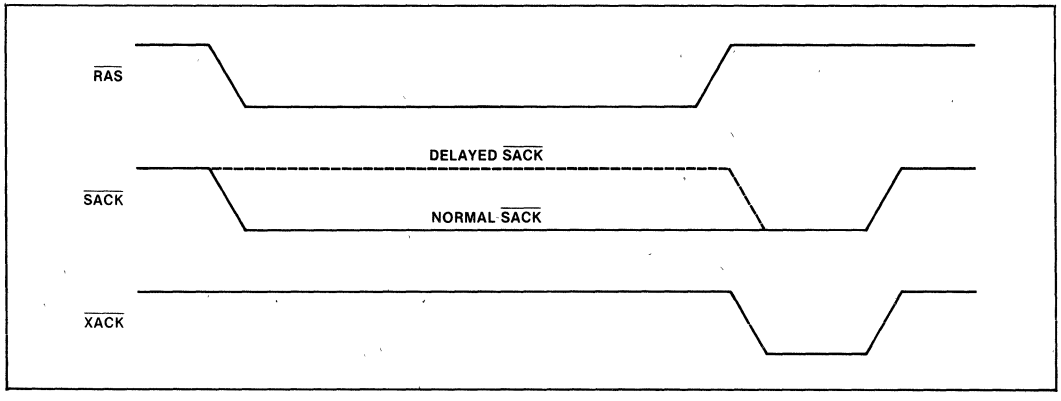


Figure 18. Delayed SACK Mode

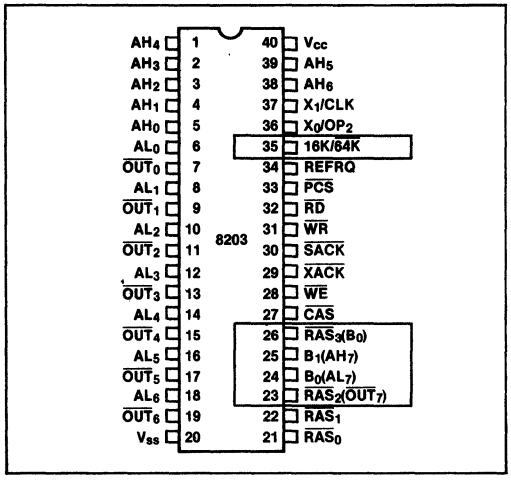


Fig. 19 8203 Pinout

16K Mode and 64K Mode

The goal of the 8203 is to provide a pin- and timing-compatible upgrade of the 8202A for use with 64K RAMs. The difficulty in doing this is that 64K RAMs require an additional address input compared to 16K RAMs, and thus the 8203 needs three more pins (one more RAM address output, and two more inputs to its internal address multiplexer). Since all but one of the 8202A's pins are already used, this is clearly a challenge — some functionality must be sacrificed to gain 64K RAM support. The 8203 reduces the maximum number of banks supported from four to two for 64K RAMs.

Pin 35 (16K/$\overline{64K}$) is used to tell the 8203 whether it is being used to control 16K RAMs or 64K RAMs. When tied to V_{CC} or left unconnected, the 8203 operates in the 16K RAM mode; in this mode all the remaining pins function identically to the 8202A. When tied to ground, it operates in the 64K RAM mode, and pins 23 through 26 change function to enable the 8203 to support 64K RAMs. Pin 35 (16K/$\overline{64K}$) contains an internal pull-up —when unconnected, this input is high, and the 8203 operates identically to the 8202A. This maintains pinout compatibility with the 8202A, in which pin 35 is a no-connect, so the 8203 may be used in 8202A sockets with no board modifications.

When the 8203 is in the 64K RAM mode, four pins change function, as shown in Table 2. The pins change function in this particular way to allow laying out a board to use either 16K or 64K RAMs with a minimum of jumpers, as shown in Figure 20. This figure shows the 8203 with two banks of RAM. Banks 0 and 1 may be either 16K RAMs or 64K RAMs; banks 2 and 3 may only be 16K RAMs, as the 8203 supports two banks of 64K RAM. For clarity, only those connections which are important in illustrating the 8203 jumper options are shown.

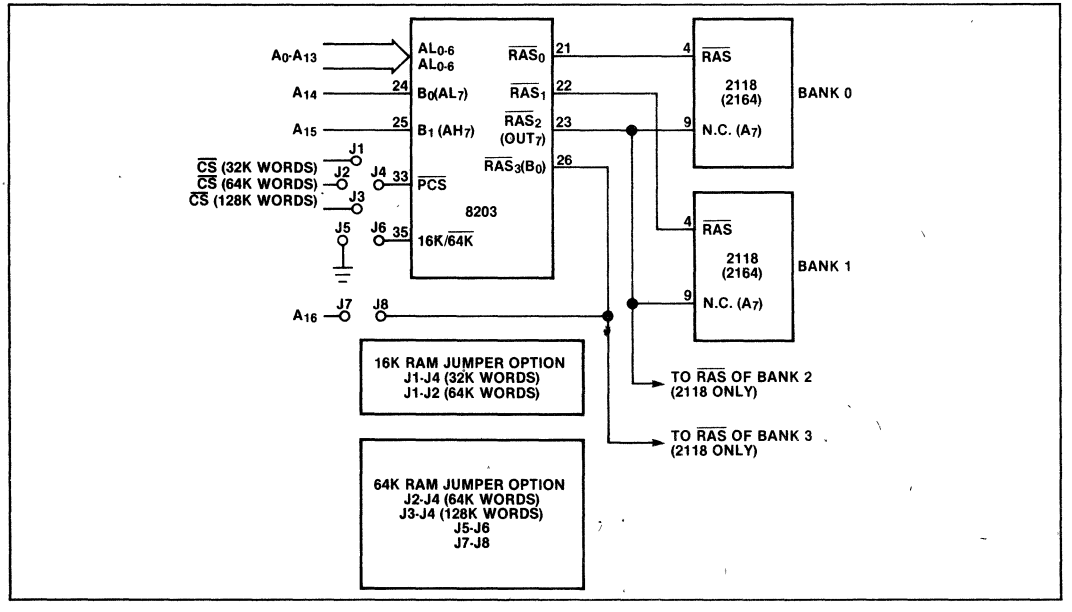


Figure 20. 8203 Jumper Options

Table 2. 16K/$\overline{64K}$ Mode Selection

Pin #	16K Function	64K Function
23	$\overline{RAS}_2$	Address Output (OUT_7)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS}_3$	Bank Select (B_0)

Jumpers J1-J4 may be used to chip select the 8203 over various address ranges. For example, if two banks of 16K RAMs are replaced with two banks of 64K RAMs, the address space controlled by the 8203 increases from 32K words to 128K words. If four banks of 16K RAMs are replaced with one bank of 64K RAMs, no chip select jumpers are needed.

In the 64K RAM mode, pins 24 and 25 ($B_0(AL_7)$ and $B_1(AH_7)$) change function from bank select inputs to address inputs for the 64K RAM. Since the bank select inputs normally come from the address bus anyway, no jumper changes are required here. The bank select function moves to pin 26 ($\overline{RAS}_{3(B_0)}$); since only two bank of 64K RAM is supported, only one bank select input is needed in this mode, not two. Jumpers J6 and J7 are shorted in the 64K RAM mode to connect pin 26 (B_0) to the address bus. In the 16K RAM mode, these jumpers must be disconnected, as pin 26 junctions as the $\overline{RAS}_3$ output; in the 64K RAM mode, this bank is not populated, so $\overline{RAS}_3$ is not needed.

Pin 23 serves two functions: in the 16K RAM mode it is the $\overline{RAS}$ output for bank 2 ($\overline{RAS}_2$), in the 64K RAM mode is the high order RAM address output ($\overline{OUT}_7$), which goes to pin 9 of the 64K RAMs. This requires no jumpers as when using 16K RAMs, pin 9 is a no-connect, and when using 64K RAMs, bank 2 is depopulated, so $\overline{RAS}_2$ is not used.

This arrangement allows converting a board from 16K RAMs to 64K RAMs with no change to the controller and changing a maximum of three jumpers.

+5v External Clock Option

Just as with the 8202A, the user has the option of an external TTL clock instead of the internal crystal oscillator as the timing reference for the 8203; unlike the 8202A, he does not need to tie pin 36 ($X_0/0P_2$) to +12v to select this option—this pin may be tied to *either* +5v *or* +12v. If pin 36 is tied to +12v, a 1K ohm (± 5%) series resistor must be used, just as for the 8202A. If pin 36 is tied to +5v, it must be tied *directly* to pin 40 (V_{cc}) with no series resistor. This is because pin 36 must be within one Schottky diode voltage drop (roughly 0.5v) of pin 40 to select the external TTL clock option; a series resistor may cause too great a voltage drop for the external clock option to be selected. For the same reason, the trace from pin 36 to 40 should be kept as short as practical.

Test Cycle

An 8203 test cycle is requested by activating the $\overline{RD}$, $\overline{WR}$, and $\overline{PCS}$ inputs simultaneously. By comparison, an 8202A test cycle requires activating only the $\overline{RD}$ and $\overline{WR}$ inputs simultaneously, independent of $\overline{PCS}$. Like the 8202A, and 8203 test cycle resets the address counter to zero and performs a write cycle.

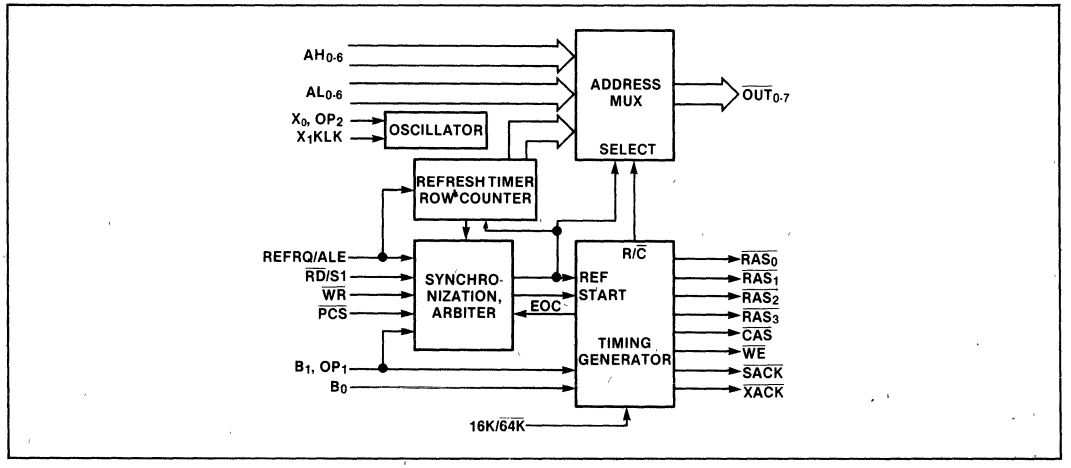


Figure 21. 8203 Simplified Block Diagram

BLOCK DIAGRAM

A simplified block diagram of the 8203 is shown in Figure 21. It is identical to the 8202A except for the following differences:

1. The 3:1 address multiplexer is 8 bits wide, instead of 7 bits wide, to support the addressing requirements of the 64K RAM.
2. The refresh address counter is 8 bits. This allows it to support RAMs which use either the 128-row or 256-row refresh schemes. Regardless of which type of RAM is used, the refresh counter cycles through 256 rows every 4 ms. RAMs which use 128-row re-fresh treat the eighth address bit as a "don't care" during refresh, so they see the equivalent of 128-row refresh every 2 ms. In either case the rate of internally-generated refresh cycles is the same—at least one every 15.6 microseconds.

INTEL iAPX-86 AND iAPX-88

Device Descriptions

The iAPX-86 and iAPX-88 are advanced 16-bit microprocessor families, based on the 8086 and 8088 microprocessors, respectively. While both have a similar architecture and are software compatible, the 8086 transfers data over a 16-bit bus, while the 8088 uses an 8-bit data bus (but has a 16-bit internal bus).

Min and Max Modes

In order to support the widest possible range of applications, the 8086 and 8088 can operate in one of two modes, called minimum and maximum modes. This allows the user to define certain processor pins to "tailor" the 8086 or 8088 to the intended system. These modes are selected by strapping the MN/MX (minimum/maximum) input pin to V_{cc} or ground.

In the minimum mode, the microprocessor supports small, single-processor systems using a minimum of components. In this mode, the 8086 or 8088 itself generates all the required bus control signals (Figure 22).

In the maximum mode, the microprocessor supports larger, higher performance, or multiprocessing systems. In this mode, the 8086 or 8088 generates status outputs which are decoded by the Intel 8288 Bus Controller to provide an extensive set of bus control signals, and Multibus compatibility (Figure 23). This allows higher performance RAM operation because the memory read and write commands are generated more quickly than is possible in the minimum mode. The maximum mode is the one most often used in iAPX-86 and iAPX-88 systems.

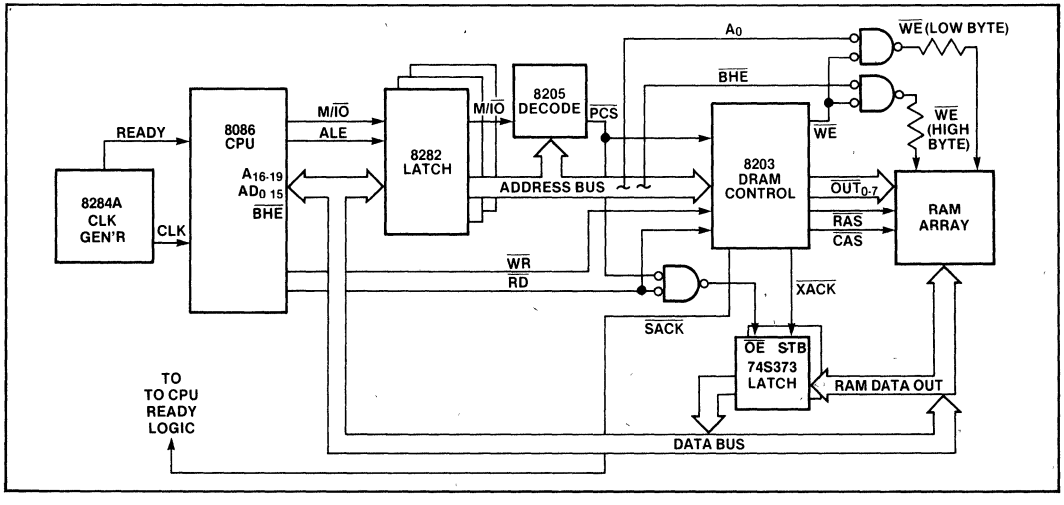


Figure 22. 8086 Minimum Mode

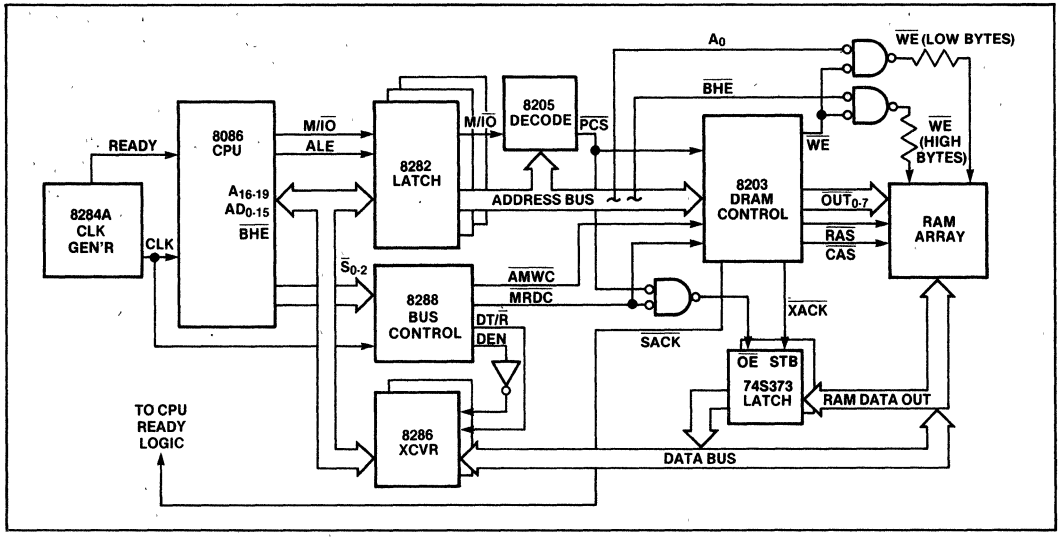


Figure 23. 8086 Maximum Mode

Alternate Configuration

The Alternate Configuration is not an operating mode of the 8086 or 8088 *per se*, but uses TTL logic along with the status outputs of the microprocesor to generate the RAM read and/or write control signals (Figure 24). The alternate configuration may be used with the microprocessor in either minimum or maximum mode. This configuration is advantageous because it activates the memory read and write signals even earlier than the maximum mode, leading to higher performance. It is possible to generate either the RAM read or write signal using this configuration, and generate the other RAM control signal using the min or max mode in the normal configuration.

Each of the three system configurations may be used with buffers on the address, data, or control bus for increased electrical drive capability.

Performance vs. Wait States

Before starting a discussion of timing analyses, it's worthwhile to look at the effect of wait states on the iAPX-86 and iAPX-88.

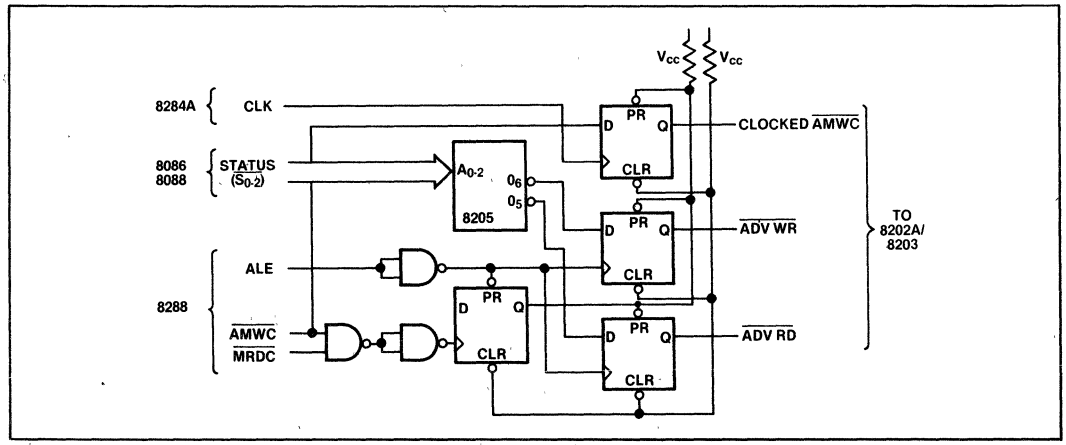


Figure 24. Alternate Configuration Logic

For most microprocessors, the effect of, say, one wait state on execution times is straightforward. If a bus cycle normally is three clocks long, adding a wait state to every bus cycle will make all bus cycles four clocks, decreasing performance by 33%. This is multiplied by the percentage of time that the microprocesor is doing bus cycles (some instructions take a long time to execute, so the microprocessor skips a few bus cycles).

The effect of wait states on the iAPX-86 and iAPX-88 is not so straightforward, however.

The 8086 and 8088 microprocessors consist of two processing units: the execution unit (EU) executes instructions, and the bus interface unit (BIU) fetches instructions, reads operands, and writes results. During periods when the EU is busy executing instructions, the BIU "looks ahead" and fetches more instructions from the next consecutive addresses in memory; these are stored in an internal queue. This queue is four bytes long for the 8088 and six bytes long for the 8086; under most conditions, the BIU can supply the next instructions without having to perform a memory cycle. Only when the program doesn't proceed serially (e. g. a Jump or Call instruction) does the EU have to wait for the next instruction to be fetched from memory. Otherwise, the instruction fetch time "disappears" as it is proceeding in parallel with execution of previously fetched instructions. The EU then has to wait for the BIU only when it needs to read operands from memory or write results to memory. As a result, the 8086 and 8088 are less sensitive to wait states than other microprocessors which don't use an instruction queue. The effect of wait states on 8086 execution time compared to the Motorola 68000 and Zilog Z8000 for a typical mix of software is summarized in Table 3.[1]

Table 3. Effects of Wait States on Execution Time

	Execution Time Increase Over 0 Wait State Execution Time		
Processor	**1 Wait State**	**2 Wait States**	**3 Wait States**
iAPX 86/10 (measured)	8.3%	16.3%	26.3%
Z8000 (computed)	19.1%	38.2%	57.3%
68000 (computed)	15.9%	31.9%	47.8%

The BIU can fetch instructions faster than the EU can execute them, so wait states only affect performance to the extent that they make the EU wait for the transfer of operands and results. How much this affects program execution time is a function of the software; programs that contain many complex instructions like multiplies and divides and register operations are slowed down less than programs that contain primarily simple instructions. The effect of wait states on the 8086 and 8088 is always less than on other microprocessors which don't use an instruction queue.

[1] From *16-Bit Microprocessor Benchmark Report: iAPX-86, Z8000, and 68000,* publ. by Intel Corp. 1980

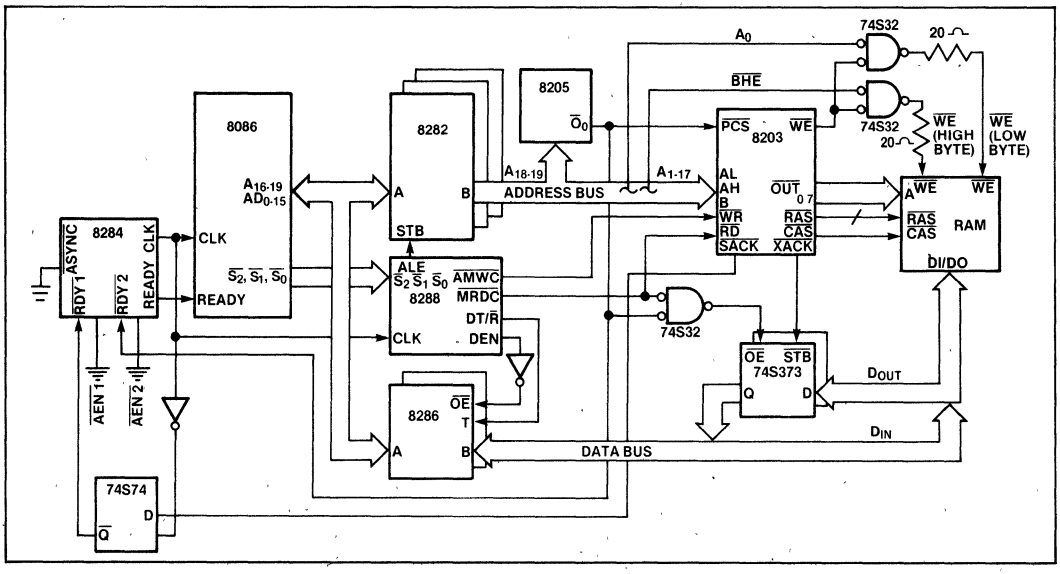


Figure 25. 8086 Max Mode System

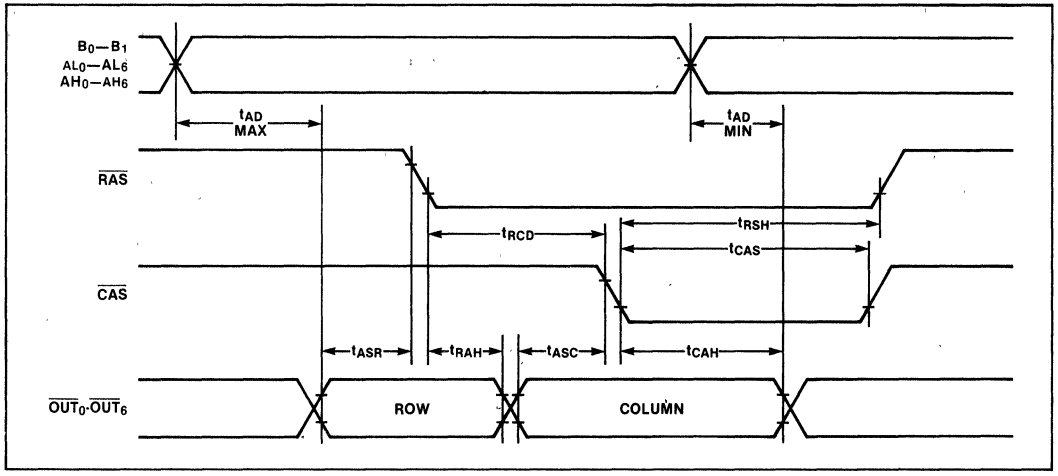


Figure 26. Memory Compatibility Timing

Timing Analysis

This section will look at two specific system configurations to show how the 8203 timing requirements are satisfied by the 8086. Methods of determining the worst case number of wait states for the various configurations are also given.

The timings of the 8202A and 8203 are identical; only the 8203 is referred to for the remainder of this note, but all comments apply equally to the 8202A. A11 timings are worst case over the range of T_A = 0 - 70°C and V_{cc} = + 5v ± 10% for the test conditions given in the devices' data sheets.

Example 1. 8086 Max Mode System (5 MHz)

This example (Figure 25) is representative of a typical medium-size microprocesor system. Example 1 requires one wait state (worst case) for memory cycles. Example 2 also uses an 8086 in Max mode at 5 MHz, but uses external logic to reduce the number of wait states to zero for both read and write cycles.

DYNAMIC RAM INTERFACE

First, look at the timing requirements of the dynamic RAM to ensure they are satisfied by the 8203. Memory compatibility timings are shown in the 8203 data sheet (Figure 26). Seven 8203 timings are given, not counting t_{AD}, which will be discussed in the next section. These timings are summarized in Table 4.

Table 4. Memory Compatibility Timings
(all parameters are minimums)

Symbol	Parameter	Value
t_{ASC}	Column Address Set-Up Time	t_p-30
t_{ASR}	Row Address Set-Up Time	t_p-30
t_{CAH}	Column Address Hold Time	$5t_p$-30
t_{CAS}	$\overline{CAS}$ Pulse Width	$5t_p$-10
t_{RAH}	Row Address Hold Time	t_p-10
t_{RCD}[1]	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	$2t_p$-40
t_{RSH}	$\overline{RAS}$ Hold Time from $\overline{CAS}$	$5t_p$-30

[1] $t_{RCD}min = t_{RAH}min + t_{ASC}min = 2_p - 40$
This parameter is the minimum RAS active to CAS active delay.

These timings are all a function of the 8203's clock period (t_p); they may be adjusted to be compatible with slower dynamic RAMs by slowing the 8203's clock (increasing t_p). The frequency of the 8203's clock may be varied from 18.432 MHz to 25 MHz; for best performance, the 8203 should be operated at the highest possible frequency compatible with the chosen dynamic RAM. In most cases, t_{RAH} or t_{CAS} will be the frequency limiting parameter, but the 8203 can operate at its maximum frequency with most dynamic RAMs available.

t_{ASR} applies only to refresh cycles. When the 8203 is in the Idle state (not performing any memory or refresh cycles) the address multiplexer allows the AL_{0-7} inputs (the RAM row address) to propagate through to the 8203 OUT_{0-7} pins, which are connected to the RAM address pins. So in read or write cycles, the row address will propagate directly from the address bus to the

RAM; the row address set-up time in this case is determined by the microprocessor's timing (see the next section). At the beginning of a refresh cycle, the 8203 has to switch its internal multiplexer to direct the refresh row address to the RAMs before activating RAS; the t_{ASR} parameter in Table 4 refers to this case only.

Assume the Intel 2164A-20 RAM (200 ns access time) is used. Equations 1(a)-(h) show that this RAM is compatible at the 8203's maximum operating frequency of 25 MHz (t_p = 1/(25 MHz) = 40 ns). This frequency will be used for now; once the rest of the system timings are calculated, the minimum 8203 frequency which will provide the same system performance can also be determined.

(a) $t_{ASC} = t_p - 30 = 10$ (Equation 1.)
(b) $t_{ASR} = t_p - 30 = 10$
(c) $t_{CAH} = 5t_p - 30 = 170$
(d) $t_{CAS} = 5t_p - 10 = 190$
(e) $t_{RAH} = t_p - 10 = 30$
(f) t_{RCD}[1] $= 2t_p - 40 = 40$
(g) $t_{RP} = 4t_p - 30 = 130$
(h) $t_{RSH} = 5t_p - 30 = 170$

[1] May be calculated as
$t_{RCD}min = t_{RAH}min + t_{ASC}min = 2t_p - 40$

ADDRESS SET-UP AND HOLD TIME MARGINS

The microprocessor must put the memory address on the address bus early enough in the memory cycle for it to pass through the 8203 and meet the row address set-up time to RAS (t_{ASR}) requirement of the dynamic RAM (Figure 27). Since the address propagates directly through the 8203, this set-up time is a function of how long the microprocessor holds the address on the bus before activating the RD or WR command, the address delay through the 8203 (t_{AD}max), and how long the 8203 waits before activating RAS (t_{CR}min). This is shown in Figure 28, and calculated in Equation 2. This and all following equations show timing margins; a positive result indicates extra margin, a zero result says the parameter is just met, and a negative result indicates it is not met for worst-case conditions.

Row Address Set-Up Time Margin (Equation 2.)
= CPU Address to $\overline{RD}$ Delay + $\overline{RAS}$ Active Delay - Address Delays
= TCLCL(5MHz) + TCLML min (8288) + t_{CR}min(8203) - [Greater of TCLAVmax(8086) + TIVOVmax (8282) or TCLLHmax(8288) + TSHOVmax(8282)] - t_{AD}max(8203) - t_{ASR}(2164A—20)
= 200 + 10 + [40 + 30] - [Greater of (110 + 30) or (15 + 45)] - 40 - 0
= 100

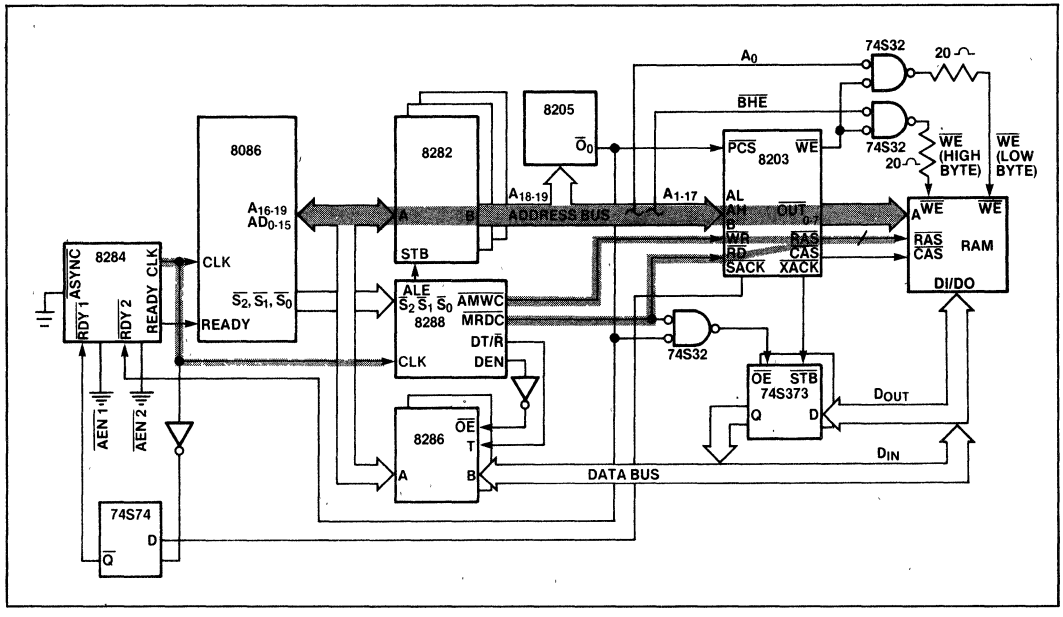


Figure 27. Address Set-Up and Hold Time Margins

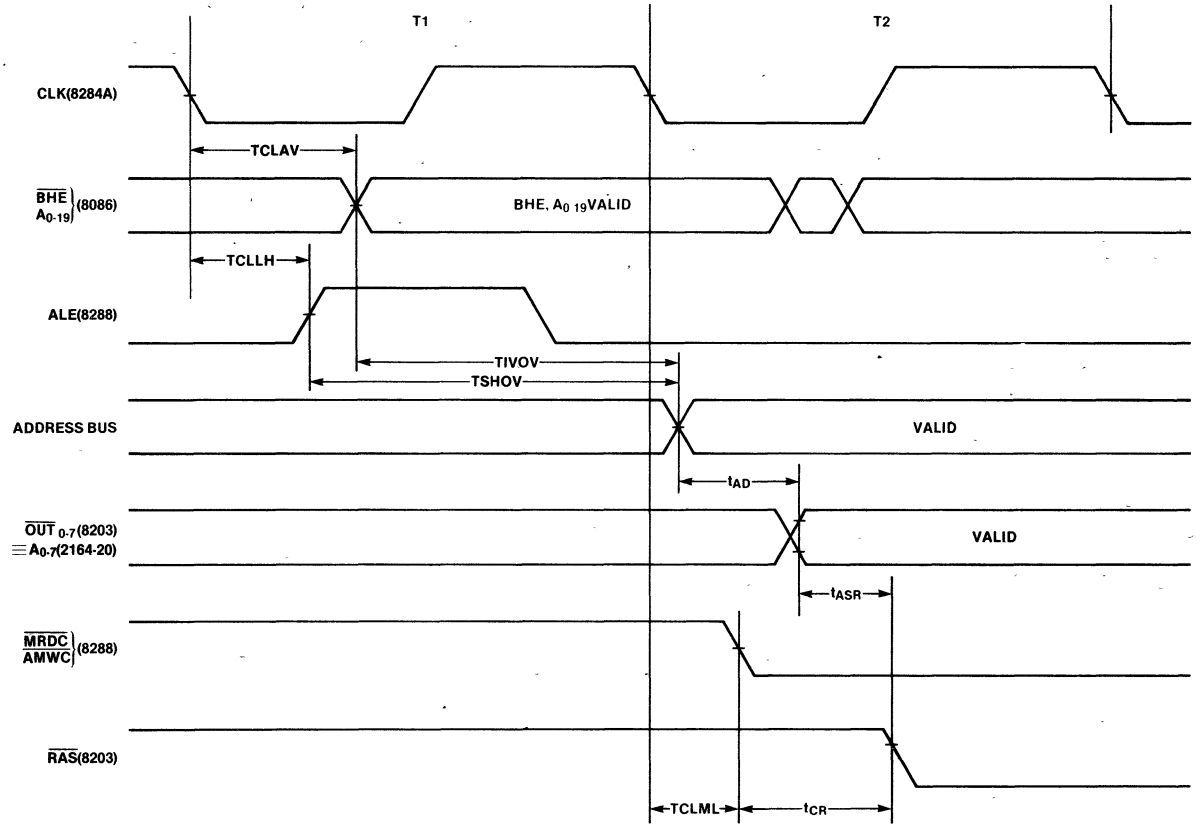


Figure 28. Address Set-up Time Margin

AFN: 02200A

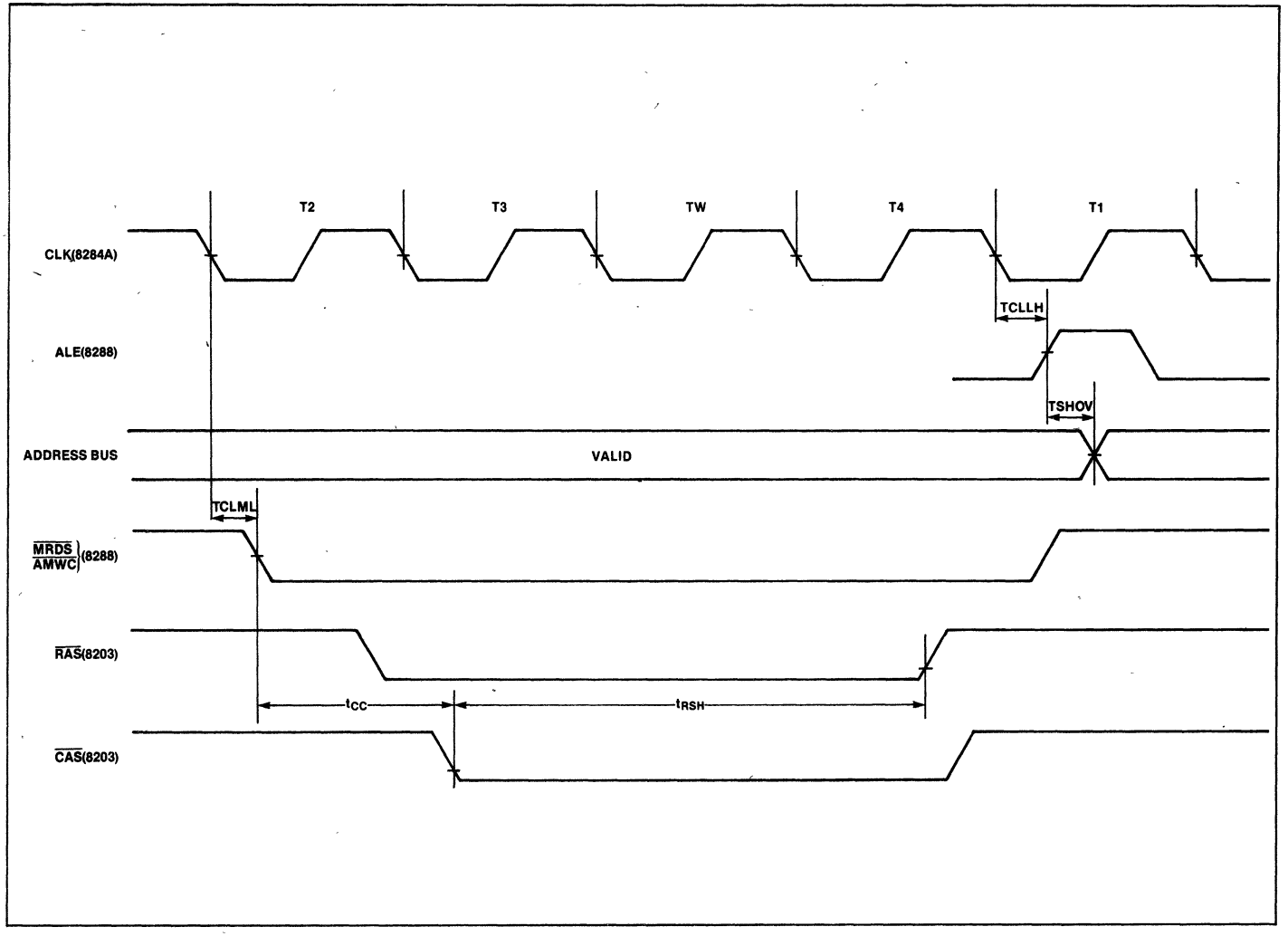


Figure 29. Address Hold Time Margin

AFN: 02200A

Similarly, the microprocessor must maintain the memory address long enough to satisfy the column address hold time (t_{CAH}) of the RAM; the 8203 T_{AD}min parameter should be used for this calculation.

More importantly, the 8203 bank select (B_{0-1}) inputs are also not latched; these are used directly to decode which $\overline{RAS}$ output is activated during read or write cycles, so these inputs must be held valid until $\overline{RAS}$ goes inactive. Since B_{0-1} are usually taken directly from the address bus, this determines the address hold time required of the system (Figure 29). These are easily satisfied by the 8086 as shown by Equation 3. N represents the number of wait states. This equation can be tried with various values for N (starting with 0 and increasing) until the equation is satisfied, or it can be set equal to zero (meaning no excess margin remains) and solved for N directly; the fractional value for N that results must be rounded *up* to get the worst-case number of wait states to satisfy this particular parameter. No wait states are required to meet address hold times.

Address Hold Time Margin (N = 0) (Equation 3.)

= CPU Address Hold Time, from $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays

= (3+N)TCLCL(5MHz) + TCLLHmin(8288)[1] + TSHOVmin(8282) - TCLMLmax(8288) - t_{CC}mas(8203) - t_{RSH}max(8203)[2]

= 3(200) + 2 + 10 - 35 - [4(40) + 85] - [5(40) + 30]

= 102

READ DATA ACCESS TIME MARGIN

Read data access times determine how many wait states are required for read cycles. Remember that dynamic RAMs have two access time parameters, $\overline{RAS}$ access time (t_{RAC}) and $\overline{CAS}$ access time (t_{CAC}). Either one may be the limiting factor in determining RAM access time, as explained in the section *Dynamic RAM - Access Times*, above. Here t_{CAC} is the limiting factor, as

$$t_{CC}\text{max} + t_{CAC}\text{max} \geq t_{CR}\text{max} + t_{RAC}\text{max}.$$

This timing is shown in Figures 30 and 31, and is calculated in Equation 4. In this system, one wait state is required to satisfy the read data access time requirements of the system; the margin is -50 ns, which is too large a difference to be made up by using a faster RAM.

[1] Not specified — use 2 ns

[2] Not specified in 8203 data sheet; t_{RSH}max(8203) = $5t_p$ + 30

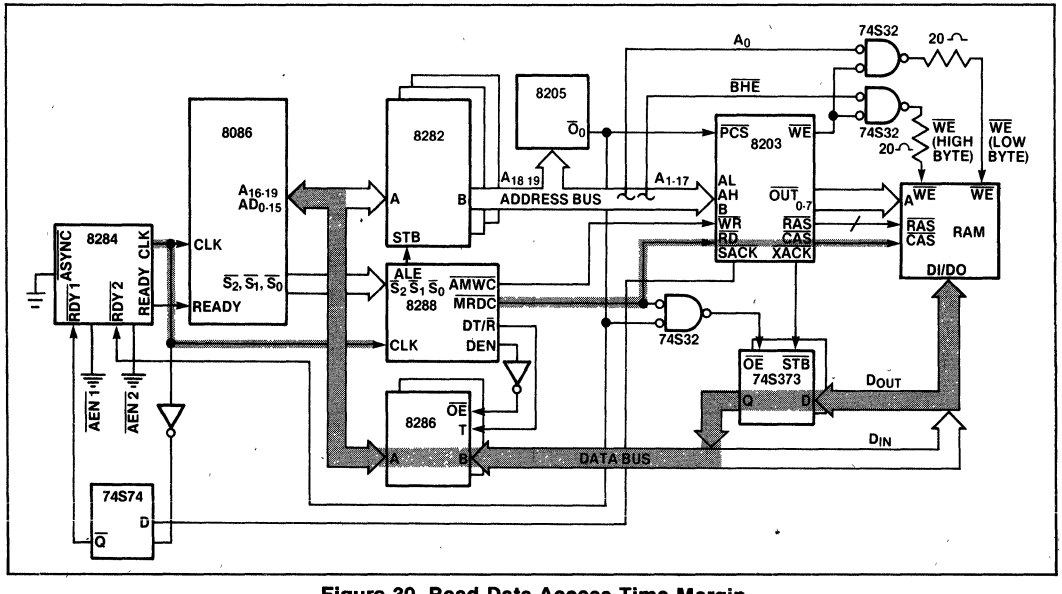


Figure 30. Read Data Access Time Margin

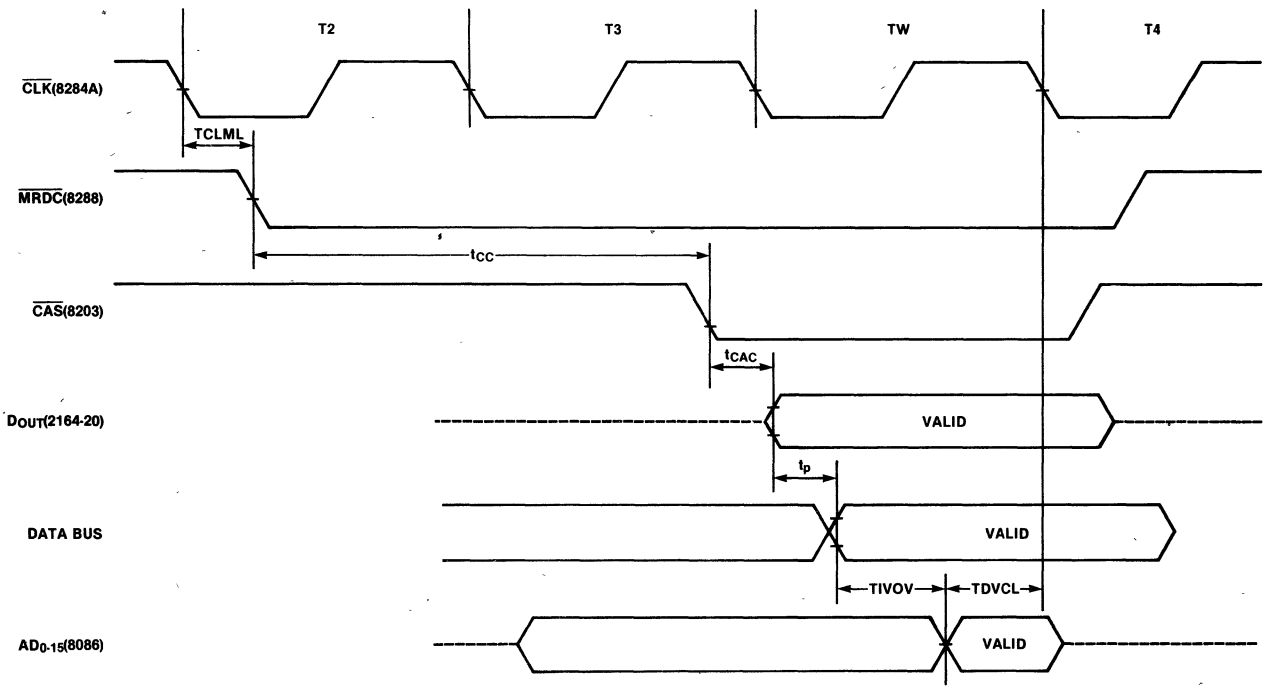


Figure 31. Read Data Access Time Margin

AFN: 02200A

Read Data Access (Equation 4.)
Time Margin (N = 0)

= CPU $\overline{RD}$ Active to Data Valid Delay - $\overline{CAS}$ Active Delay - Data Delays

= (2+N)TCLCL(5MHz) - TCLMLmax(8288) t_{CC}max(8203) - t_{CAC}max(2164A—20) - $t_P$max(74S373)[1] - TIVOVmax(8286) - TDVCLmin(8086)

= 2(200) - 35 - [4(40) + 85] - 110 - 30[1] - 30 - 30

= - 80⇒1 wait state needed (N = 1)

WRITE DATA SET-UP AND HOLD TIME MARGINS

In write cycles, the write data must

1. reach the dynamic RAMs long enough before $\overline{CAS}$ to meet the RAM's data set-up time parameter, t_{DS} (Figures 32 and 33), and

2. be held long enough after $\overline{CAS}$ to meet the RAM's data hold time parameter (t_{DH}) (Figures 32 and 34.)

Data set-up time margin is calculated in Equation 5, and data hold time margin is given in Equation 6. Again, these are margins, so a positive number indicates that system timing requirements are met for worst-case timings. Data hold time is a function of the number of 8086 wait states, represented as N, as is the read data access time margin. No wait states are required to meet this parameter.

Write Data Set-Up Time Margin (Equation 5.)

= CPU $\overline{WR}$ Active to Data Valid Delay + $\overline{CAS}$ Delay - Data Delay

= TCLMLmin(8288) + t_{CC}min(8203) - TCLDVmax(8086) - TIVOVmax(8286) - t_{DS}min(2164A-20)

= 10 + [3(40) + 25] - 110 - 30 - 0

= 15

Write Data Hold Time (Equation 6.)
Margin (N = 0)

= CPU Data Hold Time, from $\overline{AMWC}$ Active + Data Delays - $\overline{CAS}$ Active Delay

= (2 + N)TCLCL(5MHz) + TCLCHmin(8284A) + TCHDXmin(8086) + TIVOVmin(8286) - TCLMLmax(8288) - t_{CC}max(8203) - t_{DH}min(2164A—20)

= 2(200) + [⅔(200) - 15] + 10 + 5 - 35 - [4(40) + 85] - 45

= 308

[1] t_p(74S373) is the greater of t_{PHL} (from data) or t_{PLH} (from data) and is compensated for V_{cc} and temperature variations, and is derated for a 300_pF load (T.I. spec is at 15_pF).

t_p(74S373) = 13ns + 0.05ns/$_pF(300 - 15)_pF$ + 2.75ns = 30ns.

Where 13ns is T.I. spec value
0.05ns/$_pF$ is derating factor for excess capacitive load
(300 - 15) is excess capacitive load
2.75 is compensation for T_A and V_{cc} variation

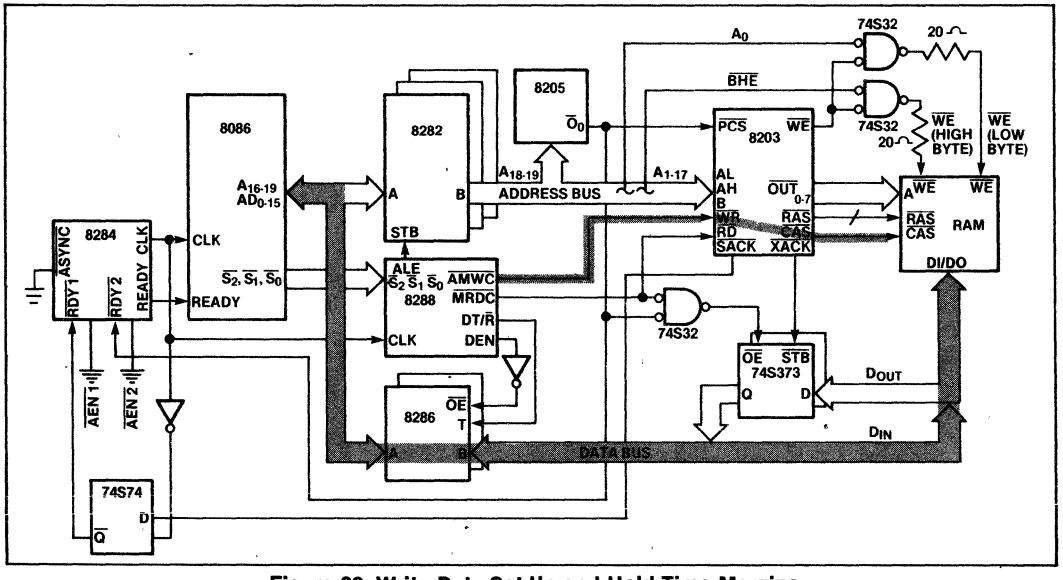


Figure 32. Write Data Set-Up and Hold Time Margins

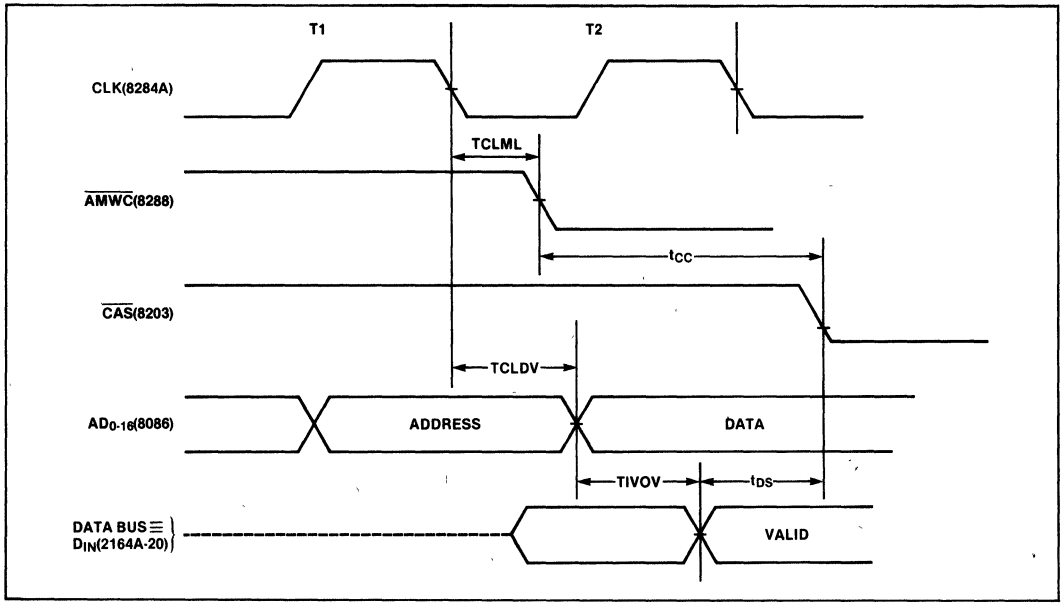


Figure 33. Write Data Set-Up Time Margin

SACK SET-UP TIME MARGIN

As explained earlier, SACK (and XACK) are "handshaking" signals used to tell the microprocessor when it may terminate the bus cycle in progress. Thus, SACK timing determines how many wait states will be generated, as opposed to how many wait states are actually required for proper operation, which is determined by the read data access time for read cycles and by the write data hold time for write cycles. If SACK causes more wait states than are required, there is a performance penalty, but the system operates; if too few wait states are generated, the system will not function.

SACK and XACK serve the same function; they differ only in timing. XACK is Multibus compatible, and is activated only when the read data is actually on the bus (in a read cycle) or when the write data has been latched into the RAM (in a write cycle). SACK is activated earlier in the memory cycle than XACK to compensate for delays in the microprocessor responding to this signal to terminate the cycle. Use of SACK is normally preferable, as it results in the fewest possible wait states being generated. But in some systems, SACK will not generate a sufficient number of wait states, so XACK or a delayed form of SACK must be used. Note that the number of wait states generated by SACK and XACK will vary, depending on whether a refresh cycle is in progress when the memory cycle was requested, and if refresh cycle is in progress, how near it is to completion. SACK is sampled by the 8284A Clock Generator Chip's RDY1 or RDY2 input. The 8284A can be programmed to treat these inputs as either synchronous or asynchronous inputs by tying its ASYNC input (pin 15) either high or low, respectively. SACK must be treated as asynchronous unless it has been synchronized to the microprocessor's clock with an external flip-flop.

SACK set-up time is shown in Figures 35 and 36, and is calculated in Equation 7. This equation indicates that, at worst case, one wait state will be generated (n = 1). This satisfies the requirements of the system, namely one wait state for reads and zero (or more) wait states for writes.

SACK Set-Up Time Margin (N = 0) (Equation 7.)

= RD or WR Active to SACK Active Delay

= (N)TCLCL(5MHz) + t_{PLH}min(7404)[1] - TCLMLmax(8288) - t_{CA}max(8203) - t_{SU}min(74S74)

= 0 + 1 - 35 - [2(40) + 47] - 3

= -164 => 1 wait state wil be generated (N = 1)

We have only looked at "worst case" SACK set-up time so far, to determine the maximum number of wait states that will be generated (assuming no delays due to a refresh cycle in progress). We should look at "best

[1] Not specified — use 1 ns.

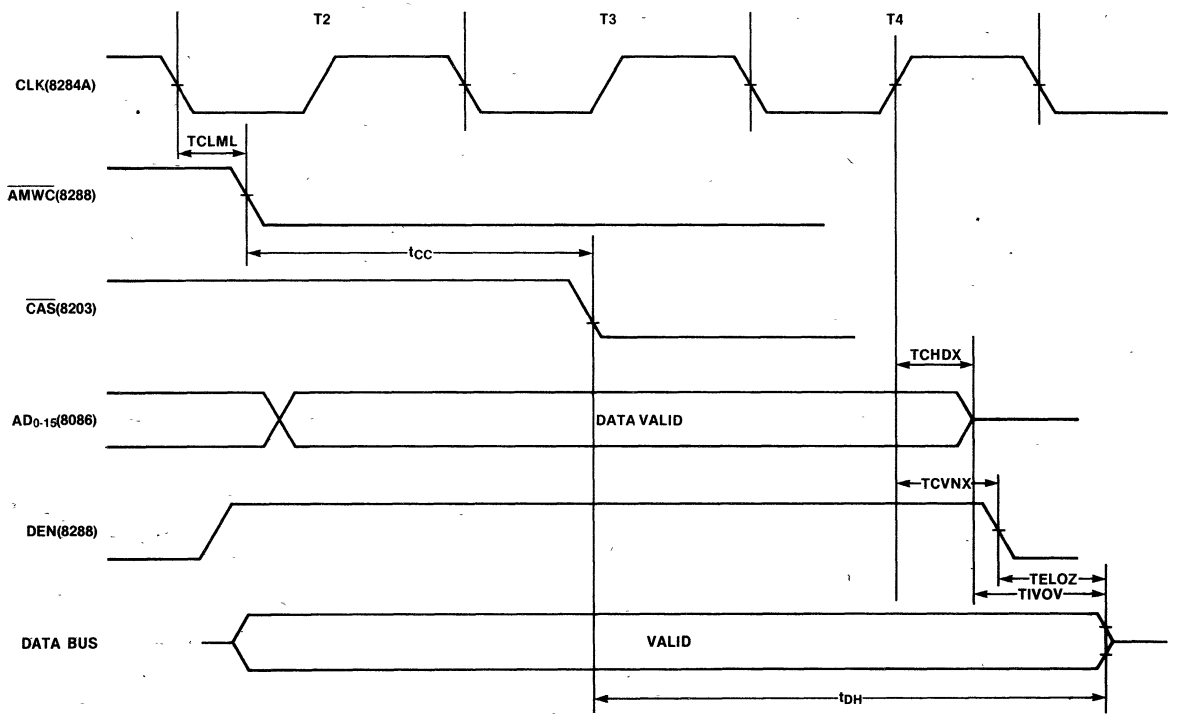


Figure 34. Write Data Hold Time Margin

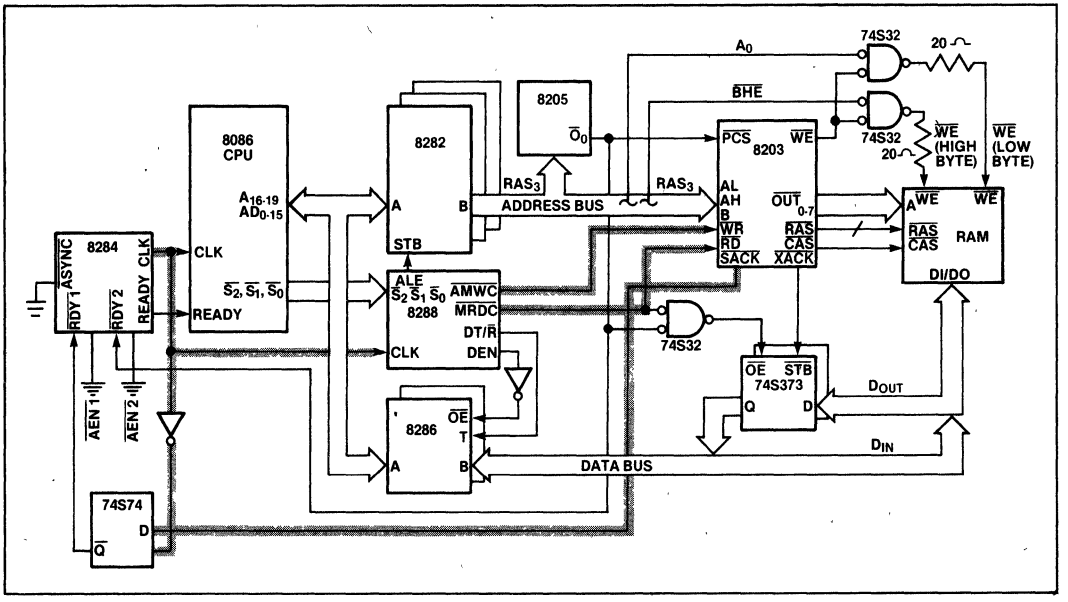


Figure 35. SACK Set-Up Time Margin

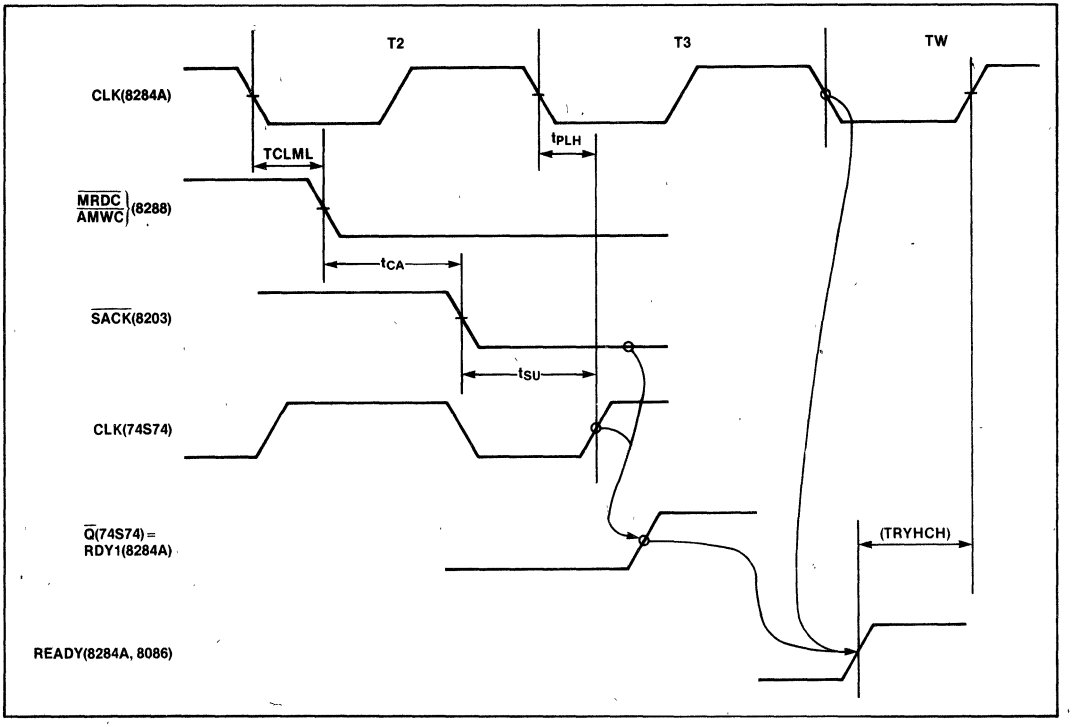


Figure 36. SACK Set-Up Time Margin

case" $\overline{SACK}$ timing also, to make sure enough wait states are always generated. Note that in Figure 35, $\overline{SACK}$ goes through an external 74S74 flip-flop; this samples $\overline{SACK}$ on-half clock cycle earlier than the 8284A does (on the same clock edge that activates $\overline{MRDC}$ or $\overline{AMWC}$), effectively reducing $\overline{SACK}$ set-up time by one-half clock period. This guarantees the proper number of wait state will be generated for "best case" $\overline{SACK}$ timing. Adding this flip-flop does not increase the worst case number of wait states generated by $\overline{SACK}$.

In the case where a memory cycle is requested while a refresh cycle is in progress, the memory cycle will be delayed by a variable amount of time, depending on how near the refresh cycle is to completion. This delay may be as long as one full memory cycle if the refresh was just starting; this time is about 650 ns, depending on the 8203's clock frequency. $\overline{SACK}$ set-up, read data set-up, and write data hold times to the microprocessor's clock are not the same as in the usual case where there is no refresh interference. In this case, $\overline{SACK}$ is delayed until the read or write cycle has been completed by the RAM, so that there is no possibility of terminating the cycle too soon.

PCS SET-UP TIME MARGIN

The 8203's $\overline{RD}$, $\overline{WR}$, and ALE inputs must be qualified by $\overline{PCS}$ in order to perform a memory cycle. If the $\overline{PCS}$ active set-up time parameter (t_{PCS}) is violated, the memory cycle will be delayed. In this case all maximum delays normally measured from command (t_{CR}, t_{CC}, t_{CA}) will be measured instead from $\overline{PCS}$ active and will be increased by t_{PCS} (20 ns). Minimum t_{CR}, t_{CC}, t_{CA} delays remain the same, but are measured from command or $\overline{PCS}$ whichever goes active later. If t_{PCS} is violated, care must be taken that $\overline{PCS}$ does not glitch low while $\overline{RD}$, $\overline{WR}$, or ALE is active, erroneously triggering a memory cycle. t_{PCS} is not violated in this system, however (Equation 8).

$\overline{PCS}$ Set-Up Time Margin (Equation 8.)

= CPU Address Valid to Command Active Delay - $\overline{PCS}$ Decode Time

= TCLCL(5MHz) + TCLMLmin(8288) - [Greater of TCLAVmax(8086) + TIVOVmax(8282) or TCLLHmax(8288) + TSHOVmax(8282)] - $t_p$max(8205) -t_{PCS}min(8203)

= 200 + 10 - [Greater of (110 + 30) or (15 + 45)] - 18 - 20

= 32

RAM DATA OUT HOLD TIME MARGIN

The 8203 CAS output is only held valid for a fixed length of time during a read cycle, after that the RAM data outputs are 3-stated. This time is not long enough to allow the 8086 to read the data from the bus, so the data must be latched externally. This latch should be a transparent type and should be strobed by $\overline{XACK}$ from the 8203. Because the minimum time from $\overline{XACK}$ active to $\overline{CAS}$ inactive is only 10 ns, a latch with a data hold time requirement of 10 ns or less (such as a 74S373) should be used (see Equation 9).

RAM Data Out Hold Time Margin, from $\overline{XACK}$ Active (Equation 9.)

= t_{ACK}min(8203) + t_{OFF}min(2164A - 20) - $t_H$min(74S373)[1]

= 10 + 0 - 10

= 0

OTHER CALCULATIONS

Equations 3, 4, 6 and 7 may be solved directly for N, where N is the number of wait states, to find how many wait states are required at a given frequency. Alternatively, a number may be substituted for N and these equations solved for the 8086's clock period, TCLCL, to find the maximum microprocessor frequency possible with N wait states. Note that the clock high and low times (TCHCL and TCLCH) are also a function of TCLCL. Be sure to use the proper speed selection of the 8086 in this calculation, as various A.C. parameters are different and the result may be different for different speed selections of the 8086, even at the same frequency. Be sure to check the other equations at this frequency to make sure they are OK, too.

Finally, for given values of TCLCL and N, Equations 3, 4, 6, and 7 may be checked to find the lowest 8203 clock frequency which will allow the same system performance, if it is desired to operate at some frequency other than the 25 MHz we assumed.

CONCLUSION

This design will operate with, at worst case, one wait state (except for refresh) at microprocessor frequencies up to 6 MHz, using slow (200 ns access time) dynamic RAMs. At 6 MHz, it is limited by a lack of $\overline{SACK}$ set-up

[1] A 74S373 must be used to meet this timing requirement. Even though worst case margin is 0 ns, this is not a critical timing, as valid data will hold on the latch inputs for a considerable time after the RAM outputs 3-state.

time. At 5 MHz, the 8203 can be operated at any clock frequency from 18.432 MHz to 25 MHz, still with only one wait state.

Example 2. 8086 Alternate Configuration System (5 MHz)

Figure 37 shows another 8086 Max mode system at 5 MHz, but this time using the Alternate Configuration, which allows it to operate with *no wait states* (except for refresh).

The system in the previous example was limited by $\overline{SACK}$ set-up time. $\overline{SACK}$ set-up time can be improved by sampling $\overline{SACK}$ later; this has been done by changing the clock edge used to sample $\overline{SACK}$, allowing roughly ⅔ clock period longer. $\overline{SACK}$ set-up time (and read data access time and write data hold time) margin can also be improved by activating the $\overline{RD}$ or $\overline{WR}$ inputs of the 8203 earlier in the 8086's bus cycle; this is the purpose of the extra logic in Figure 37 (I.C.s A8 - A11). These generate advanced $\overline{RD}$ and $\overline{WR}$ signals timed from the falling edge of ALE, which occurs roughly ⅓ clock period sooner than the $\overline{MRDC}$ and $\overline{AMWC}$ are generated by the 8288 Bus Controller. Altogether, these changes allow about one 8086 clock period more set-up time for $\overline{SACK}$.

Let's look at this logic in more detail. An Intel 8205 (A8) is used to decode the 8086's status outputs $\overline{S_{0\text{-}2}}$. An opcode fetch, memory read, or memory write decode to 8205 outputs 4, 5, and 6, respectively. These outputs go to the D inputs of two 74S74 flip-flops. The Q output of flip-flop A10.2 is an advanced memory read signal and the Q output of A11.2 is an advanced memory write signal. As shown in Figure 37, the 8203 is not activated for opcode fetches, but it can be if 8205 outputs 4 and 5 are ORed with the unused 74S00 gate (A9.4) and the $\overline{Q}$ output of A10.2 used instead of Q. Both flip-flops are clocked by the falling edge of ALE to generate the advanced commands. Flip-flop A10.1 is clocked by the trailing edge of either $\overline{AMWC}$ (Advanced Memory Write Command) or $\overline{MRDC}$ (Memory Read Command) from the 8288 bus controller (A6), indicating that the 8086 has completed the memory cycle. A10.1, in turn, presets both the A10.2 and A11.2 flip-flops to terminate the advanced memory read and write signals to the 8202A. A10.1 is then preset to its initial state by ALE going active at the start of the next bus cycle.

Because RAM write cycles are started very early in the 8086's bus cycle using this logic, the 8203 will activate $\overline{CAS}$ to the RAMs (latching write data) before the data is valid from the 8086. This requires delaying $\overline{WE}$ to the RAMs and performing a "late write" (explained earlier under *Dynamic RAMs*) in order to allow more time for the write data to arrive. But the $\overline{WE}$ signal must not be delayed so long that there is no longer enough data hold time, measured from when $\overline{WE}$ goes active; or that the $\overline{WE}$ active to $\overline{CAS}$ inactive delay spec or the RAM (t_{RWL}) is violated. None of the control signals from the 8086 or 8288 bus controller satisfy both of these timing constraints, so such a signal is generated by flip-flop A11.1, which serves to delay $\overline{AMWC}$ from the bus controller by an amount of time equal to TCLCH (the low time of the 8086's clock). A11.1 is also preset by A10.1 at the end of the memory cycle. The Q output of A11.1 is ANDed with $\overline{WE}$ from the 8203 by A14.1 to form a delayed RAM $\overline{WE}$. As in the previous example, this signal is then ANDed with $\overline{BHE}$ and AO to form the $\overline{WE}$ for the high and low bytes of RAM, respectively.

A total of four packages (three 14-pin and one 16-pin) of TTL logic are required.

The dynamic RAM interface timings are identical to the last example (Equations 1 (a)-(h)); 2164A-20 RAMs will be used again.

ADDRESS SET-UP AND HOLD TIME MARGINS

Address set-up and hold time margins are given in Equations 10 and 11, respectively. An 8086-*2* microprocessor has been used instead of the standard 8086, as this speed-selected part gives better address set-up to $\overline{RD}$ or $\overline{WR}$ times, which this design needs since it uses advanced $\overline{RD}$ and $\overline{WR}$ commands.

Row Address Set-Up Time Margin[1] (Equation 10.)

= CPU Address to Adv. $\overline{RD}$ Delay
+ $\overline{RAS}$ Delay - Address Delays

= TCLCHmin(8284A) + TCHLLmin(8288)[2]
+ t_{PLH}min(74S00)[3] + t_{PHL}min(74S74)[2]
+ t_{CR}min(8203) - [Greater of
TCLAVmax(8086 - 2) + TIVOVmax(8282)
or TCLLHmax(8288) + TSHOVmax(8282)]
- t_{AD}max(8203) - t_{ASR}min(2164A-20)

= [⅔(200) - 15] + 2 + 1 + 2 + [(40) + 30]
- [Greater of (60 + 30) or (15 + 45)] - 40 - 0

= 63

[1] Read or write cycles only. Eq. 1b gives this timing for refresh cycles.

[2] Not specified — use 2 ns.

[3] Not specified — use 1 ns.

Address Hold Time Margin (N = 0) (Equation 11.)
= CPU Address Hold Time from Adv. $\overline{RD}$ Active - $\overline{RAS}$ Inactive Delays
= (3+N)TCLCL(5MHz) + TCHCLmin(8284A) + TCLLHmin (8288) + TSHOVmin(8282) - TCLMLmax(8288) - t_{CC}max(8203) - t_{RSH}max(8203)
= (3)200 + [⅓(200) + 2] + 2 + 5 - 35 - [4(40) + 85] - [5(40) + 20]
= 175

READ DATA ACCESS TIME MARGIN

Read data access time margin is shown in Equation 12; no wait states are required for read cycles, even with 200 ns access time RAMs.

Read Data Access Time Margin (N = 0) (Equation 12.)
= Adv. $\overline{RD}$ to Data Valid Delay - $\overline{CAS}$ Delay - Read Data Delays
= (2+N)TCLCL(5MHz) + TCHCLmin(8284A) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CC}max(8203) - t_{CAC}max(2164A—20) - t_{p}max(74S373) - TIVOVmax(8286) - TDVCLmin(8086—2)
= (2)200 + [⅓(200) + 2] - 15 - 5 - 10 - [4(40) + 85] - 110 - 30 - 30 - 20
= 3

WRITE DATA SET-UP AND HOLD TIME MARGINS

Write data set-up and hold times are shown in Equations 13 and 14, respectively. No wait states are required during write cycles. Note that write data set-up has been guaranteed by delaying $\overline{WE}$ from the 8203 with clocked $\overline{AMWC}$ from the bus controller and performing "late write" cycles; write data set-up time would not be satisfied otherwise. Equation 15 verifies that $\overline{WE}$ has not been delayed too long to meet the RAM's $\overline{WE}$ active to $\overline{RAS}$ inactive set-up time (t_{RWL}). The RAM's $\overline{WE}$ active to $\overline{CAS}$ inactive set-up time (t_{CWL}) is also satisfied, since $\overline{CAS}$ does not go inactive until at least 20 ns after $\overline{RAS}$.

Write Data Set-Up Time Margin (Equation 13.)
= CPU Data to Clocked $\overline{AMWC}$ Set-Up + $\overline{WE}$ Delays - Data Delays
= TCLCHmin(8284A) + t_{PHL}min(74S74)[1] + (2)t_{PHL}min(74S32)[1] - TCLDVmax(8086-2) - TIVOVmax(8286) - t_{DS}min(2164A—20)
= [⅔(200) - 15] + 2 + (2)2 - 60 - 30 - 0
= 34

Write Data Hold Time Margin (N = 0) (Equation 14.)
= CPU Data Hold Time from Clocked $\overline{AMWC}$ + Data Delays - $\overline{WE}$ Delays
= (2 + N)TCLCL(5MHz)
= TCHDXmin(8086—2) + TIVOVmin(8286) - t_{PHL}max(74S74) - (2)t_{PHL}max(74S32) - t_{DH}min(2164A—20)
= (2)200 + 10 + 5 - 10 - (2)7 - 45
= 346

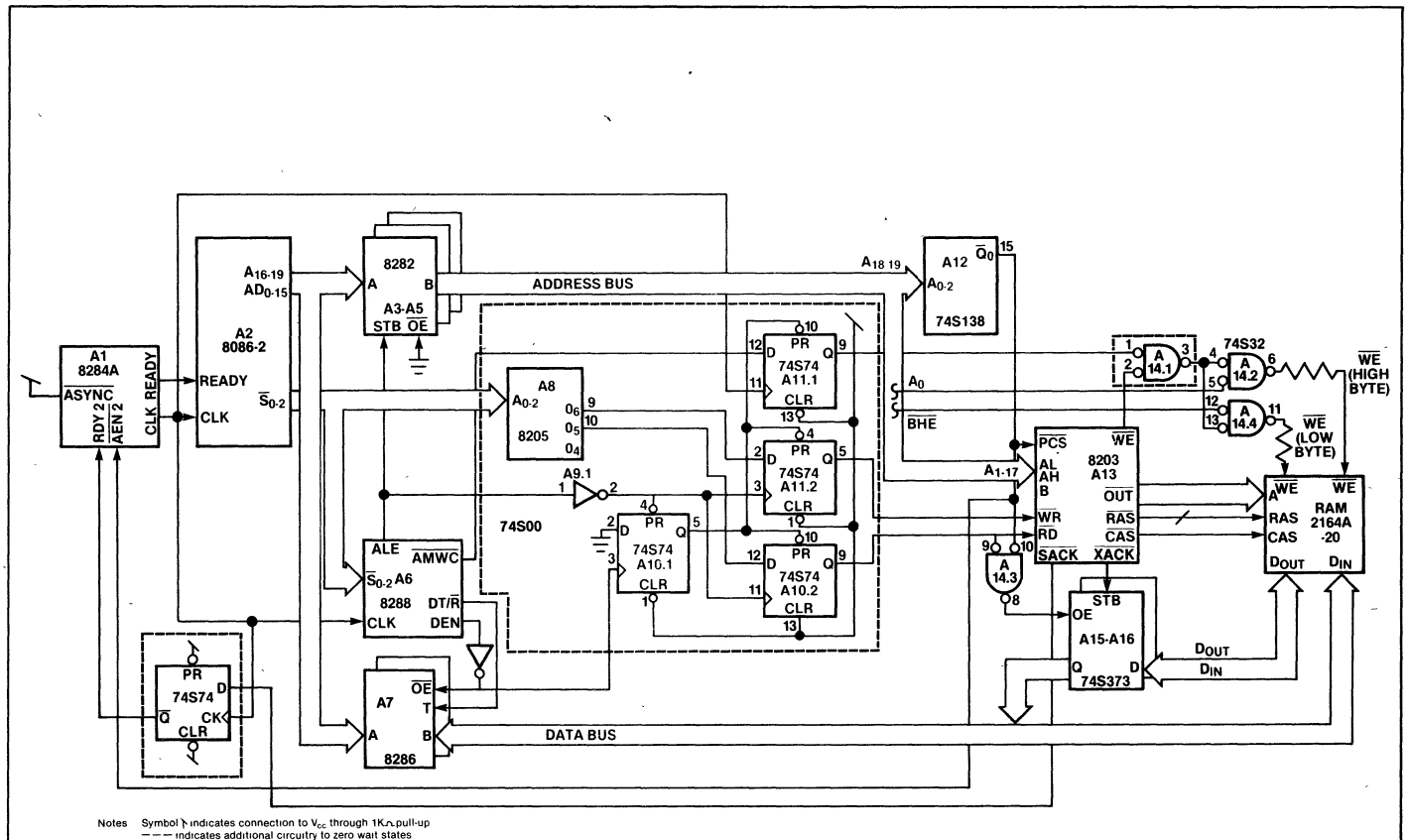
$\overline{WE}$ Active Set-Up Time Margin to $\overline{RAS}$ Inactive (Equation 15.)
= TCHLLmin(8284A)[1] + t_{PLH}min(74S00)[2] + t_{CC}min(8203) + t_{RSH}min(8203) - t_{SKEW}(74S74)[3] -(2)t_{PHL}max(74S32) - t_{RWL}min(2164A—20) - TCLCL(5MHz)
= 2 + 1 + [3(40) + 25] + [5(40) - 30] - 2 - (2)7 - 50 - 200
= 52

$\overline{SACK}$ SET-UP TIME MARGIN

Equation 16 shows that $\overline{SACK}$ set-up time is satisfied; no wait states will be generated for read or write cycles (except for refresh).

$\overline{SACK}$ Set-Up Time Margin (N = 0) (Equation 16.)
= (1 + N)TCLCL(5MHz) - TCHLLmax(8288) - t_{PLH}max(74S00) - t_{PHL}max(74S74) - t_{CA}max(8203) - t_{SU}min(74S74)
= 200 - 35 - 5 - 10 [2(40) + 47] - 3
= 20

[1] Not specified — use 2 ns.
[2] Not specified — use 1 ns.
[3] t_{SKEW}(74S74) is max. skew between t_{PHL}(Q output, from CLK) of two Q outputs in same package — use = 2 ns.



Notes Symbol ⟩ indicates connection to V_{cc} through 1KΩ pull-up
— — — indicates additional circuitry to zero wait states

Figure 37. 8086 Alternate Configuration System

$\overline{PCS}$ Set-Up Time Margin (Equation 17.)

= CPU Address Valid to Adv. $\overline{RD}$ or Adv. $\overline{WR}$ Delay - $\overline{PCS}$ Decode Time

= TCLCHmin(8284A) + TCHLLmin(8288)[1] + t_{PLH}min(74S00) + t_{PHL}min(74S74)[1] - TCLAVmax(8086—2) - TIVOVmax(8282) - $t_p$max(74S138[3] - t_{PCS}min(8203)

= [⅔(200) - 15] + 2 + 1 + 2 - 60 - 30 - 12 - 20

= 1

PCS SET-UP TIME MARGIN

$\overline{PCS}$ set-up time for the 8203 (t_{PCS}) is satisfied, but not with as much margin in the last example (Figure 17).

---

[1] Not specified — use 2 ns.

[2] Not specified — use 1 ns.

[3] Must use 74S138 to maintain $\overline{PCS}$ Set-Up Time Margin.

This is because the $\overline{RD}$ and $\overline{WR}$ commands are activated earlier in the microprocessor's bus cycle, leaving less time to decode $\overline{PCS}$ from the address bus.

CONCLUSION

This design will operate with a guaranteed zero wait states up to 5 MHz using slow (200 ns access time) RAMs. At this frequency, it is limited by both read and write data set-up times, and to a lesser extent, by $\overline{SACK}$ set-up time. Using faster RAMs will not raise the maximum frequency, as write data and $\overline{SACK}$ set-up times are not affected by the RAM speed. The 8203 operating frequency must be 25 MHz.

This design can be used (with some modifications) to allow one wait state performance up to 8086 clock frequency of 8 MHz.

intel

APPLICATION NOTE

AP-141

October 1981

8203/8206/2164A Memory Design

Brad May
Peripherals Applications

order number:210315.001

8203/8206/2164A Memory Design

Contents

ABSTRACT

This Application Note shows an error corrected dynamic RAM memory design using the 8203 64K Dynamic RAM Controller, 8206 Error Detection and Correction Unit and 150 ns 64K Dynamic RAMs with a minimum of additional logic.

The goals of this design are to:

1. Control 128K words × 16 bits (256 KB) of 64K dynamic RAM.
2. Support 150 ns dynamic RAMs.
3. Write corrected data back into dynamic RAM when errors are detected during read operations.
4. To use a minimum of additional logic.

It is not the goal of this design to:

1. Provide the maximum possible performance.
2. Provide features like error logging, automatic error scrubbing and dynamic RAM initialization on power-up, or diagnostics, although these features can be added.

DESIGN

Figure 1 shows a memory design using the 8206 with Intel's 8203 64K Dynamic RAM Controller and 150 ns 64K Dynamic RAMs. As few as three additional ICs complete the memory control function (Figure 2).

For simplicity, all memory cycles are implemented as single-cycle read-modify-writes, shown in Figure 3. This cycle differs from a normal read or write primarily when the dynamic RAM write enable ($\overline{WE}$) is activated. In a normal write cycle, $\overline{WE}$ is activated early in the cycle; in a read cycle, $\overline{WE}$ is inactive. A read-modify-write cycle consists of two phases. In the first phase, $\overline{WE}$ is inactive, and data is read from the dynamic RAM; for the second phase, $\overline{WE}$ is activated and the (modified) data is written into the same word in the dynamic RAM. Dynamic RAMs have separate data input and output pins so that modified data may be written, even as the original data is being read. Therefore data may be read and written in only one memory cycle.

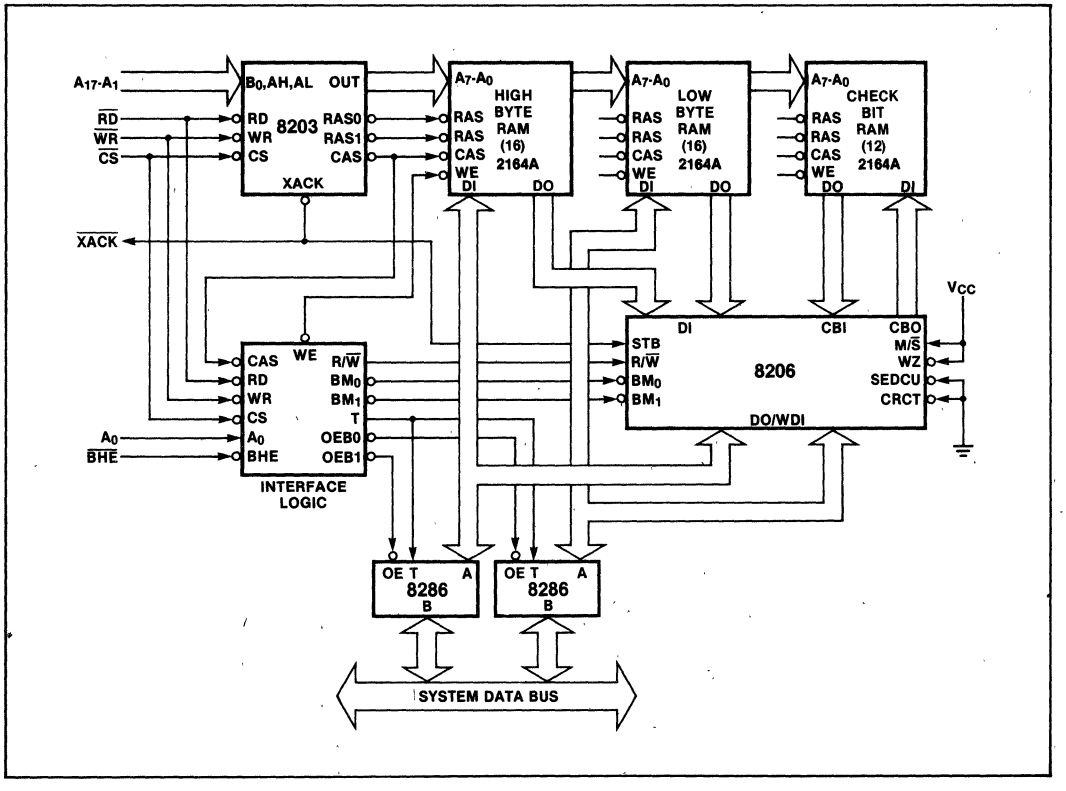


Figure 1. 8203/8206 Memory System

In order to do read-modify-writes in one cycle, the dynamic RAM's $\overline{CAS}$ strobe must be active long enough for the 8206 to access data from the dynamic RAM, correct it, and write the corrected data back into the dynamic RAM. $\overline{CAS}$ active time is an 8203 spec (t_{CAS}), and is dependent on the 8203's clock frequency. The clock frequency and dynamic RAM must be chosen to satisfy Equation 1.

(Eq. 1)

8203		Dynamic RAM		8206		8206		Dynamic RAM		Dynamic RAM
t_{CAS}min	≥	t_{CAC}	+	TDVQV	+	TQVQV	+	t_{DS}	+	t_{CWL}
5(54)−10	≥	85	+	67	+	59	+	0	+	40
260	≥	251		✓						

The 8203 itself performs normal reads and writes. In order to perform read-modify-writes, all that is needed is to change the timing of the $\overline{WE}$ signal. In this design, $\overline{WE}$ is generated by the interface logic in Figure 2—the 8203 $\overline{WE}$ output is not used. All other dynamic RAM control signals come from the 8203. A 20-ohm damping resistor is used to reduce ringing of the WE signal. These resistors are included on-chip for all 8203 outputs.

The interface logic generates the R/$\overline{W}$ input to the 8206. This signal is high for read cycles and low for write cycles. During a read-modify-write cycle, R/$\overline{W}$ is first high, then low. The falling edge of R/$\overline{W}$ tells the 8206 to latch its syndrome bits internally and generate corrected check bits to be written to dynamic RAM. Corrected data is already available from the DO pins. No control signals at all are required to generate corrected data.

R/$\overline{W}$ is generated by delaying $\overline{CAS}$ from the 8203 with a TTL-buffered delay line. This allows the 8206 sufficient time to generate the syndrome; this delay, $t_{DELAY\ 1}$, must satisfy Equation 2.

(Eq. 2)

		Dynamic RAM		8206
$t_{DELAY\ 1}$	≥	t_{CAC}	+	TDVRL
150	≥	85	+	34
150	≥	119	✓	

The 8206 uses multiplexed pins to output first the syndrome word and then check bits. This same R/$\overline{W}$ signal may be used to latch the syndrome word externally for error logging. The 8206 also supplies two useful error signals. $\overline{ERROR}$ signals the presence of an error in the data or check bits. CE tells if the error is correctable (single bit in error) or uncorrectable (multiple bits in error).

In the event that an uncorrectable error is detected, the 8206 will force the Correctable Error (CE) flag low; this may be used as an interrupt to the CPU to halt execution and/or perform an error service routine. In this case the 8206 outputs data and check bits just as they were read, so that the data in the dynamic RAM is left unaltered, and may be inspected later.

After R/$\overline{W}$ goes low, sufficient time is allowed for the 8206 to generate corrected check bits, then the interface logic activates $\overline{WE}$ to write both corrected data and check bits into dynamic RAM. $\overline{WE}$ is generated by delaying CAS from the 8203 with the same delay line

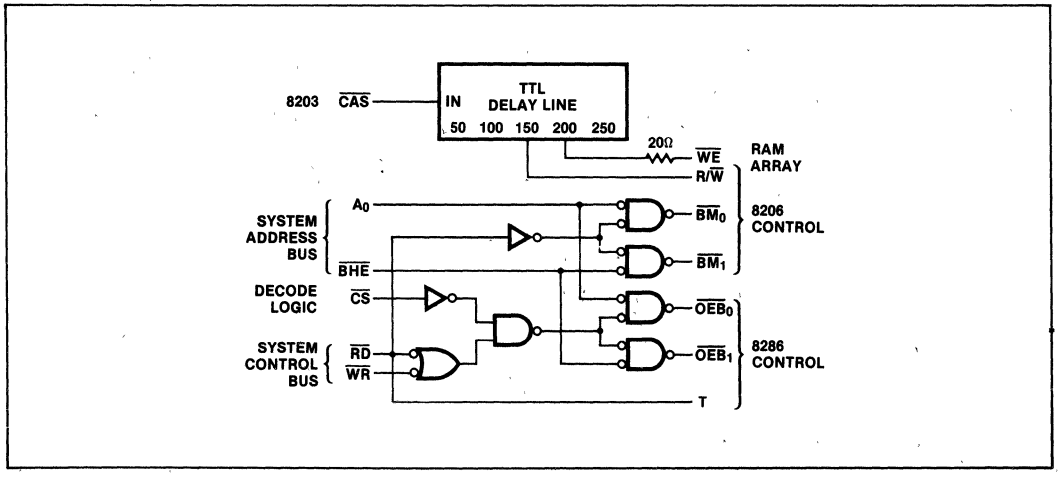


Figure 2. Interface Logic

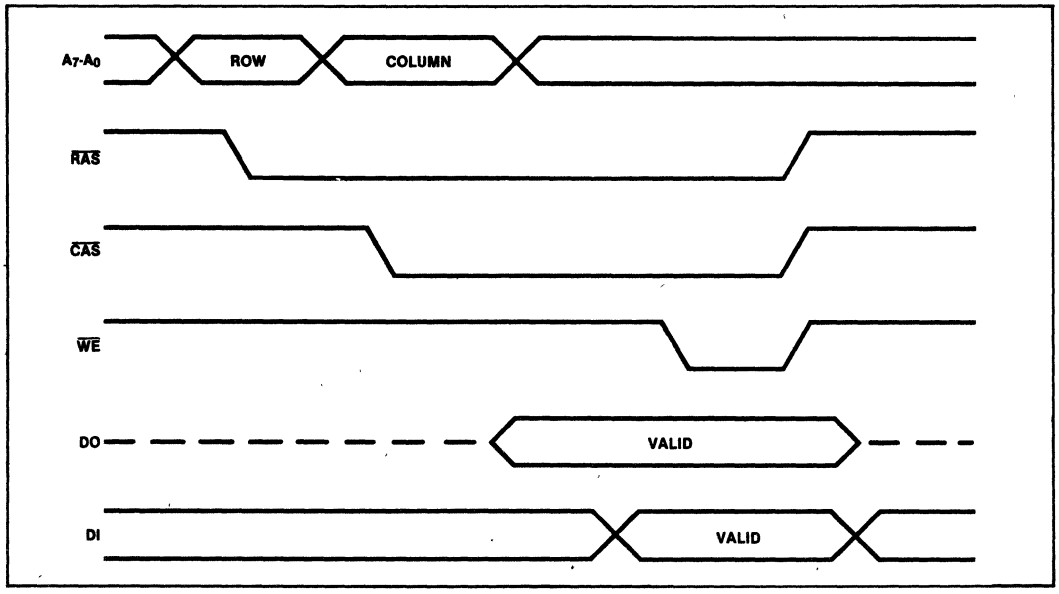


Figure 3. Single-Cycle Read-Modify-Write

used to generate R/$\overline{W}$. This delay, $t_{DELAY\ 2}$, must be long enough to allow the 8206 to generate valid check bits, but not so long that the t_{CWL} spec of the RAM is violated. This is expressed by Equation 3.

(Eq. 3)

		8206				8203		Dynamic RAM
$t_{DELAY\ 1}$	+	TRVSV	≤	$t_{DELAY\ 2}$	≤	t_{CAS}min	−	t_{CWL}
150	+	42	≤	200	≤	260	−	40
		192	≤	200	≤	220	✓	

Unlike other EDC chips, errors in both data *and check bits* are automatically corrected, without programming the chip to a special mode.

Since the 8203 terminates $\overline{CAS}$ to the dynamic RAMs a fixed length of time after the start of a memory cycle, a latch is usually needed to maintain data on the bus until the 8086 completes the read cycle. This is conveniently done by connecting $\overline{XACK}$ from the 8203 to the STB input of the 8206. This latches the read data and check bits using the 8206's internal latches.

The 8086, like all 16-bit microprocessors, is capable of reading and writing single byte data to memory. Since the Hamming code works only on entire words, if you want to write one byte of the word, you have to read the entire word to be modified, do error correction on it, merge the new byte into the old word inside the 8206, generate check bits for the new word, and write the whole word plus check bits into dynamic RAM. A byte write is implemented as a Read-Modify-Write.

Why bother with error correction on the old word? Suppose a bit error had occurred in the half of the old word not to be changed. This old byte would be combined with the new byte, and new check bits would be generated for the whole word, *including the bit in error*. So the bit error now becomes "legitimate"; no error will be detected when this word is read, and the system will crash. You can see why it is important to eliminate this bit error before new check bits are generated. Byte writes are difficult with most EDC chips, but easy with the 8206.

Referring again to Figure 2, the 8206 byte mark inputs ($\overline{BM_0}$, $\overline{BM_1}$), are generated from A0 and $\overline{BHE}$, respectively, of the 8086's address bus, to tell the 8206 which byte is being written. The 8206 performs error correction on the entire word to be modified, but tri-states its DO/WDI pins for the byte to be written; this byte is provided from the data bus by enabling the corresponding 8286 transceiver. The 8206 then generates check bits for the new word.

During a read cycle, $\overline{BM_0}$ and $\overline{BM_1}$ are forced inactive, i.e., the 8206 outputs both bytes even if 8086 is only reading one. This is done since all cycles are implemented as read-modify-writes, so both bytes of data (plus check bits) must be present at the dynamic RAM data input pins to be rewritten during the second phase of the read-modify-write. Only those bytes actually be-

ing read by the 8086 are driven on the data bus by enabling the corresponding 8286 transceiver.

The output enables of the 8286 transceivers ($\overline{OEB0}$, $\overline{OEB1}$) are qualified by the 8086 $\overline{RD}$, $\overline{WR}$ commands and the 8203 $\overline{CS}$. This serves two purposes:

1. It prevents data bus contention during read cycles.
2. It prevents contention between the transceivers and the 8206 DO pins at the beginning of a write cycle.

CONCLUSION

Thanks to the use of a 68-pin package, the 8206 Error Detection and Correction Unit is able to implement an architecture with separate 16 pin input and output busses. The resulting simplification of control requirements allows error correction to be easily added to an 8203 memory subsystem with a minimal amount of interface logic.

intel®

APPLICATION NOTE

AP-167

August 1983

Interfacing the 8207 Dynamic RAM Controller to the iAPX 186

JIM SLEEZER
APPLICATION ENGINEER

ORDER NUMBER: 230809-001
NOVEMBER 1983

INTRODUCTION

Most microprocessor based workstation designs today use large amounts of DRAM for program storage. A drawback to DRAMs is the many critical timings that must be met. This control function could easily equal the area of the DRAM array if implemented with discrete logic.

The VLSI 8207 Advanced Dynamic RAM Controller (ADRC) performs complete DRAM timing and control. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating all of the above functions (plus a dual port and error correcting interfaces) allows the user to realize significant cost savings over discrete logic. For example, comparing the 8207 to the iSBC012B 512K byte RAM board (where the DRAM control is done entirely with TTL), an 8207 design saved board space (3 in^2 vs 10 in^2); required less power (420 ma vs 1220 ma); and generated less heat. Moreover, design time was reduced, and increased margins were achieved due to less skewing of critical timings. This comparison is based on a single port design and did not include the 8207's RAM warm-up, dual-port and error correcting features. If these features were fully implemented, there would be no change to the 8207 figures, listed above, while the TTL figures would easily double.

This Application Note will illustrate an iAPX design with the 8207 controlling the dynamic RAM array. The reader should be familiar with the 8207 data sheet, the 80186 data sheet, and a RAM data sheet*.

DESIGN GOALS

The main objective of this design is for the 80186 to run with no wait states with a Dynamic RAM array. The design uses one port of the 8207. The dual port and error correcting interfaces of the 8207 are covered in separate Application Notes.

The size of the RAM array is 4 banks of 64k RAMs or 512k bytes. The memory is to be interfaced locally to the 80186.

USING THE 8207

The three areas to be considered when designing in the 8207 are:

- 8207 programming logic
- Microprocessor interface
- RAM array

8207 Programming

The 8207 requires up to two 74LS165 shift registers for programming. This design needs one 8 bit shift register, as shown in Figure 1. The 16 bits in the Program Data Word are set as shown in Figure 2. Refresh is done internally, so the REFRQ input must be tied high. The memory commands are iAPX 86 status, so

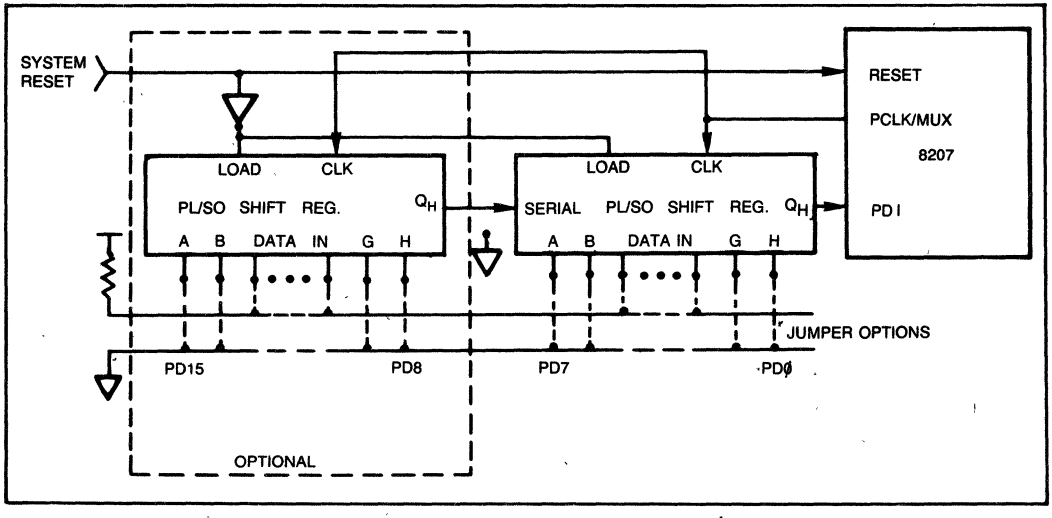
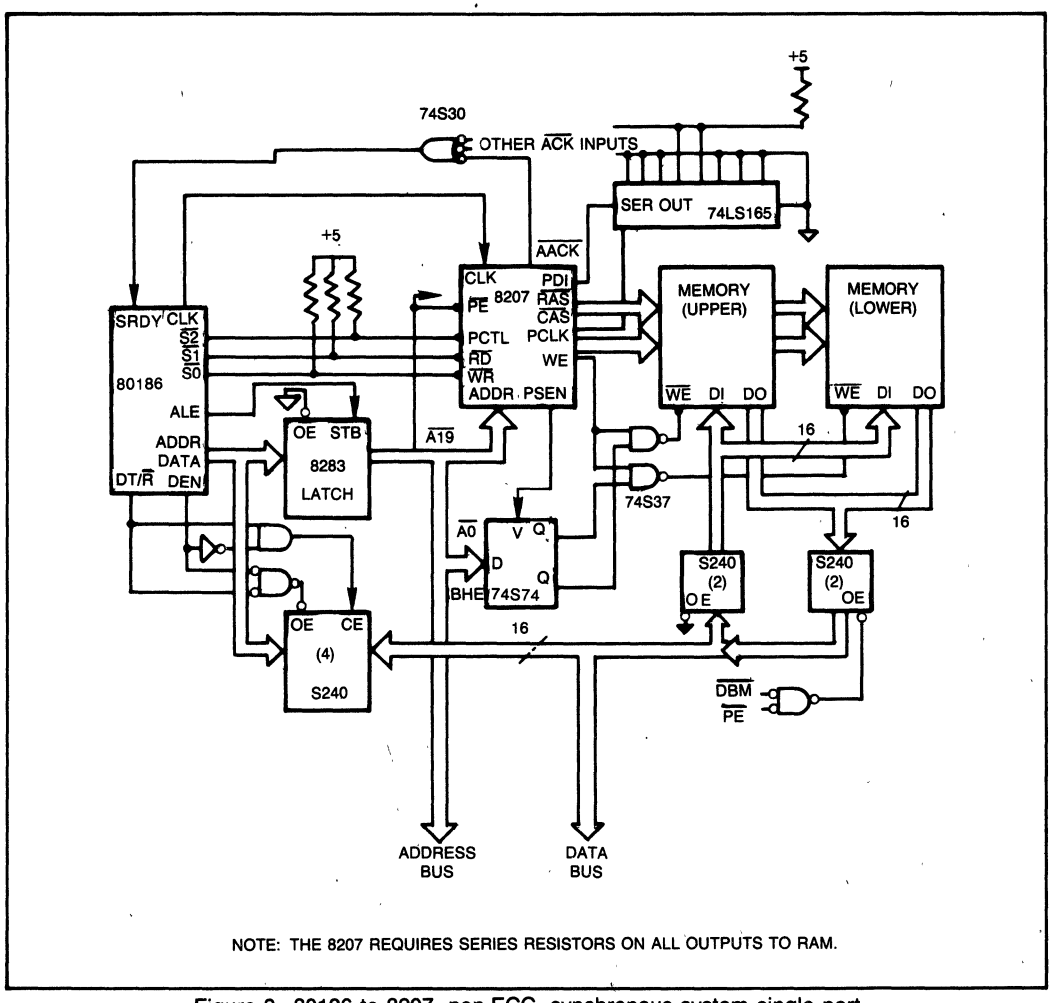


Figure 1. 8207 programming shift registers

*All RAM references in this Application Note are based on Intel's 2164A 64k Dynamic RAM.



NOTE: THE 8207 REQUIRES SERIES RESISTORS ON ALL OUTPUTS TO RAM.

Figure 3. 80186 to 8207, non-ECC, synchronous system single port.

the timing of EAACK will always guarantee 2 clocks of address hold time from RAS.

Acknowledge Setup Time

The margin between the 8207 issuing EAACK and the 80186 ready input for no wait states minus delays from clock edges, logic delays, and setup time is calculated as follows.

1 clock - 8207 TCLAKL max -74S30 tPLH @ 15 pf - 80186 TSRYCL ⩾ 0

125 ns - 35 - 22 - 35 = 33 ns

Read Access Margin

The 8207 starts a memory cycle on the falling clock edge between the 80186's T1 and T2. Data must be valid within 2 clocks. Valid data from the RAMs is based upon the CAS access period minus buffer, clock, setup requirements.

2 TCLCL - 8207 TCLCSL @ 150 pf (t34) - DRAM tCAC - 74S240 propagation delay @ 50 pf - additional bus loading delay (250 pf)(1) - 74S240 delay @ 50 pf - 80186 TDVCL ⩾ 0

250 ns - 122 - 85 - 7 - 7 - 7 - 20 = 2 ns

Write Data Setup and Hold Margin

Data from the processor must be valid when WE is issued by the 8207 to meet the RAM specification tDS (2164A = 0 ns), and then held for a minimum of 30 ns.

(1) 74STTL logic derated by .05 ns/pf. 74STTL buffers (240, 37) derated by .025 ns/pf.

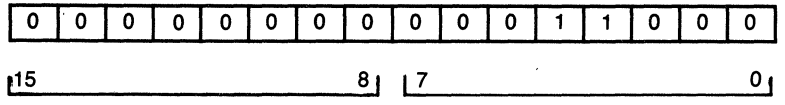


Figure 2. Program data word

the PCTLA input must be high when RESET goes inactive.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

Microprocessor Interface

To achieve no wait states, the 8207 must connect directly to the microprocessor's CLKOUT and status lines. The 8207 Acknowledge (EAACK) must connect to the SRDY input of the 80186.

When the 80186 is reset, it tristates the status lines. The 8207 PCTLA input requires a high to decode the proper memory commands. This is accomplished by using a pull-up resistor or some component that incorporates a pull-up on S2.

The 8207 address inputs are connected directly to the latched/demultiplexed address bus.

RAM Array

The 8207 provides complete control of all RAM timings, warm up cycles, and refresh cycles. All write cycles are "late writes." During write cycles, the data out lines go active. This requires separate data in/out lines in the RAM array.

To operate the 80186 with no wait states, it is necessary to chose sufficiently fast DRAMs. The 150 ns version of the 2164A allows operating the 80186 at 8 MHz, and the 200 ns version up to 7 MHz.

HARDWARE DESIGN

Figure 3 shows a block diagram of the design, and Figure 4 is a timing diagram showing the relationship between the 8207 and the 80186.

8207 Command Setup

Two events must occur for a command to be recognized by the 8207. The 80186 status outputs are sampled by a rising clock edge and Port Enable (PE) is sampled by the next falling clock edge (refer to the Data Sheet wave forms).

The command timing is determined by the period between the status being issued and the first rising clock edge of the 8207, minus setup and delays.

80186 status valid to 8207 rising clock - status from clock delay - 8207 setup to clock ≥ 0

1 TCLCL - 80186 TCHSV max - 8207 TKVCH min ≥ 0

125 ns - 55 - 20 = 50 ns

PE is a chip select for a valid address range. It can be generated from the address bus or from the 80186's programmable memory selects. This design uses an inverted A19. The timing is determined by the interval between the address becoming valid and the falling clock edge, minus setup and delays.

80186 address valid to 8207 falling clock edge - 80186 address from clock delay - 8283 latch delays - 8207 PE setup ≥ 0

1 TCLCL - 80186 TCLAV max - 8283 IVOV @ 300 pf - 8207 TPEVCL ≥ 0

125 ns - 44 - 22 - 30 = 29 ns

The hold times are 0 ns and are met.

Address Setup

For an 80186 design, the 8207 requires the address to be stable before RAS goes active, and to remain stable for 2 clocks. Unused 8207 address inputs should be tied to Vcc.

tASR is a RAM specification. If it is greater than zero, this must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued and RAS going active, minus appropriate delays.

80186 address valid to 8207 RAS active - 80186 address from clock delay - bus delays - (8207 setup + RAM t_{ASR}) ≥ 0

TCLCL + 8207 TCLRSL min @ 150 pf(1) - 80186 TCLAV max - 8283 IVOV max @ 300 pf - (8207 TAVCL min + DRAM t_{ASR}) ≥ 0

125 ns + 0 - 44 - 22 - (35 + 0) = 24 ns

The address hold time of 2 clocks + 0 ns is always met, since the addresses are latched by the 8282/3. Even when the processor is in wait states (for refresh),

(1) Not specified—use 0 ns.

TCLCL + TCLCH + 8207 TCLW min(1) + 74S37 delay tPHL min @ 50 pf + additional loading (142 pf) - 80186 TCVCTV - 74S240 tPZL - bus delays (250 pf) - 74S240 delay - 2164A tDS ⩾ 0

125 + 62.5 + 0 + 6.5 + 3.5 - 70 - 15 - 7 - 7 - 0 = 98.5 ns

The hold time, tDH, is from WE going low to the 80186 DEN going high plus buffer delays minus WE from clock delays.

TCLCL - 80186 TCVCTX min + 74S32 tPD(2) min + 74S240 tPHZ (min)(2) + 250 pf bus delays + 74S240 propagation delay min - 8207 TCLW max - 74S37 tPHL @ 50 pf - 142 pf loading delays - DRAM tDH ⩾ 0

62.5 ns + 10 + 2 + 3 + 7 + 3.5 - 35 - 3.5 - 30 = 19.5 ns

All margins are actually better by about 10-20 ns. No improvement in timing was allowed for lower capacitive loads when additional buffers are used (i.e. the 80186 address out delay is at 200 pf, but the 8283 latch only loads these lines with about 20 pf).

SUMMARY

The 8207 supports the 80186 microprocessor running with no wait states. The 8207 interfaces easily between the microprocessor and dynamic RAM. There are no difficult timings to be resolved by the designer using external logic.

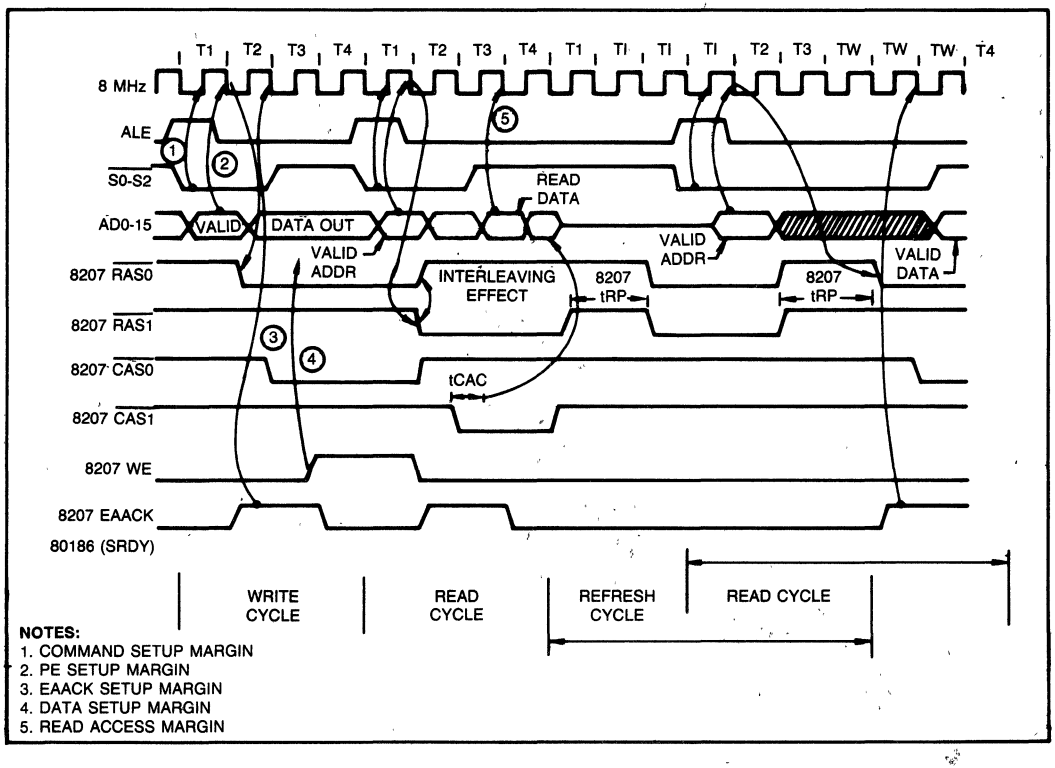


Figure 4. 8207/80186 timing relationship

(1) Not specified, use 0 ns.
(2) Not specified, use one half of typical value.

intel

APPLICATION NOTE

AP-168

August 1983

Interfacing the 8207 Advanced Dynamic RAM Controller to the iAPX 286

JIM SLEEZER
APPLICATION ENGINEER

ORDER NUMBER: 230862-001
NOVEMBER 1983

INTRODUCTION

The 80286 high speed microprocessor pushes microprocessor based systems to new performance levels. However, its high speed bus requires special design considerations to utilize that performance. Interfacing the 80286 to a dynamic RAM array require many timings to be analyzed, refresh cycle effects on bus timing examined, minimum and maximum signal widths noted, and the list continues.

The 8207 Advanced Dynamic RAM Controller was specifically designed to solve all interfacing issues for the 80286, provide complete control and timing for the DRAM array, plus achieve optimum system performance. This includes the normal RAM 8 warm-up cycles, various refresh cycles and frequencies, address multiplexing, and address strobe timings. The 8207 Dynamic RAM Controller's system interface and RAM timing and control are programmable to permit it to be used in most applications.

Integrating these functions (plus dual port and error correcting interfaces) allows the user to realize significant savings in both engineering design time, PC board space and product cost. For example, in comparing the 8207 to the ISBC012B 512k byte RAM board (where the DRAM timing and control is done entirely with TTL), the 8207 design saved board space (3 in^2 vs 10 in^2); used less power (420 ma vs 1220 ma); reduced the design time; and increased margins due to less skewing of timings. The comparison is based upon a single port 8207 design and does not include its RAM warm-up, dual port, error correcting, and error scrubbing or RAM interleaving features.

This Application Note will detail an 80286 and 8207 design. The reader should have read the 8207 and the 80286 data sheets, a DRAM data sheet*, and have them available for reference.

DESIGN GOALS

The main objective of this design is to run the RAM array without wait states, to maximize the 80286's performance, and to use as little board space as possible. The 80286 will interface synchronously to Port A of the 8207 and the 8207 will control 512k bytes of RAM (4 banks using 64k DRAMs). The dual port and error correcting features of the 8207 are covered in separate Application Notes.

8207 INTERFACE

The 8207 Memory design can be subdivided into three sections:

- Programming the 8207.
- The 80286/8207 interface.
- The Dynamic RAM array.

Programming the 8207

The RAM timing is configured via the 16 bit program word that the 8207 shifts-in when reset. This can require two 74LS165 shift registers to provide complete DRAM configurability. The 8207 defaults to the configuration shown in Table 1 when PDI is connected to ground. This design does not need the flexibility the shift registers would allow since standard 8207/80286 clock frequencies, DRAM speeds and refresh rates are used. Table 1 details the 8207/80286 configuration and Table 10 in the Data Sheet identifies "CO" as the configuration of the 8207 all timings will be referenced to (80286 mode at 16 MHz using fast RAMs = CO).

Table 1. Default Non-ECC programming, PD1 pin (57) tied to ground.

Port A is Synchronous (EAACKA and XACKA)
Port B is Asynchronous (LAACKB and XACKB)
Fast-cycle Processor Interface (10 or 16 MHz)
Fast RAM 100/120 ns RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

The 8207 will accept 80286 status inputs when the PCTLA pin is sampled low at reset. This pin is not necessary for an 80286 design (besides programming) and is tied to ground.

Refresh is the final option to be programmed. If the Refresh pin is sampled high at reset, an internal timer

*All RAM references in this Application Note are based upon Intel's CMOS 51C64-12 64k Dynamic RAM. Any DRAM with similar timings will function. Refer to section 4.4.

is enabled, and if low at reset, this timer is disabled. The first method is the easiest to implement, so the RFRQ pin is tied to Vcc.

The differential reset circuit shown in the Data Sheet is necessary only to ensure that memory commands are not received by the 8207 when Port A is changed from synchronous to asynchronous (vice versa for Port B). This design keeps Port A synchronous so no differential reset circuit is needed.

RAM Array

The 8207 completely controls all RAM timings, warm-up cycles, and refresh cycles. To determine if a particular RAM will work with the 8207, calculate the margins provided by the 8207 (Table 15, 16—8207 Data Sheet) and ensure they are greater than the RAM requirement. An additional consideration is the access times of the RAMs. The access time of the system is dependent upon the number of data buffers between the 80286 and the DRAMs. To operate the 80286 at zero wait states requires access times of 100-120 ns. Slower RAMs can be used (150 ns) by either adding a wait state (programming the 8207 for "C1") or reducing the clock frequency (to 14.9 MHz approximately and maintaining the CO configuration.)

All write cycles are "late writes" and the data out lines of the RAM will go active. This will require separate data in and out lines in the RAM array. Another consideration for the RAM array is the proper layout of the RAM, and impedance matching resistors on the 8207 outputs. Proper layout is covered in Intel's RAM Data Sheets and Application Notes.

Microprocessor Array

To achieve no wait state operation, the 8207's clock input must be connected to the 80286's clock input. The EAACK (early acknowledge) output of the 8207 must connect to the SRDY input of the 82284. The 8207's address inputs connect directly to the 80286 address outputs and the addresses are latched internally. This latch is strobed by an internal signal with the same timing as LEN (which is for dual port 80286 designs). Figure 2 shows the timing relationship between LEN and the 80286.

LEN will fall from high to low, which latches the bus address internally, when a valid command is received. LEN can go high in two clock cycles if the RAM cycle started (RAS going low) at the same time LEN went low. If the 8207 is doing a refresh cycle, the 80286 will be put into wait states until the memory cycle can start. LEN will then go high two clocks after RAS starts, since addresses are no longer needed for the current RAM cycle. Thus the low period of LEN could be much longer than listed in the Data Sheet.

DESIGNING THE HARDWARE

Figure 1 shows a detailed block diagram of the design and Figure 2 shows the timing relationship between the 8207 and the 80286.

The following analysis of six parameters will confirm that the design will work. These six system parameters are generally considered the most important in any microprocessor—Dynamic RAM design.

8207 Command Setup Margin

Two events must occur for the 8207 to start a memory cycle. Either RD or WR active (low) and PE must be low when the 8207 samples these pins on a falling clock edge. If PE is not valid at the same clock edge that samples RD or WR active, the memory cycle will be aborted and no acknowledge will be issed.

The command setup time is based upon the status being valid at the first falling clock edge.

80286 status valid to 8207 falling clock − 80286 status from clock delay − 8207 command setup to clock ⩽ 0

TCLCL − 80286 t12 (max) − 8207 TKVCL (min) ⩽ 0

62.5 − 40ns − 20ns = 2.5ns

PE is decoded from the address bus and must be set up to the same falling clock edge that recognizes the RD, WR inputs. This margin is determined from the clock edge that issues the address and the clock edge that will recognize RD or WR, minus decoding logic delays.

There are 2 clocks between addresses being issued by the 80286 and PE being sampled by the 8207. Then the 80286 address delay from the clock edge and decoding logic delays are subtracted from this interval. This margin must be greater than 0.

2TCLCL − 80286 t13 (max) − 8207 TPEVCL (min) ⩽ 0

125 − 60 − 30 = 35ns

The address decode logic must use no more than 35 ns (and less is better). Figure 3 shows an easy implementation which uses a maximum of 12 ns.

The 8207 requires a zero ns hold time and is always met.

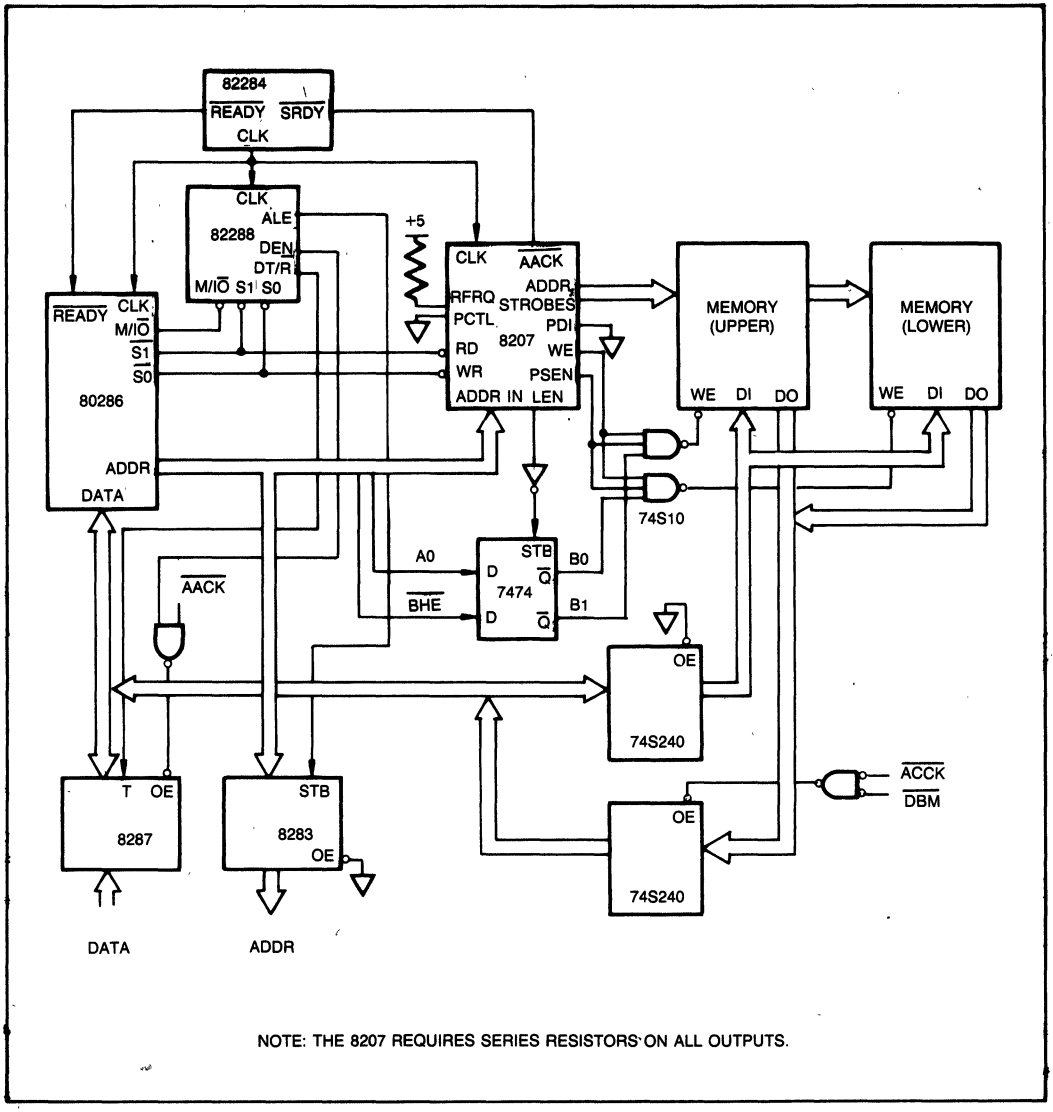


Figure 1. 80286 to 8207, non-ECC, Synchronous System Single Port

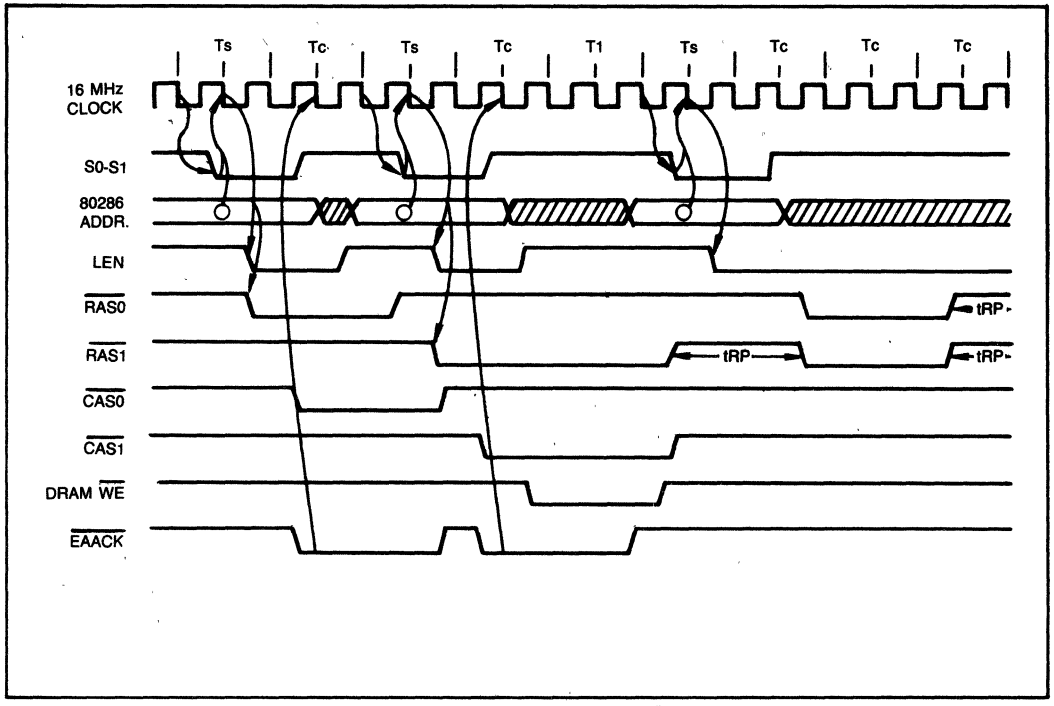


Figure 2. 80286/8207 Timing—"CO".

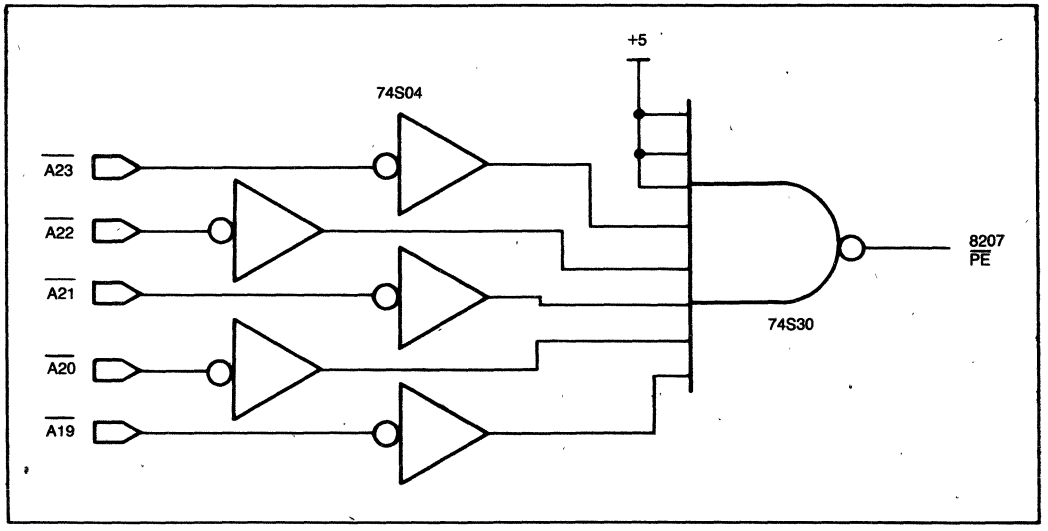


Figure 3. Address Decode Logic

Address Setup Margin

The 8207 must have stable addresses up to two clocks after RAS goes active. This is of no concern to the user, since LEN latches the address internally and will not admit a new address until two clocks after RAS goes active.

Addresses must be stable at least 35 ns (tAVCL) before RAS goes active to allow for propagation delays through the 8207, if a RAM cycle is not delayed by the 8207.

tASR is a RAM specification. If it is greater than zero, tASR must be added to the address setup time of the 8207. Address setup is the interval between addresses being issued, by the 80286, and RAS going active, minus appropriate delays.

The margin is determined from the number of clocks between addresses being issued from the 80286 to RAS going active. Exactly when RAS goes active is unimportant, since here we are interested only in the clock edge.

2TCLCL − 80286 t13 (max) − 8207 TAVCL (min) $\leqslant$ 0

125 − 60ns − 35ns = 35ns

Acknowledge Setup Margin

The 8207 acknowledge (EAACK) can be issued at any point in the 80286 bus cycle (end of ϕ1 or ϕ2 of Ts or Tc). If EAACK is issued at the end of ϕ2 (Ts or Tc), the 80286 will complete the current bus cycle. If EAACK is issued at the end of ϕ1 of Tc, the 82284 will not generate READY to the 80286 in time to end the current bus cycle. A new Tc would then be generated and EAACK would now be sampled in time to terminate the bus cycle. EAACK is 3 clocks long in order to meet setup and hold times for either condition.

We need the margin between the 8207 issuing EAACK and the 82284 needing it. Figure 4, shows a worst case example.

TCLCL − 8207 TCLAKL max − 82284 t11 $\leqslant$ 0

62.5 − 35 − 15 = 12.5ns

Read Access Margin

The 8207 will typically start a memory cycle (i.e. RAS goes low) at the end of ϕ1 of Ts. But if the start of a memory cycle is delayed (by a refresh cycle for instance), then RAS will be delayed. In the first case,

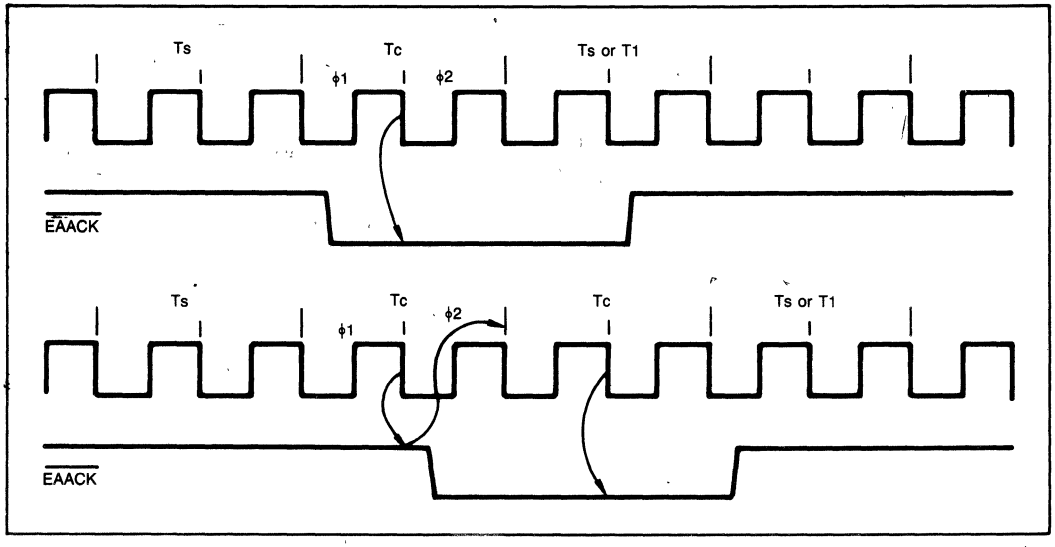


Figure 4. Acknowledge to the 82284

this represents 3 clocks and the second case could require 4 clocks to meet the data setup requirements of the 80286. In either case, data must be valid at the end of Tc. The 8207 holds CAS active long enough to ensure valid data is received by the 80286 in either case.

DRAMs specify two access times, RAS access (tRAC) and CAS access (tCAC) Both access periods must be calculated and the one with the least margin used. Also the number of data buffers should be kept to a minimum. Too many buffers would require either faster (more expensive) DRAMs, or a reduction in the performance of the CPU (by adding wait states).

RAS Access Margin

3TCLCL − 8207 TCLRSL max @ 150 pf − DRAM tRAC − 74S240 propagation delay max @ 50 pf − 80286 t8 ⩽ 0

187.5 − 35 − 120 − 7 − 10 = 15.5ns

CAS Access Margin

2TCLCL − 8207 TCLCSL max @ 150 pf − DRAM tCAA (or tCAC − 74S240 tplh max @ 50 pf − 80286 t8 ⩽ 0

125 − 35 − 60 − 7 − 10 = 13ns

By solving each equation for tRAC and tCAC, the speed requirement of the RAM can be determined.

DRAM tRAC = 3 TCLCL − 8207 TCLRSL − 74S240 tplh − 80286 t8 = 135.5ns

DRAM tCAC = 2 TCLCL − 8207 TCLCSL − 74S240 tplh − 80286 t8 = 73ns

So any DRAM that has a RAS access period less than 135 ns, a CAS access period less than 73 ns, and meets all requirements in the DRAM Interface Timing (Table 15, 16—8207 Data Sheet), will work.

Write Data Setup and Hold Margin

Write data from the processor must be valid when the 8207 issues WE to meet the DRAM specification tDS and then held to meet the tDH requirement. Some write cycles will be byte writes and the information to determine which byte is decoded from A0 and BHE/. Since the 80286's address bus is pipelined, these two signals can change before the RAM cycle starts, hence they must be latched by LEN. PSEN is used in the WE term to shorten the WE pulse. Its use is not essential.

Data must be set up to the falling edge of WE, since WE occurs after CAS. The 2 clocks between valid write data and WE going active (at the RAM's) minus propagation delays determines the margin.

2 TCLCL − 80286 t14 (max) @ 100 pf − 74S240 tplh + 8207 TCLW (min)[1] + 74S10 tphl @ 192 pf[2] − DRAM tDS = 0

125 − 50 − 7 + 0 + 14 − 0 = 82ns

The timing of the 8207's acknowledge is such that data will be kept valid by the 80286, for more than two clocks after WE goes active. This easily meets all RAM tDH specifications.

SUMMARY

The 8207 complements the 80286's performance and high integration with its own performance, integration and ease of use. No critical timings or logic design has been left to the designer. The 80286/8207 combination allows users to realize maximum performance from their simpler design.

1. Not specified. Assume no delay for worst case analysis.
2. STTL derated by .05ns/pf.

October 1982

Dynamic-RAM Controller Orchestrates Memory Systems

Jim Nadir
Mel Bazes
Intel Corporation
Santa Clara, California

OCTOBER 1982
ORDER NUMBER: 210758-001

Dynamic-RAM controller orchestrates memory systems

Up to 88 chips take their cues from an n-channel MOS IC that both housekeeps and supports error-corrected dual-port memories

by Jim Nadir and Mel Bazes, *Intel Corp., Santa Clara, Calif.*

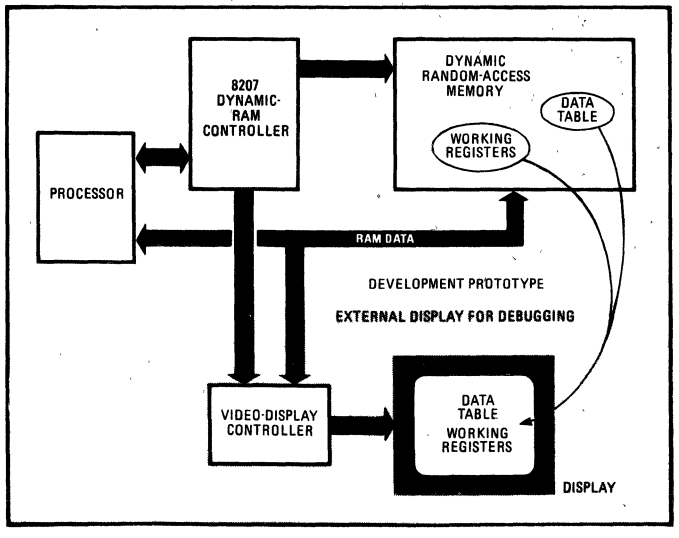
☐ Designing a dynamic–random-access-memory system means balancing the goals of high performance, reliability, and versatility against the often contrary aims of economy, simplicity, and compactness. In the last five or so years, the advent of dynamic-RAM controller chips relieved designers of some of the onus of tending to the needs of dynamic chips: standard supportive integrated circuits brought together the counters, timers, multiplexers, and other elements needed.

But controllers diverged into two types. One bought the high performance to ride with fast memory systems at the expense of functionality, while the other took on more and more functions to do a complete but slower job. The 8207—an advanced dynamic-RAM controller—blunts the horns of that dilemma and also solves a variety of less severe design problems.

A dynamic-RAM controller is charged with making a dynamic memory system appear static to the host processor. At a minimum, therefore, the controller takes over refreshing the memory chips, multiplexing the row and column addresses, generating control signals, timing the precharge period, and signaling the processor when data is available or no longer needed. But, beyond those local housekeeping chores, the controller can also go a long way to solving more global design problems, like sharing memory between two processors, not to mention detecting and correcting errors.

To realize this potential for a highly integrated solution, the 8207 has a dual-port interface and, when used with the 8206 error-checking and -correction unit, ensures data integrity in large dynamic-RAM systems. In addition to doing the jobs of refreshing, address multiplexing, and control timing, the unit supports memory-bank interleaving for pipelined accesses, overlaying RAM and read-only-memory locations, and initializing RAM.

The exact implementation of most of these functions is programmable, letting designers tailor their systems in detail. Systems containing up to 88 dynamic-RAM chips—whether 16-, 64-, or 256-K versions—in one, two, or four banks need only a single 8207 and no external buffering. Attesting to the high performance claimed, the 8207 mates dynamic RAMs having 100-nanosecond access times to the iAPX-286 processor operating at 8 megahertz without introducing any wait states.



1. Window on a micro. One use for a dual-port memory shared by independent processors is the development system shown. Adding a video display to the prototype itself gives a window on the system memory.

To achieve that speed and include all those functions, the 8207 relies on a dense, high-speed n-channel MOS process (H-MOS II) and requires a chip some 230 by 200 mils in area. To meet the rigors of operation with even faster processors, novel logic and integrated-circuit designs are employed. Replacing the two-phase logic common in n-MOS ICs, single-phase edge-triggered logic simplifies logic and circuit design, precludes problems of clock-pulse overlap, and reduces the sensitivity to clock high and low times. Voltage-controlled capacitive loads form the delay elements that time critical output pulses, such as the address strobes, and compensate the output-switching delays for variations in power-supply voltage, temperature, and processing.

A low 20-ns setup time for input signals is achieved by cutting the RC delay of input-protection devices and moving the TTL-to-MOS signal buffering from the input pads to the pulse generators. A short 35-ns delay from input to output switching is achieved by triggering the output generators directly from the external clock, saving a buffer delay time. With the resulting high-speed performance and a high level of integration, the 8207 successfully attacks the stringent requirements of today's memory systems.

One system feature gaining popularity currently is the use of multiple processors operating on shared data to obtain higher performances and reliability. For example, a separate processor dedicated to input/output tasks frees the main processor for full-time data processing. Alternatively, multiple main processors can execute different tasks simultaneously. In all such cases, sharing a common memory space among the cooperating processors is the key to effective operation.

Unfortunately, when more than one processor accesses shared memory through a single bus, the limited bus bandwidth and the time spent in exchanging bus control slow down data transfers. Dual-port memory systems overcome this limitation by giving two processors access to a common memory through two independent buses. The 8207 includes a dual-port interface to simplify the design of shared memory systems.

Two-port memories can be used with multiprocessing or multitasking architectures. In the former, independent processors run independent programs, sharing only a common memory. Multitasking processors cooperate on different parts of the same task.

An example of a multiprocessing architecture is the dynamic video display (Fig. 1) that provides a window on a processor's memory. Centering the display over a data table, for example, immediately reveals how program execution affects the data, which aids in debugging programs. If a microcomputer is implemented with a dual-port memory—the second port for a dynamic video display—then the prototype itself can serve as a development and debugging system, reverting to single-port operation in the final version.

A dual-port architecture in a multitasking environment, on the other hand, adds a margin of safety to a shared-resource bus, such as Intel's Multibus. Although one of the biggest benefits of such a bus is the sharing of expensive peripherals among several users' programs, an intimidating problem is that a single program gone haywire can easily corrupt the entire system. A two-port memory, properly configured, circumvents this occurrence. Because each port has its own address, data, and control lines, problems on one side are confined by hardware to that side.

Port of call

As a general rule for multitasking architectures, one port of a two-port memory operates in a local environment, and the other port runs remotely, off the expandable shared-resource bus. The local processor is likely to require a synchronous port to reap the benefit of higher performance. Remote buses, in contrast, are usually configured asynchronously. Unless programmed other-

Dynamic-RAM controllers get in step

Synchronous and asynchronous signals have different requirements for interfacing with a controller. The terms synchronous and asynchronous are conventionally applied to dynamic random-access memory depending on whether it exists in a local or a remote environment, respectively. However, they more properly characterize the dynamic-RAM controllers, for the RAMs themselves need no clocks—the only restrictions as to the start of a memory access cycle involve ensuring that the refresh and precharge requirements are satisfied.

Because the controller decides both when to refresh and whether or not precharge and other timing requirements have been met, it does need a clock. Incoming commands can either always arrive with a fixed relationship to the controller's clock or have no particular relationship to it. The former are, of course, synchronous operations, the latter asynchronous.

The major difference between an asynchronous and a synchronous controller (or port of a controller, in the case of the dual-port 8207) is that the asynchronous controller must first synchronize the incoming commands to its own internal clock. From that point on, the asynchronous controller looks just like a synchronous device.

Whereas various techniques for synchronization are available off chip, on-chip synchronization is restricted to the resolution and sampling of states of a flip-flop. The incoming command is clocked into a resolving flip-flop. After a predetermined time, a sampling flip-flop reads the state of the resolving flip-flop, thereby synchronizing the command. Assuming that both flip-flops are triggered on the same edge of the controller's internal clock, the fastest that an asynchronous signal can be synchronized is one clock period. The slowest synchronization takes two clock periods; on the average, getting the signals in step takes one and a half clock cycles.

Because the processor typically requires four or fewer clock periods to complete a cycle, adding a cycle and a half for synchronizing increases the access time by approximately 25%. Synchronous controllers are therefore always preferred when the environment permits them, and local environments, such as single-board computers, generally do so.

wise, the 8207 configures one port synchronously, and the other asynchronously. For specific applications, both ports may be programmed as either synchronous or asynchronous (see "Dynamic-RAM controllers get in step," p. 129).

Whether the ports are programmed for synchronous or asynchronous operation, some mechanism must decide which processor will gain access to memory when both request it almost simultaneously. That mechanism consists of arbitration logic that controls access and always leaves one port selected. When a port is selected, its associated control and interface signals are passed directly to the RAM timing logic by the command multiplexer (Fig. 2). Both ports' command and control lines, after being synchronized, go into both the command multiplexer and the arbitration logic.

However, the arbitration logic enables the command multiplexer to pass only commands that appear at the selected port. At the same time as a command appears at a selected port, arbitration logic initiates the cycle-control logic that completes the timing of the RAM cycle that ensues. If a command appears on the unselected port, it will not get through the multiplexer to initiate a RAM cycle but will instead wait in the status-command decoder until the current command is completed, at which time the command multiplexer switches to the unselected port. The arbitration logic will then service this queued access request by starting a new cycle.

The arbitration logic examines all port requests, including the internal refresh port. The refresh-request port is subject to arbitration like the other two ports, except that it is always assigned a higher priority than an unselected external access port. Thus, refreshing can be delayed, at most, one RAM cycle.

While the current RAM cycle is running, the arbiter determines the next cycle to be initiated. Thus, the arbitration time of two or more simultaneous port requests is hidden by the memory cycle time. In other words, in cases where both a selected and an unselected port request access simultaneously, the arbitration time for the unselected port does not extend that port's access time, which is delayed by one memory cycle anyway. Only when an unselected port requests a free memory does the arbitration time slow access, because then the command must pass through the arbitration logic before a RAM cycle can be initiated. To minimize such delays in most cases, there are two arbitration algorithms to be selected by the user.

The first algorithm, intended for multiprocessing environments, automatically returns the arbiter to a designated preferred port, generally the higher-performance, synchronous port. Thus any command on the selected port generally has immediate access, whereas any command arriving at the unselected port must wait.

The second, or last-accessed-port, algorithm, which is applicable in multitasking environments, leaves the most recently accessed port as the selected port. This algorithm optimizes port selection for task passing in a multitasking environment. In task passing, the host processor sends a task to an execution processor; until the task is received, the execution processor seldom accesses memory. Conversely, once the task is passed, the host processor seldom accesses memory until the task is completed. Thus, the ports are used in spurts.

Because timely refreshing is needed to preserve dynamic-RAM data, a refresh request is always serviced on the next available cycle. The refresh algorithm, however, may be selected by the user. The options available are: no refresh, user-generated single refresh, automatic refresh, or user-generated burst refresh.

No refresh would be selected for applications like bit-mapped–video displays, where continuous, sequential access of all RAM locations itself refreshes every cell periodically. User-generated refresh modes allow the designer greater control over power dissipation, for example, in large memory systems. Automatic refreshing, in which the controller itself times the refresh interval and initiates the operation, lets the designer ignore the refresh requirements entirely. As mentioned, the refresh requests are subject to arbitration just like other access requests. However, once a burst refresh is selected, it remains active until completed.

Cleaning up errors

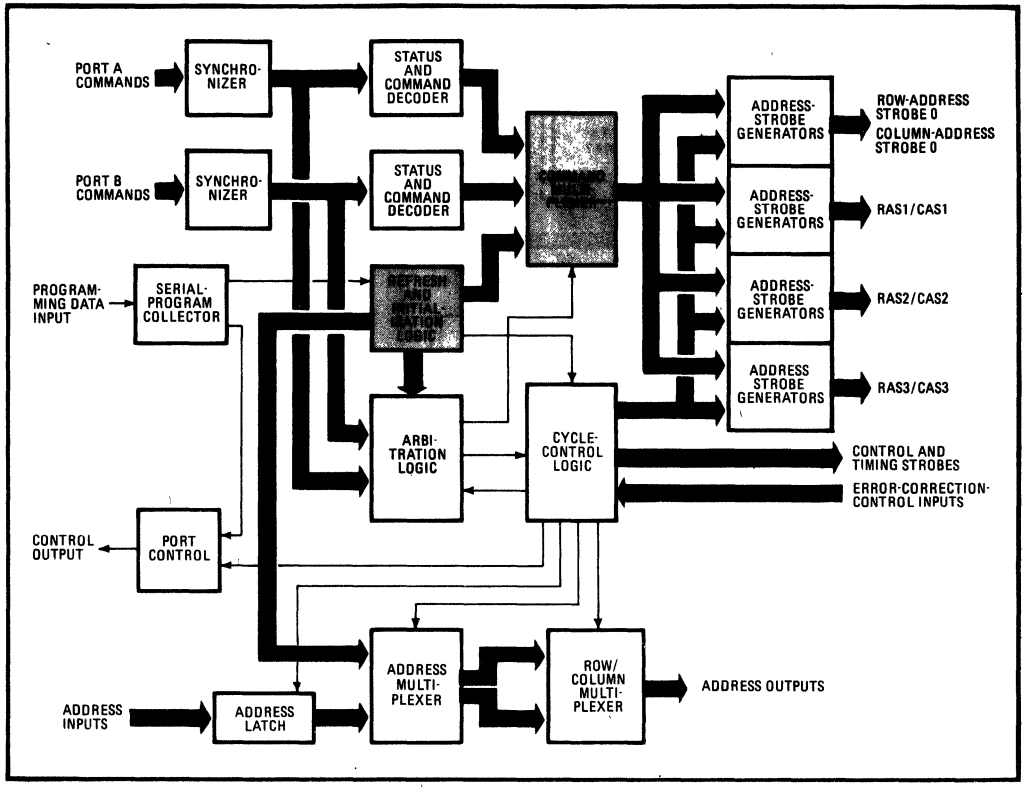
Ensuring data integrity is a major concern in large dynamic-RAM systems, particularly because of their susceptibility to soft errors caused by alpha-particle radiation. Various parity encoding techniques have been developed to detect and correct memory-word errors [*Electronics*, June 2, 1982, p. 153]. The parity bits, called check bits when used for correction as well as detection, are stored in the memory array along with their associated data word. When the data is read, the check bits are regenerated and compared with the stored check bits. If an error exists, whether in the retrieved check bits or in the retrieved data word, the result of the comparison—called the syndrome—gives the location in the group of the bit in error.

Two drawbacks surface in the design of any memory system that is to be protected by error-correction circuitry. First, the memory-word width must be increased to store the check bits; second, extra time must be allotted for the error-correction circuitry to generate the check bits on write cycles, plus more time to regenerate and compare the check bits on read cycles. The 8207 provides several ways to minimize both problems.

Error-correction schemes require a smaller proportion of check bits to protect wider memory words. For example, an 8-bit word needs 5 check bits, for a 63% increase in memory. Put the other way around, 38% of the available memory would be dedicated to the check bits. Six check bits are required to protect a 16-bit data word—only a 27% overhead. Clearly, the wider the memory array, the more economical the error correction.

The 38% overhead necessary to protect such 8-bit–bus machines as the 8088 or 8085 makes error correction an unattractive proposition. However, if the memory width could be doubled, with the 8088 accessing only half a word at a time, the overhead would drop to 27%.

Reading a double-width word, checking for soft errors, and then sending the desired portion of the word to the processor presents no major problems, unlike writing to such an array. The check bits cannot be calculated from only a portion of the word—they must be calculated for



2. Arbiter's labor. Two external ports plus the internal refresh port can request access to the memory system at once. Arbitration logic decides which to service, based on programmable algorithms. High-speed logic design cuts the delay from input to output switching to 55 ns.

the entire word at once. Whenever the processor writes a partial word to memory, it must first read the entire word, check it, substitute for that portion of the word to be rewritten, and recalculate the check bits. Only then can the entire word be written to memory. The 8207, working in conjunction with the 8206 error-checking and -correction unit, contains mechanisms to expedite this potentially arduous process.

Whenever the 8207 performs a partial-write cycle, it initiates a read-modify-write cycle wherein the entire memory word is first read and latched into the 8206 (Fig. 3). After the retrieved data has been verified as correct, new data is supplied to the RAM, half from the processor and half from the 8206, which also generates the check bits for the entire new word.

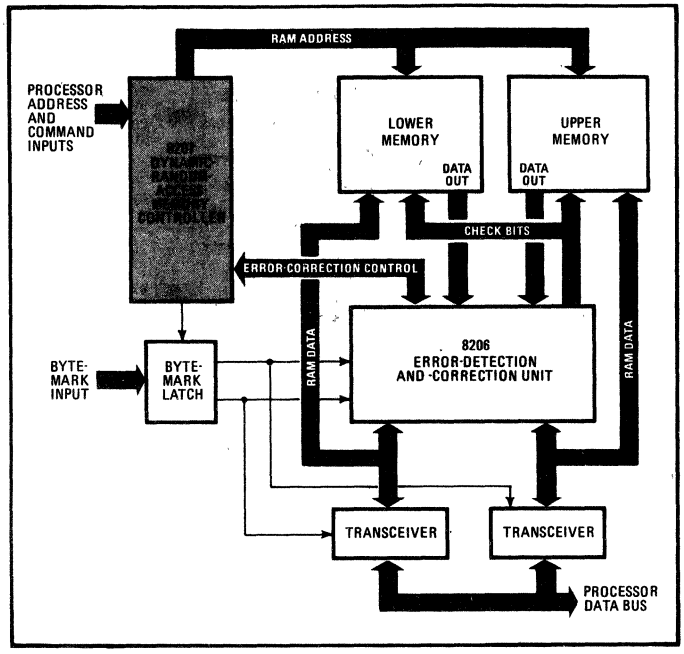
Control signals—called byte marks—specify which portion of the new data word is coming from the processor and which from the 8206. The byte marks determine whether the processor or the 8206 drives the RAM data bus—for example, if the 8206 is driving one portion of the data bus, the processor is prevented from driving the same portion. The byte-mark signals simply disable the appropriate transceivers. If, on the other hand, the processor is driving a portion of the RAM data bus, the byte marks change the 8206 data outputs to inputs, allowing the 8206 to read the data from the processor and calculate new check bits.

The ability of the 8207 to handle memories organized as one, two, or four banks allows tradeoffs between the cost and performance of an error-correction system. For maximum performance, memory would be organized in four banks, each 16 bits wide. In applications requiring error correction, but where maximum performance is not critical, concatenation of RAM banks into two banks of 32-bit words, or even one bank of 64-bit words, can make error correction very economical.

Holding to high performance

Even though the cost of error correction has thus been reduced to where it becomes an attractive solution, the problem remains of minimizing performance degradation. Tackling that challenge depends on the particulars of the configuration, such as whether the memory is to be used with a high-performance local processor, as system memory on a shared-resource bus, or is to be shared between a local high-performance processor and a shared-resource bus.

The method chosen to handle errors depends on the type of bus. Intel's Multibus is the kind that requires data to be valid prior to the issuance of a transfer-



3. Teamwork. The 8206 error-correction chip joins forces with the random-access-memory controller so that an 8-bit–bus processor may utilize the 16-bit-wide memory that is more economical for error-correction schemes. Byte marks configure the data buses for partial-word transfers.

acknowledge signal, in contrast to the local buses of the iAPX-86, -186, and -286 processors. A local bus will usually be synchronous, with a single processor or coprocessor group attached to it; the processor characteristics are known, as is the processor's response to a transfer-acknowledge signal.

With Multibus and other shared-resource buses, the processor types that will eventually be connected are not known in advance, and the buses themselves are generally asynchronous. Hence the time between the transfer-acknowledge signal and data becoming valid is not known. Therefore, the rule with such buses is to acknowledge a transfer only when data is valid. (On some asynchronous buses, the acknowledgment is issued earlier to compensate for synchronization delay at the receiving processor.)

Two basic configurations for checking and correcting errors derive from these system considerations and the fact that it takes longer to correct data than to detect an error. One is for buses that connect to processors and coprocessors receiving a transfer acknowledge prior to data becoming valid, and the other for buses that connect to processors receiving a transfer acknowledge after data is valid. Both configurations are supported by the 8206–8207 team.

Buses among the former type of processors always get corrected data from the 8206, whether an error exists or not, and will carry a transfer acknowledge from the 8207 before data becomes valid on the bus. Though this means data is delayed for error correction on every transaction, the extra delay is immaterial, since it is hidden behind the processor's response time to the transfer-acknowledge signal. By the time the processor requires data, it is already corrected and on the bus. As a result, system performance is not degraded at all because of single-bit errors.

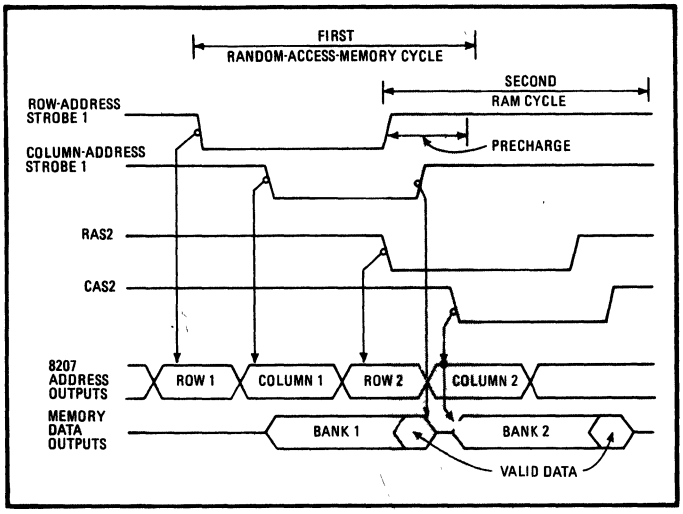
For buses among processors that receive the transfer acknowledge after the data is valid, the 8206 always checks for errors but does not routinely correct data. In this mode, RAM data passes through faster, because the 8207 will issue an acknowledgment sooner. If, however, an error is found, the 8207 will lengthen the cycle, command the 8206 to correct the data, and delay the transfer-acknowledge signal until the corrected data can be placed on the bus. For those buses with an acknowledge-synchronization delay, the 8207 can be programmed to issue the acknowledgment earlier to compensate for the delay.

Power-up problems

Another problem with memories protected by ECC circuits crops up when the power is turned on. At power-up, the data stored in memory is completely random; any attempt to read or perform a partial write will be aborted because the check bits will indicate multiple, and therefore uncorrectable, errors. For processors whose word width is the same as that of the memory array, the processor could simply initialize the entire memory array, taking some additional time and software. For memories whose word width is greater than that of the processor, however, initialization of the memory is not possible unless the error-checking or -correction circuitry is disabled by hardware, for example, by gating off the error flags.

The 8207 is equipped to deal with the initialization problem by itself. At system reset, the 8207 performs

4. Interleaving. Overlapping accesses to different banks increases memory throughput. Once the column-address hold time is satisfied, the 8207 starts a second cycle, pulling the second row-address strobe low.



eight cycles on all banks at once to warm up the dynamic RAMs, a typical RAM requirement for stable operation. The chip then individually initializes all memory locations to 0, adding the proper check bits. Though all memory banks could be initialized in parallel, that would require more power than any other memory operation, calling for a heftier and more expensive power supply needed only at system reset.

One final problem associated with memories protected by error-correction circuitry stems from the fact that only data that is accessed by the processor is corrected. If the processor continually accesses one particular segment of memory, the rest of the array may be accumulating soft errors. The possibility of two soft errors accumulating in a word of seldom accessed memory now becomes significant—and not all double-bit errors are correctable in simple ECC schemes. The 8207 scrubs memories to clean up this problem. During each refresh cycle, one word of memory is read, checked for errors, and if necessary, corrected before data is written back to memory. Because scrubbing occurs during refresh cycles with a read cycle replacing a row-address-strobe–only refresh cycle, no performance penalty is incurred. Scrubbing rids the entire memory of errors at least once every 16 seconds, reducing the probability of two soft errors accumulating in the same word almost to nil.

Bells and whistles

All dynamic RAMs require a recovery period for precharging internal lines after each access. If the processor were immediately to reaccess the RAM, the controller would have to delay it until the precharge time was over. By automatically organizing memory into banks so that sequential addresses are in different banks, the 8207 is usually able to hide the precharge time of one bank behind the access time of another. That organization follows from using the 2 least significant bits of the address to select the bank. Of course, a break in the program flow, such as would be caused by a jump or call instruction, raises the probability that the same bank may be immediately re-accessed. This probability is less in four-bank memories than in two-bank configurations.

Further performance advantages are gleaned by organizing memory into multiple banks. For example, the 8207 can speed throughput by pipelining cycles. Once the row and column addresses to one bank have been latched, the controller sends the row address for the next cycle to the next bank (Fig. 4).

The 8207's manifold features can be tailored to a given system with the use of a serial programming pin. This pin can either be strapped high or low to select one of two default modes or be programmed by means of a shift register. The external register is completely controlled by the 8207, eliminating any local processor support. Sixteen bits are shifted into the 8207 to configure up to nine different features. The bits are arranged in order of increasing importance; using a shift register with less than 16 bits permits just those features needed to be programmed.

Programmable features of the processor interface include the choice of arbitration algorithm, clock compensation, and preferred port. At the RAM interface, the user can specify fast or slow memory chips, indicate bank configuration, and select the optimal refreshing scheme. In anticipation of the next generation of 256-K dynamic RAMs, the 8207 can support a 256-row–1-millisecond refresh convention, in addition to the 128-row–2-ms one for current 16- and 64-K parts.

Helping facilitate system design is a self-programming processor interface. By decoding the command input pins at power-up, the 8207 automatically determines whether it is connected to the status lines of an 8086, iAPX-286 or to the command lines of the Multibus. Because the 8207 can directly decode the status lines of Intel microprocessors, it can anticipate the next memory cycle and start a new cycle before actually receiving a command. This extra pipelining enables the designer to specify slower RAMs then would otherwise be required.□

A SYSTEM-ORIENTED RAM CONTROLLER

Mel Bazes
James Nadir
Bradley A. May
INTEL Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

INTRODUCTION

Microprocessor-based systems are making increasing use of dynamic RAM over static RAM, as this is the most cost-effective device for implementing a large random access read/write memory. Dynamic RAM requires complex control circuitry which static RAMs do not, but the cost of this control circuitry is outweighed for memory of more than 16K bytes by the lower cost per bit and higher density of dynamic RAM.

However, successive generations of microprocessors are demanding higher performance of dynamic RAMs, as shown in Table 1 for several families of Intel microprocessors:

Some previously available controllers have provided all the required dynamic RAM support functions on a single chip, but have not been fast enough to provide no wait state performance with today's faster microprocessors. Other recently introduced dynamic RAM controllers have offered higher performance, but required several additional shipe to complete the control function. The 8207 is the first dynamic RAM controller to integrate all RAM control functions and also provide no wait state performance with all Intel microprocessors, including the iAPX-286 (80286 CPU).

Another factor of concern to a memory system designer is soft errors. Soft errors are random, nonpermanent errors, usually of a single bit. The primary mechanism of these errors in dynamic RAMs was discovered in 1978 to be loss of stored charge in the dynamic RAM cells caused by alpha particles.[1] These result chiefly from the radioactive decay of trace uranium and thorium in the packaging material. The rate of these errors increased as shrinking geometries made storage cells more susceptible to the effects of alpha particles. These soft errors have been reduced by packaging innovations, but residual errors remain[2], and so are still a concern. Since the error rate of a memory system is the sum of the error rates of all the memory components, the trend to larger RAM capacities in microprocessor systems will result in a higher system error rate even if RAM soft error rates remain constant.

Table 1. Required Memory Performance for No Wait State Operation

Year Introduced	Part Number	Clock Freq.	Access Time from Address	Access Time from Command	Bus Cycle Time
1976	8085	3 MHz	545 ns	330 ns	960 ns
1978	8086	5	430	335	800
1979	8086-2	8	265	195	500
1982	80286	8	117[a]	90	250[b]

Notes:
[a] The 80286 uses pipelined addresses. Access time from address is 222 ns for interleaved memories.
[b] The 80286 bus cycle has three clock states, but the first state is overlapped with the third state of the last cycle, for an effective bus cycle of two clocks, or 250 ns.

Coupled to these facts are the increasing reliablility requirements of many microprocessor applications. A single soft error in an automatic bank teller machine, for example, can cause an account balance error resulting in many dollar's worth of clerical work finding and correcting the error, as well as possible down time for the teller machine and frustration for the customer.

Error checking and correction, or ECC, as implemented by modified Hamming codes uses redundant memory bits to encode the data. This code allows detection and correction of all single bit errors in any memory word, and detection (but not correction) of all double bit and some higher-number-of-bit errors.

The effect of ECC of memory system reliability is shown in Table 2. These figures are based on the analysis of all types of errors (hard and soft) as measured for the Intel 2117 dynamic RAM, and shows an improvement in reliability as measured by the mean time between failures (MTBF) of between 24 and 301 times for the memory configurations shown.[3,4]

Clearly, for the assumed error rates, soft errors become unnacceptable for large RAMs, and ECC is desirable. The 8207 facilitates the addition of ECC by directly controlling the companion 8206 Error Detection and Correction Unit, and adjusting memory cycle timings as required for ECC operation automatically.

Lastly, architectural factors are important to designers of memory systems. Higher performance microprocessor systems can be obtained by the use of multiple processors operating on shared data. Common examples in microprocessor-based systems are:

1. A separate microprocessor may be used to control all I/O activities, leaving the main microprocessor free to do data processing tasks full time. If the program being executed by the main processor generates any I/O requests, these requests (read a disk file, send a file to a line printer, etc.) are formatted as a message to the I/O processor, containing the desired activity and where the input or output file is located is shared memory, and the message is placed in a reserved area of shared memory, to be read and executed by the I/O processsor. All the mechanical activities of I/O (polling disk status, spooling print files, etc.) are handled by the I/O processor, while the main processor executes the main program. The I/O processor may be on the same board or another board connected by a global bus, such as the Intel Multibus.™

2. Separate processors may execute different real-time tasks simultaneously, as in a process control application. Various process parameters will be stored in shared memory, where it may be sampled and/or updated as necessary by each of the processors to control the process in real time. Again these processors may be on the same or different boards.

In these and other applications, this multiprocessing is facilitated by dual-port memory. A dual-port memory is one in which two processors, each on its own separate bus have independent access to the same physical mem-

Table 2. Improvement in Memory Reliability with ECC

Memory Size and Organization	MTBF (no ECC)	MTBF (with ECC)	Improvement Ratio
32 Kbyte (16K x 16 bits)	5.6 yrs	133.6 yrs	24
64 Kbyte (16K x 32 bits)	2.7 "	75.1 "	28
128 Kbyte (16K x 64 bits)	1.4 "	40.5 "	29
4 Mbyte (2M x 16 bits)	16.3 days	10.8 "	246
8 Mbyte (2M x 32 bits)	8.1 "	6.1 "	278
16 Mbyte (2M x 64 bits)	4.1 "	3.3 "	301

ory. Since the processors are independent, the dual port memory itself must resolve the conflict that arises if each processor tries to access the memory simultaneously.

If multiple processors are used on a single bus, performance will be limited by the bandwidth of the bus, and the time spent exchanging bus control. A dual-port RAM allows each processor full use of its own bus. To prevent the bandwidth of the dual-port RAM from limiting system performance, at least one processor should have its own (single-port) RAM, and only those memory segments shared between processors should be placed in dual-port RAM.

Also, some method (called a "semaphore") must be provided to control access to the memory so that one processor can modify sections of data without the other processor being able to see that section while it is being modified. As an example, if a shared memory contained a list of passengers on an airline flight, an error would result if two processors at almost the same time found an empty space on the passenger list, and each wrote a new passenger's name into that same space. One processor must prevent the other from accessing the passenger list before it can look to see if there are empty places.

The 8207 has a dualport memor interface. Port arbitration is don on-chip. Semaphores are supported i hardware by a LOCK input which may b activated by one port to prevent memo ry accesses by the other port.

DEVICE DESCRIPTION

The 8207 Advanced Dynamic RAI Controller (described by Figure 1 provides all required dynamic RAM control functions in a single chip, including:

1. Address multiplexing The 8207 generates the row and columr addresses used by the dynamic RAMs.

2. Refresh The 8207 internally generates refresh cycles when necessary; an external input allows the user's system to generate refresh cycles when desired. An 8-bit counter determines the row to be refreshed.

3. Arbitration Since read, write, and refresh cycles cannot be done simultaneously, the 8207 determines which will be performed, and when. The 8207 arbitrates between memory requests from each of the two ports, and the refresh logic. Because a requested mem

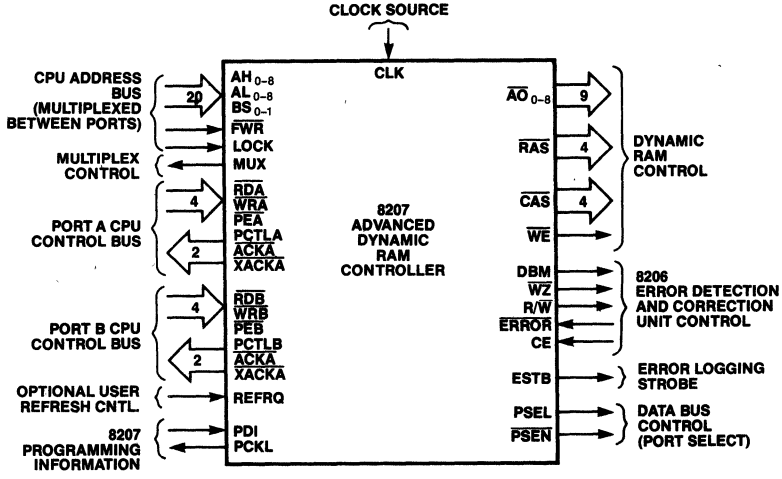


Figure 1. 8207 Logic Symbol

ory cycle may be delayed by a cycle in progress, the 8207 provides separate acknowledge signals to each port to indicate when the requested cycle has been completed.

4. Cycle timing Timing of the RAM addresses and control strobes, ECC and error strobes, port multiplexing signals, and memory cycle acknowledge signals are all generated internally by the 8207. Refresh timing, arbitration, and cycle timing are all done from a single clock input. Timing changes may be made by programming options on the 8207, or by adjusting the clock frequency.

The 8207 directly addresses and drives up to 88 RAMs (16K, 64K, or 256K), with no external drivers. The 8207 Advanced Dynamic RAM Controller, like its companion, the 8206 Error Detection and Correction Unit, is implemented in HMOS II, a production proven NMOS process, and is packaged in a 68-pin JEDEC Type A chip carrier.

Programming

In order to optimize its performance in as many system environments as possible, the 8207 programs itself at system reset with information about the application system; what type of microprocessor it is interfaced to, whether ECC is used or not, how port access priority should be resolved, etc. The 8207 uses one pin (PDI) forprogramming, through which programming data is serially shifted, with another pin (PCLK) used as a shift clock, as shown in Figure 2.

Tying PDI to ground programs the 8207 to operate in non-error-correcting mode; tying PDI high puts the 8207 in error-correcting mode. All other programming options default to values optimized for those configurations. If it desired to change any options from their default values, a parallel-in-serial-out (PISO) shift register, such as the 74LS165, may be attached to the programming pin. The system reset input loads this shift register with the jumper-selected programming options, which are then clocked into the 8207. One or two shift registers may be used to provide up to 16 bits of programmability.

Microprocessor Interface

The 8207 can operate in single-ported or dual-ported memory configurations. Each port is individually programmable to operate in a variety of system configurations.

Each port may respond to standard demultiplexed read and write commands, as shown in Figure 3b. The $\overline{RD}$ and $\overline{WR}$ inputs are internally qualified by a port enable ($\overline{PE}$) signal, which is normally decoded from the address bus. No memory cycle can start unless $\overline{PE}$ is

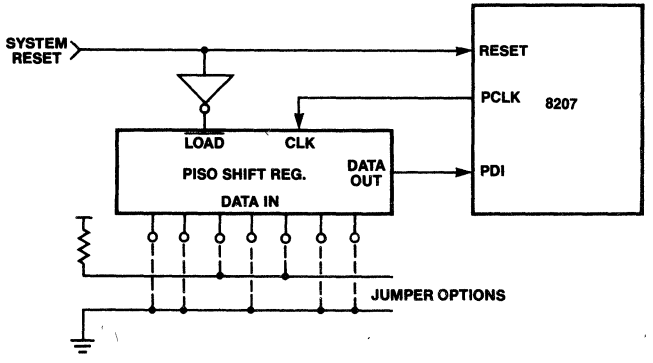
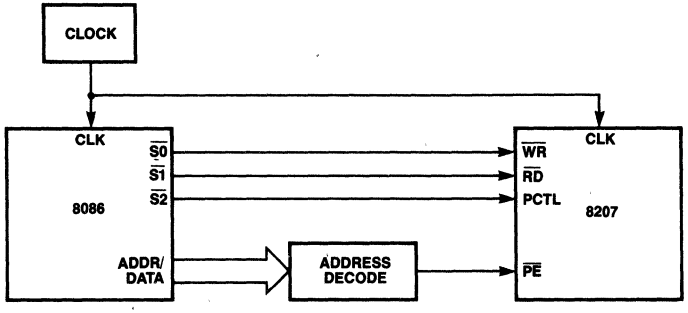


Figure 2. Programming Interface

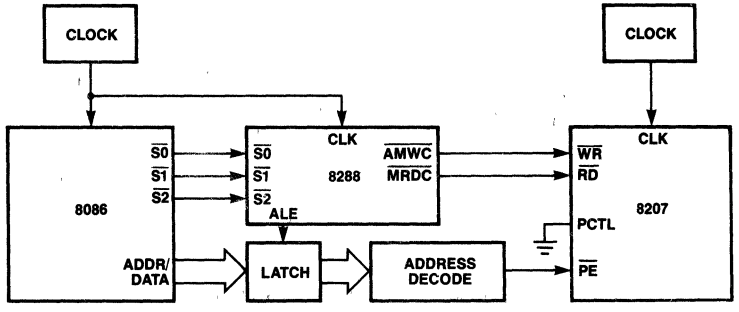
active. A port control (PCTL) signal is provided, which qualifies the start of a memory cycle, and can also inhibit a memory cycle that has already been started. If PCTL is deactivated before the memory cycle starts, no cycle will be performed. If it is deactivated after the RAM cycle starts, the memory cycle will complete, but the RAM write enable, memory cycle acknowledge, and error logging strobes will be disabled; no data will be written to memory. The user must use PCTL to disable data transceivers to prevent data from being read.

PCTL is useful if RAM and ROM are overlaid in the same address space, and it is desired to de-select the RAM in those addresses where ROM is present. This is often the case at system reset, where a "boot-strap" ROM is selected in part of the RAM address space to start the system. RAM is selected in this same address space once the system is up and running. PCTL may also be used for those microprocessors which have a separate memory management unit which generates an inhibit signal if a program attempts to access protected memory.

Each port may also be programmed to directly decode the status outputs of Intel's iAPX-86, 88, 186, and 286 families of microprocessors, as shown in Figure 3b. In this mode, the RD, WR, PE, and PCTL inputs are redefined to be the apporpriate status inputs. When used in this way, the 8207 is normally operated synchronously to the microprocessor, from the same clock generator, as shown in Figure 3a. Because the 8207 must be able to operate syncronously with several different microprocessors with clock rates from 5 MHz to 16 MHz and with different bus cycle timings, the 8207 varies memory cycle timings so as to be compatible with the microprocessor it is interfaced to.



A. SYNCHRONOUS



B. ASYNCHRONOUS

Figure 3. 8086 Interface

When used asynchronously to the microprocessor, all inputs are internally synchronized by the 8207. When used synchronously, these synchronizers are bypassed, eliminating synchronization delays.

Arbitration

Each port has physically separate RD, WR, PE, and PCTL inputs, as shown in Figure 4. In this application, the 8207 interfaces synchronously to Port A (8086) and asynchronously to Port B (Multibus); all Port B inputs are synchronized, but synchronizers are bypassed on Port A.

The abitration logic arbitrates between three ports; Port A, Port B, and Port C (the refresh port). Any port may request memory at any time.

Once Port A or B is selected, it has immediate access to the cycle timing generators and the arbitration logic is bypassed for subsequent memory cycles. The arbitration delay in this case is zero. If the unselected port requests a memory cycle, the arbiter must first select it. Arbitration for the next memory cycle is done in parallel with a memory cycle in progress, so usually this arbitration time is hidden, and the new cycle starts as soon as the cycle in progress to the previously selected port is complete.

One of two arbitration algorithms may be selected to "tailor" the 8207 to the application. In one method, Port A is the preferred port, so whenever the 8207 is idle, Port A is selected, minimizing access time for that port. In the other method, the most recently used port is selected.

As an example, if Port A was attached to a high-speed microprocessor used in for data processing, and Port B was attached to an I/O processor, it would probably be better to make Port A the preferred port, to minimize its average access time and maximize processing throughput, especially since the I/O processor on Port B is limited by the slow speed of the mechanical peripherals to which it is attached, not memory access time.

A LOCK input is provided which allows either port to lock out the other and gain sole access to the memory. This is useful for testing and setting semaphores in shared memory segments. It may also be used to allow one port to transfer bursts of data to or from memory at maximum bandwidth, without the other port stealing any memory cycles.

Due to pin limitations, the address, RAM bank select, and LOCK inputs are multiplexed between Ports A and B. The arbiter generates a MUX output which is used for this purpose, as shown in Figure 4. This figure

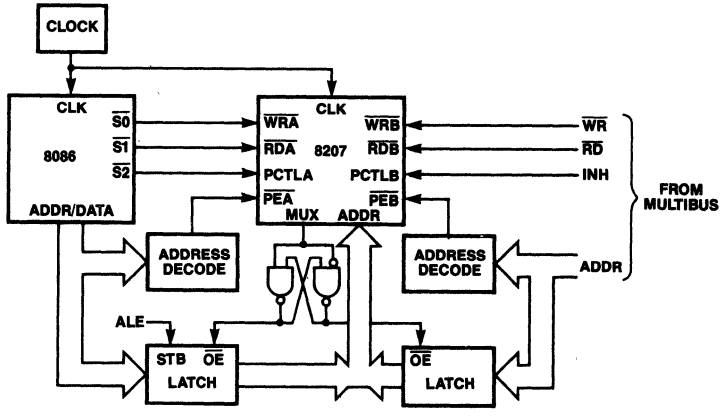


Figure 4. 8086/Multibus Dual-Port Interface

shows multiplexing being done by alternately enabling 3-state latches; cross-coupled NAND gates are used to create non-overlapping enable signals.

RAM Interface

Nine RAM address outputs and four pairs of Row Address Strobe ($\overline{RAS}$) and Column Address Strobe ($\overline{CAS}$) signals allow the 8207 to interface to four banks of either 16K, 64K, or 256K RAMs.

Output drivers with high capacitive drive capability allow the 8207 to drive up to 88 dynamic RAMs, arranged as four banks of 22 RAMs each (16 data bits plus 6 ECC check bits) without external drivers. A.C. timings are specified at a load of 550 pF on Address outputs and 250 pF on $\overline{RAS}$ and $\overline{CAS}$ outputs. With this large a load, the transient currents are in the ampere range; because of this, the output drivers are isolated from the rest of the circuitry with separate I_{CC} and ground pins for each. The driver circuitry was also designed to prevent any "boot-strapping" effect and to limit voltage overshoot.

Novel circuit design techniques were used to allow RAM outputs to be generated with less than 35 ns propagation delay from the 8207 clock input, while requiring input set-up times of only 20 ns[5].

If only one or two banks of RAM are occupied, the $\overline{RAS}$/$\overline{CAS}$ output drivers re-allocate themselves to increase drive capacity. If only two banks here are occupied, $\overline{RAS}$ 0 and $\overline{RAS}$ 1 work in tandem to drive Bank 0, and $\overline{CAS}$ 0 and $\overline{CAS}$ 1 do similarly; $\overline{RAS}$ 2 and $\overline{RAS}$ 3; and $\overline{CAS}$ 2 and $\overline{CAS}$ 3 work in tandem to drive Bank 1. In this way, the two banks may consist of up to 44 RAMs (twice as many as before) without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Since the total number of RAMs remains the same, the loading on the Address outputs, which go to all RAMs also remains the same. If only one bank is occupied, all four $\overline{RAS}$ and $\overline{CAS}$ driver work together to allow driving a single bank of 88 RAMs.

Separate $\overline{RAS}$ and $\overline{CAS}$ outputs also allow the 8207 to interleave memory cycles to alternate banks of RAM, as shown in Figure 5. In this way, the RAM's precharge time (t_{RP}), which is normally part of the RAM cycle time, may be hidden in the following memory cycle. If consecutive cycles are to the same bank, the second cycle must be delayed, as shown by the dotted line in Figure 5.

Refresh

The 8207 supports both the 128 row/2 ms or the 256 row/4 ms refresh conventions. A 256 row/2 ms option may also be programmed for possible use with 256K RAMs.

The 8207 generates one refresh cycle approximately every 15 µs, thus executing 128 refresh cycles in 2 ms, or 256 cycles in 4 ms. If the 256 row/2 ms option is used, one refresh cycle is generated approximately every 7.5 us. An 8-bit counter keeps track of the row to be refreshed; this address is provided to the RAM by the 8207 address outputs.

An external refresh request input (REFRQ) is provided so the system may cause refreshes to be perfomed whenever desired. If no refreshes are

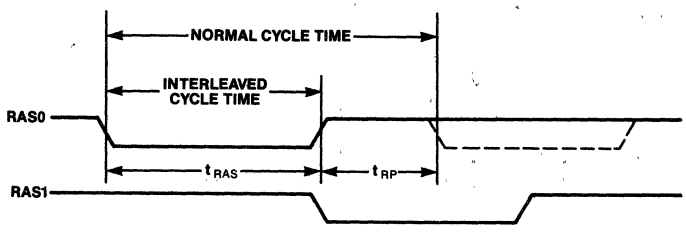


Figure 5. 8207 Bank Interleaving

equested in 15 μs, the 8207 may enerate an internal "failsafe" efresh. The internal refresh may be urned off entirely, so that the only efresh cycles are those externally enerated. In this case, bursts of 128 efresh cycles (internally counted) ill also be performed by activating he REFRQ for a slightly longer period f time.

Since the refresh cycles are timed from the 8207 clock input, the number of clocks between refresh cycles may be programmed to allow for differences in clock frequencies between 3.5 MHz and 16 MHz.

ECC Interface

The 8207 may also be programmed to support error checking and correction (ECC). When in this mode, cycle timings are adjusted and several pins change function to support the companion 8206 Error Detection and Correction Unit, as shown in Figure 6.

The 8206 (described in Figure 7) can correct single bit errors in any memory word, and detect (but not correct) all double bit errors, and some errors of more than two bits. Each 8206 does error correction on a 16-bit slice of the memory word, and up to five 8206s may be cascaded for 80-bit data words.

The 8206 generates modified Hamming code check bits on write cycles, checks for and corrects errors on read cycles, and generates an error flag, a correctable/uncorrectable error flag, and syndrome outputs which may be used to pinpoint the bit in error for error logging. Both data and check bit errors are corrected automatically, without changing the mode of the 8206. Control inputs are provided for byte writes. A data/check bit input latch is provided. Unique separate data/check bit input and data/ check bit output busses reduce the amount of control required to implement an ECC memory system.

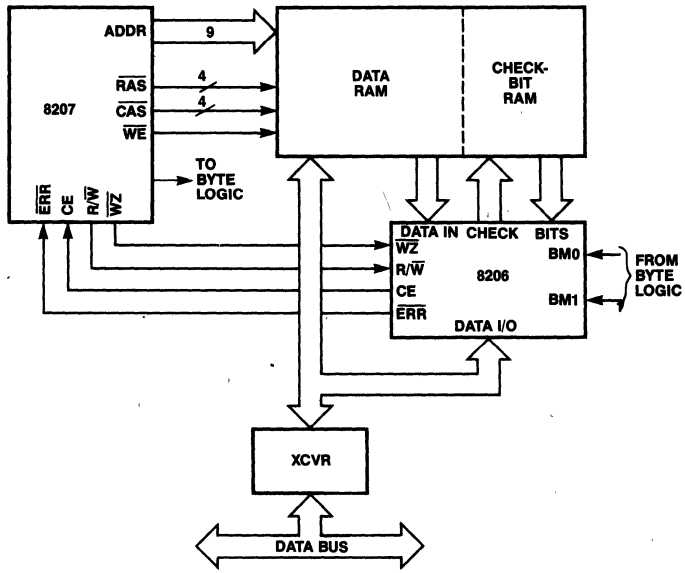


Figure 6. ECC Interface

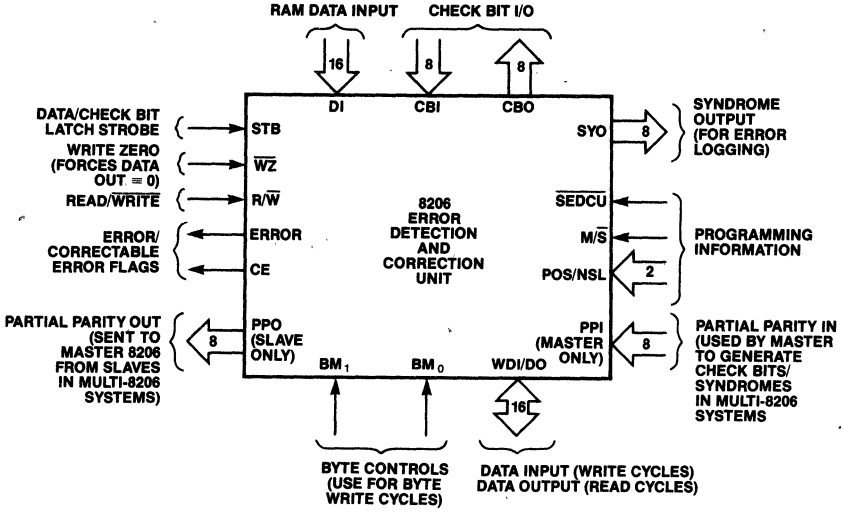


Figure 7. 8206 Logic Symbol

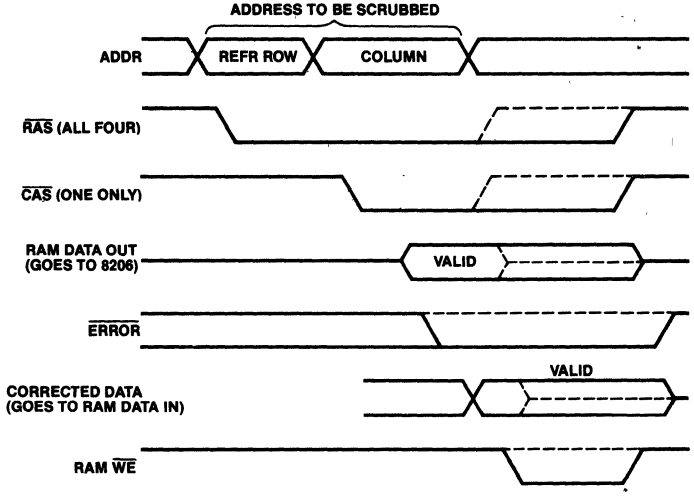
All correctable errors dicsovered during read cycles are immediately corrected in RAM. The 8207 monitors the 8206 $\overline{\text{ERROR}}$ flag during read cycles. If it is active, the read cycle is lenghtened to become a read-modify-write cycle; the 8207 also activates an Error Strobe which can be used to latch the 8206 error flags and syndrome outputs for error loggging, or to interrupt the microprocessor. During byte write cycles, error correction is automatically done on the byte(s) of the word not being changed.

To further increase memory reliability, the 8207 does memory scrubbing during RAM refresh, as shown in Figure 8. During a refresh cycle, the 8207 refreshes one row in all banks. It also outputs a column address and activates one of the four $\overline{\text{CAS}}$ outputs, to read one word of memory. That word is checked for errors by the 8206; if any are found, the corrected data is written back into RAM. By this process, the entire memory is scrubbed of errors every 30 seconds or less, making the possibility of two soft errors accumulating in the same word (an uncorrectable error) virtually nil.

RAM Initialization

Upon system reset, the 8207 resets its refresh counter and performs eight cycles to all four banks at once to "warm up" the dynamic RAMs (many dynamic RAMs require this warm up period after power up for stable operation).

In ECC mode, the 8207 also initializes all of memory to zero data and the corresponding check bits. It does this by activating the 8206 Write Zero input and performing consecutive write cycles to each memory location. This is done because on power up the contents of the RAM are undefined, and subsequent byte writes may cause false "errors" due to the undefined data in memory. Because the time to do this



NOTE: DOTTED LINES SHOW CASE WHERE NO ERROR IS DETECTED.

FIGURE 8. SCRUB CYCLE (ERROR DISCOVERED AND CORRECTED IN RAM)

is fairly long, about 1 second, the 8207 may be programmed to skip initialization on reset.

Any memory requests during the warm up/initialization sequence will be latched internally and responded to as soon as the sequence is complete. By having the 8207 do these "housekeeping" tasks, they are no longer a burden on the system software, and they are done faster than would be possible by software.

SUMMARY

Previous dynamic RAM controllers have offered either integrated control or high performance. The 8207 is the first dynamic RAM controller to provide complete dynamic RAM control on a single chip and no wait state performance with all Intel microprocessors, including the iAPX-286.

In addition, the 8207 offers other advanced features, such as ECC control, automatic error scrubbing, and a dual-port RAM interface.

ACKNOWLEDGEMENTS

The authors wish to thank Moti Mebel, David Perlmutter, Beni Mantel, and Omer Zak, for their contributions to the design of the 8206 and 8207.

REFERENCES

[1] "A New Physical Mechanism for Soft Errors in Dynamic Memories"
Timothy C. May, Murray H. Woods, INTEL Corp.
IEEE Proceedings of the Int'l Reliability Physics Symposium, April 1978.

[2] "Alpha-Particle-Induced Soft Error Rate Modeling"
George A. Sai-Halasz, IBM Research Center
IEEE Int'l Solid State Circuits Conference, Feb. 1982.

[3] "Keep Memory Design Simple Yet Cull Single-Bit Errors"
M. Bazes, L. Farrell, B. May, M. Mebel, INTEL Corp.
Electronic Design, Sept. 30, 1981.

[4] "Memory System Reliability with ECC"; Intel Ap Note 73
Dennis Marston, INTEL Corp.

[5] "An NMOS DRAM Controller"
M. Bazes, J. Nadir, D. Perlmutter, B. Mantel, O. Zak, INTEL Corp.
IEEE Int'l Solid State Circuits Conference, Feb. 1982.

intel

TECH
PAPER

August 1982

An NMOS DRAM Controller

Mel Bazes, James Nadir, David Perlmutter,
Beny Mantel, Omer Zak
Intel Corp.

AUGUST 1982
ORDER NUMBER: 210590-001

SESSION VII: DYNAMIC RAMs

WPM 7.4: An NMOS DRAM Controller

Mel Bazes, James Nadir, David Perlmutter, Beny Mantel, Omer Zak

Intel Corp.

Santa Clara, CA

TO DATE, integrated dynamic RAM controllers have been either optimized for speed or for features, with one usually at the expense of the other. Controllers designed to give no wait states have required several additional TTL devices and delay lines to complete the basic dynamic RAM control functions. Integrated controllers have slowed today's microprocessors by adding wait states. The chip to be described, the DRC (Dynamic RAM Controller), with its high level of integration, can support 8MHz CPUs with 150ns DRAMs without wait states. The controller, operating in both single-port and dual-port configurations and in synchronous and asynchronous environments, supports error correction circuitry, and provides all of the timing and control signals necessary for a memory module. Two microprocessor bus ports are independently programmable for either synchronous or asynchronous operation, and each is capable of supporting several possible bus structures.

The chip, shown in Figure 1, is 233 by 199 mils and is fabricated in HMOS II, a high-speed NMOS process.

The architecture of the DRC has been optimized with consideration given both to performance goals and to technology limitations. Figure 2 illustrates a simplified block diagram of the DRC. The controller simultaneously processes commands from three ports, two externals ports which give read/write commands, and an internal port which gives refresh commands. To introduce only a minimal delay in the start of a RAM cycle, "up front" arbitration between the three ports was ruled out. Instead, the port commands are directly multiplexed into the RAM timing generators, with the arbiter selecting only one of the multiplexer channels at any time, even when no cycle is in progress. When two or more commands arrive simultaneously, the commands not serviced are queued. While the RAM cycle for the selected port is in progress, the arbiter processes the commands from the other ports. The arbiter will select the next port to be serviced even before the current RAM cycle has finished, but only late enough in the cycle so that the cycle is not affected. Thus, as soon as the current cycle is completed, the cycle for the next port starts with little or no delay. In general, a command from a selected port is serviced with little or no delay, while a command from an unselected port is serviced with a delay equal to the arbitration time. Two user-selectable arbitration algorithms are available on the DRC.

The DRC operates in either of two modes, an 8MHz mode for optimal performance with the present day CPUs and an 16MHz mode for optimal performance with future CPUs*. The relatively high frequency of the clock in 16MHz mode, coupled with its variable clock parameters (15ns low time and 20ns high time) necessitated taking a novel approach to the DRC logic and circuit design. Instead of using two-phase logic, most common in NMOS design, the DRC was implemented using single-phase edge triggered flipflops. The use of only one clock phase inherently precludes the problem of clock overlap, while sequencing logic on a single clock edge provides several important benefits. First, the sensitivity to clock low and high times is greatly reduced. Second, logic design is significantly simplified. Third, overall logic throughput can be made more efficient than with two-phase logic. Finally, the straightforward logic design results in simplified circuit design, as evidenced by the fact that the first iteration of the DRC that was fabricated was functional at the full 16MHz clock frequency.

The DRC output pulse timings have been selected for optimal performance within at least the 62.5ns resolution provided by the clock. In the case of the critical Row Address Select (RAS), Column Address Select (CAS), and address outputs, the resolution obtained is down to only a few nanoseconds and is provided internally by specially designed delay elements. These delay elements are implemented as voltage controlled capacitive loads which compensate the chip output switching delays against supply, temperature, and processing variations.

In asynchronous environments, access time is improved by requiring an overhead of only one 62.5ns clock period in synchronizing up asynchronous inputs.

An important feature of the DRC is its ability to sample a single input with a setup time of only 20ns and to clock out several different output signals from that single input in under 35ns. This low setup time is obtained by using zero-resistance input protection devices to reduce RC delays, and to provide TTL-to-MOS buffering directly at the generator circuits, rather than at the input pads. The short output delay is obtained by triggering the output generators directly off of the unbuffered external clock, thereby saving the buffer delay time present in the buffered clock. Figure 3a illustrates an example of a high-speed pulse generator circuit which is triggerable off of either the clock rising edge or the clock falling edge. The pulse generator timing behavior, is shown in Figure 3b.

The objective of creating a highly-integrated dynamic RAM controller has resulted in a flexible device with sophisticated arbitration between ports, optimized output pulse timings, and minimal input setup and output delay times, and which allows designing higher performance memories using slower speed dynamic RAMs.

Acknowledgments

The authors would like to acknowledge the contributions of the staff at Intel/Israel and in particular to the following individuals for their significant contributions to the project: Lizette Ouzan and all the other mask designers, Richard Studnicki and Aurora Fitlovitch – tester support, and finally to Dick Hodgman and Brad May for their system support.

*INTEL 286

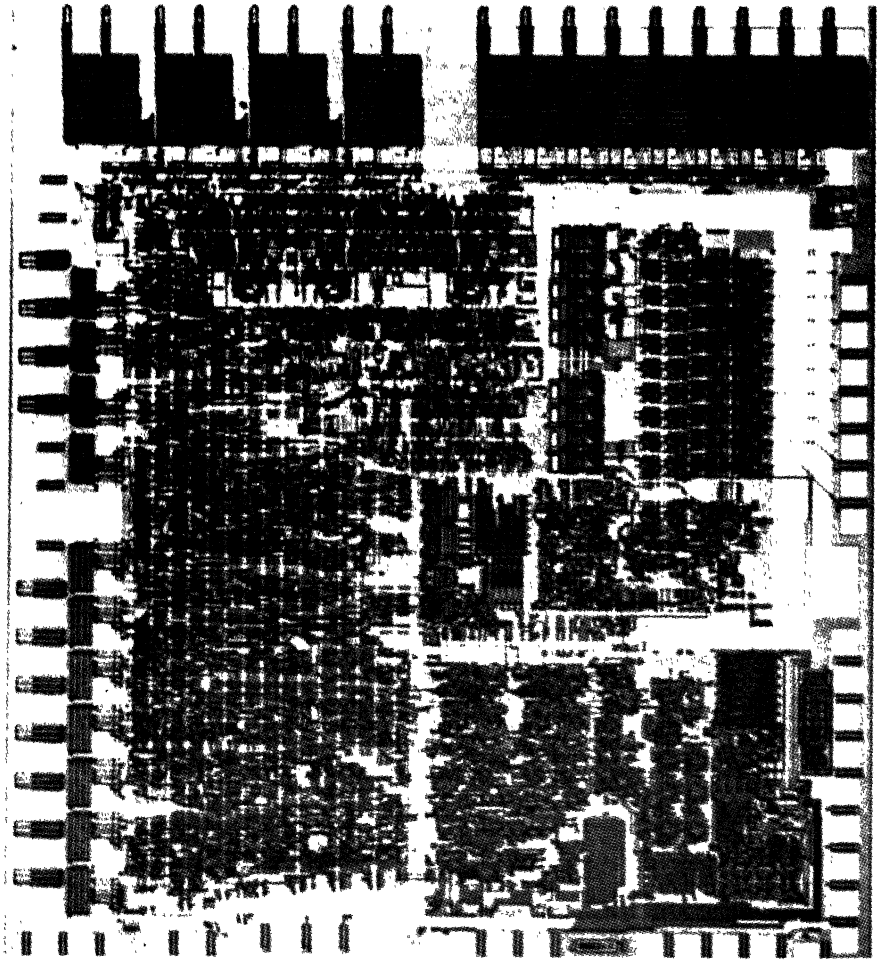


FIGURE 1–Die photograph of the DRAM controller.

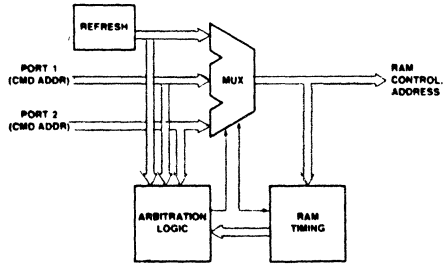


FIGURE 2–Simplified block diagram of DRAM controller.

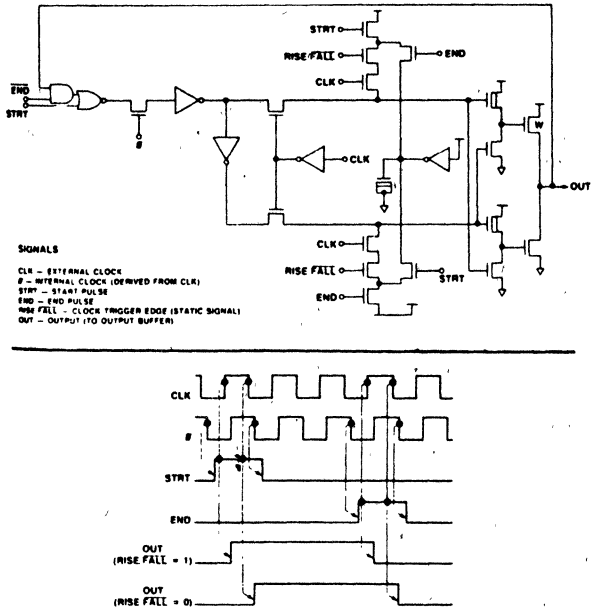


FIGURE 3: (*a*)-Above–high speed pulse generator circuit, triggerable off of either clock edge; (*b*)–below–timing wave forms for the pulse generator circuit.

8202A
DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 2117, or 2118 Dynamic Memories**
- **Directly Addresses and Drives Up to 64K Bytes Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Refresh Cycles May be Internally or Externally Requested**
- **Provides Transparent Refresh Capability**
- **Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors**
- **Decodes CPU Status for Advanced Read Capability with the 8202A-1 or 8202A-3**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Internal Clock Capability with the 8202A-1 or 8202A-3**

The Intel® 8202A is a Dynamic Ram System Controller designed to provide all signals necessary to use 2117 or 2118 Dynamic RAMs in microcomputer systems. The 8202A provides multiplexed addresses and address strobes, as well as refresh/access arbitration. The 8202A-1 or 8202A-3 support an internal crystal oscillator.

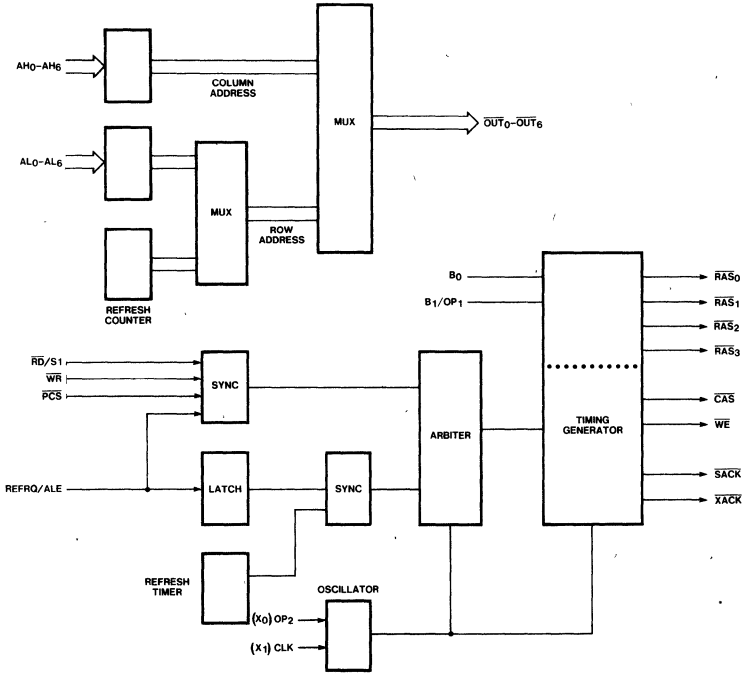


Figure 1. 8202A Block Diagram

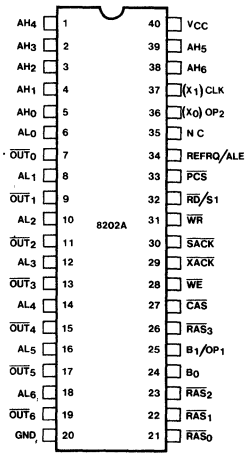


Figure 2. Pin Configuration

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6/	6 8 10 12 14 16 18	I I I I I I I	**Address Low:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**Address High:** CPU address inputs used to generate memory column address.
BO B_1/OP_1	24 25	I I	**Bank Select Inputs:** Used to gate the appropriate $\overline{RAS}_0$-$\overline{RAS}_3$ output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode.
$\overline{PCS}$	33	I	**Protected Chip Select:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**Memory Write Request.**
$\overline{RD}$/S1	32	I	**Memory Read Request:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ ALE	34	I	**External Refresh Request:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT}_0$ $\overline{OUT}_1$ $\overline{OUT}_2$ $\overline{OUT}_3$ $\overline{OUT}_4$ $\overline{OUT}_5$ $\overline{OUT}_6$	7 9 11 13 15 17 19	O O O O O O O	**Output of the Multiplexer:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0-6}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**Write Enable:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**Column Address Strobe:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS}_0$ $\overline{RAS}_1$ $\overline{RAS}_2$ $\overline{RAS}_3$	21 22 23 26	O O O O	**Row Address Strobe:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8202A Bank Select pins (B_0, B_1/OP_1).
$\overline{XACK}$	29	O	**Transfer Acknowledge:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.
$\overline{SACK}$	30	O	**System Acknowledge:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
(X_0) OP_2 (X_1) CLK	36 37	I/O I/O	**Oscillator Inputs:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is connected to a 1KΩ resistor pulled to +12V then X_1/CLK becomes a TTL input for an external clock.
N.C.	35		Reserved for future use.
V_{CC}	40		**Power Supply:**+5V.
GND	20		**Ground.**

NOTE: Crystal mode for the 8202A-1 or 8202A-3 only.

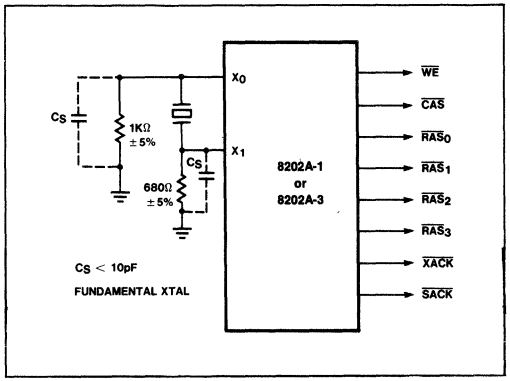


Figure 3. Crystal Operation for the 8202A-1 and the 8202A-3

AFN 01838A

Functional Description

The 8202A provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2117 and 2118 dynamic RAMs.

All 8202A timing is generated from a single reference clock. This clock is provided via an external oscillator or an on chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$ (trailing edge).

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The advanced READ mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal refresh timer which generates internal refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8202A clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

Option Selection

The 8202A has two strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a RD input to an S1 input, and pin 34 changes from a REFREQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8202A-1 or 8202A-3 only.

When OP_2 is selected, by connecting pin 36 to +12V through a 1K ohm resistor, pin 37 changes from a crystal input (X_1) to the CLK input for an external TTL clock.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_6 are gated to $\overline{OUT}_0$–$\overline{OUT}_6$, then AH_0–AH_6 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_6$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 8202A has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE allows refresh requests to be requested external to the 8202A.

All three of these inputs may be asynchronous with respect to the 8202A's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 8202A is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8202A is normally in the IDLE state. Whenever one of the other cycles is requested, the 8202A will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8202A will return to the IDLE state.

Description	Pin #	Normal Function	Option Function
B1/OP1	25	Bank (RAS) Select	Advanced-Read Mode (see text)
X_0/OP_2	36	Crystal Oscillator (8202A-1 or 8202A-3)	External Oscillator

Figure 4. 8202A Option Selection

Test Cycle

The TEST Cycle is used to check operation of several 8202A internal functions. TEST cycles are requested by activating the $\overline{RD}$ and $\overline{WR}$ inputs, independent of $\overline{PCS}$. The TEST Cycle will reset the refresh address counter. It will perform a WRITE Cycle if $\overline{PCS}$ is low. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8202A has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8202A refresh cycles activate all of the $\overline{RAS}$ outputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8202A clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds. If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8202A clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8202A is not in the middle of a cycle.

Simultaneous memory request and external refresh request will result in the memory request being honored first. This 8202A characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 5 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8202A performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 8202A can complete the refresh cycle before the next memory request is generated.

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

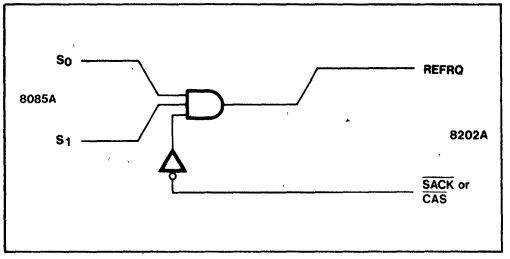


Figure 5. Hidden Refresh

Read Cycles

The 8202A can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B1 input	+12 Volt Option
Pin 32	RD input	S1 input
Pin 34	REFRQ input	ALE input
# RAM banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh Req.	Yes	No

Figure 6. 8202A Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 8202A responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similiar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8202A will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8202A will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-SACK latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will

normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similiar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 8202A Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests, and should also pullup $\overline{WR}$ to keep the 8202A out of its test mode.

In order to minimize propagation delay, the 8202A uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The 8202A t_{AD} AC parameter should be used for this system calculation.

The B0-B1 inputs are similiar to the address inputs in that they are not latched. B0 and B1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 8202A uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8202A synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8202A.

A microprocessor system is concerned with the time data is valid after $\overline{RD}$ goes low. See Figure 7. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (t_{RAC}) and the CAS-access time (t_{CAC}). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable $t_{CC,max}$ (8202A) + t_{CAC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8202A normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the $\overline{WR}$ input signal or delay the 8202A $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 8202A timing, including the READY handshake signals, $\overline{SACK}$ and $\overline{XACK}$, which may increase the number of WAIT states generated by the CPU.

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the CAS active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

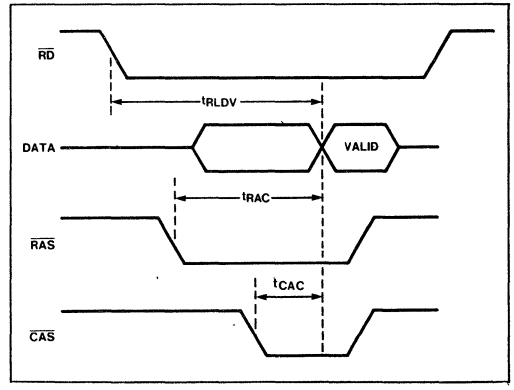


Figure 7. Read Access Time

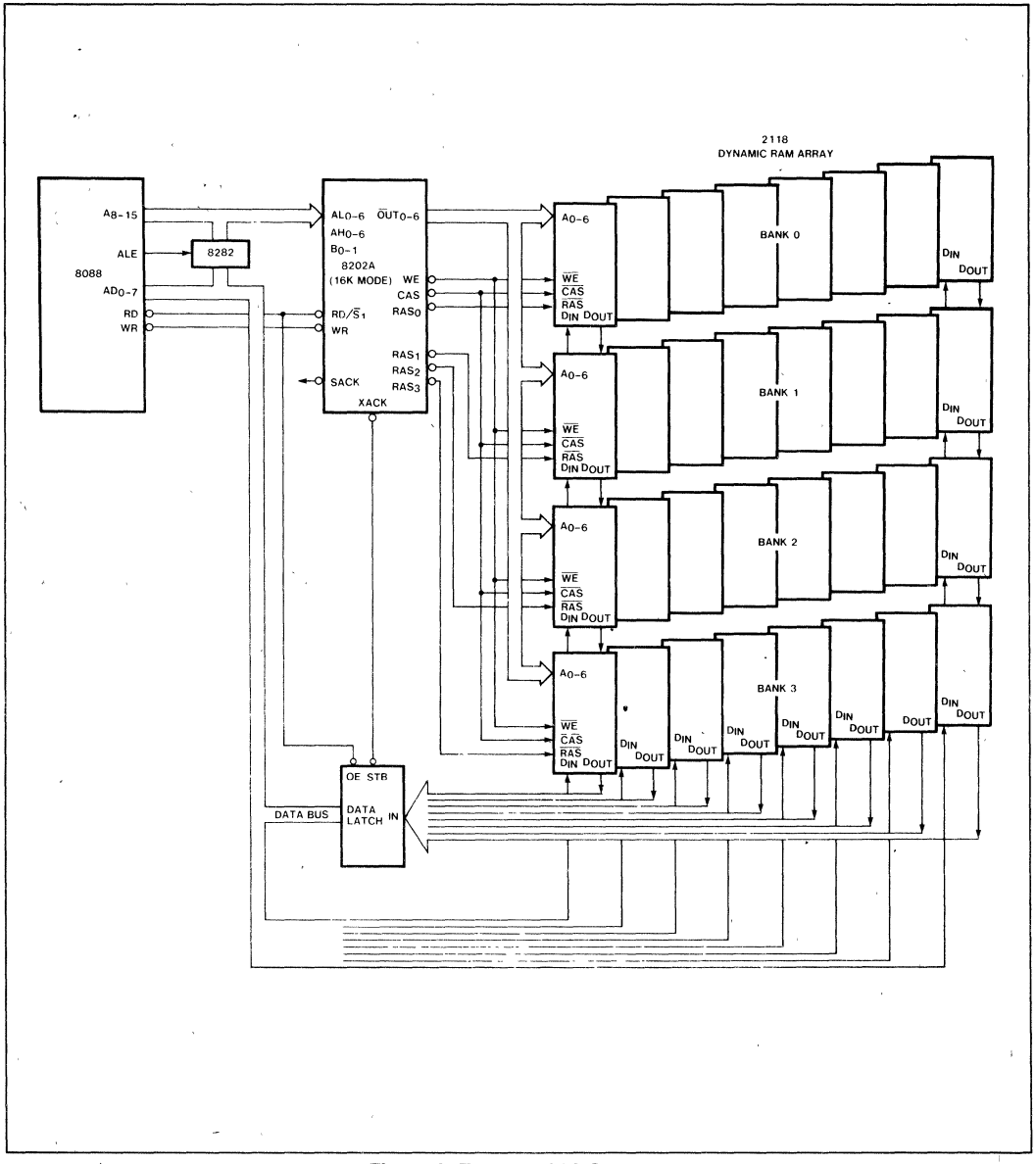


Figure 8. Typical 8088 System

ABSOLUTE MAXIMUM RATINGS*

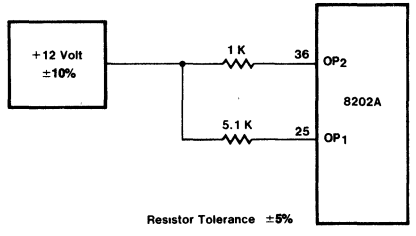
Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On any Pin
With Respect to Ground .....−0.5V to +7V[4]
Power Dissipation .....1.5 Watts

**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ± 10%, V_{CC} = 5.0V ± 5% for 8202A-3, GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		270	mA	
I_F	Forward Input Current				
	CLK		−2.0	mA	V_F = 0.45V
	All Other Inputs[3]		−320	μA	V_F = 0.45V
I_R	Reverse Input Current[3]		40	μA	$V_R = V_{CC}$ (Note 1)
V_{OL}	Output Low Voltage				
	$\overline{SACK}$, $\overline{XACK}$		0.45	V	I_{OL} = 5 mA
	All Other Outputs		0.45	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage				V_{IL} = 0.65V
	$\overline{SACK}$, $\overline{XACK}$	2.4		V	I_{OH} = −1 mA
	All Other Outputs	2.6		V	I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V (Note 2)
V_{IH1}	Input High Voltage	2.0		V	V_{CC} = 5.0V
V_{IH2}	Option Voltage			V	(Note 4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTES:
1. I_R = 200μA for pin 37 (CLK) for external clock mode
2. For test mode RD & WR must be held at GND.
3. Except for pin 36.
4. 8202A-1 and 8202A-3 supports both OP_1 and OP_2. 8202A only supports OP_2



A.C. CHARACTERISTICS

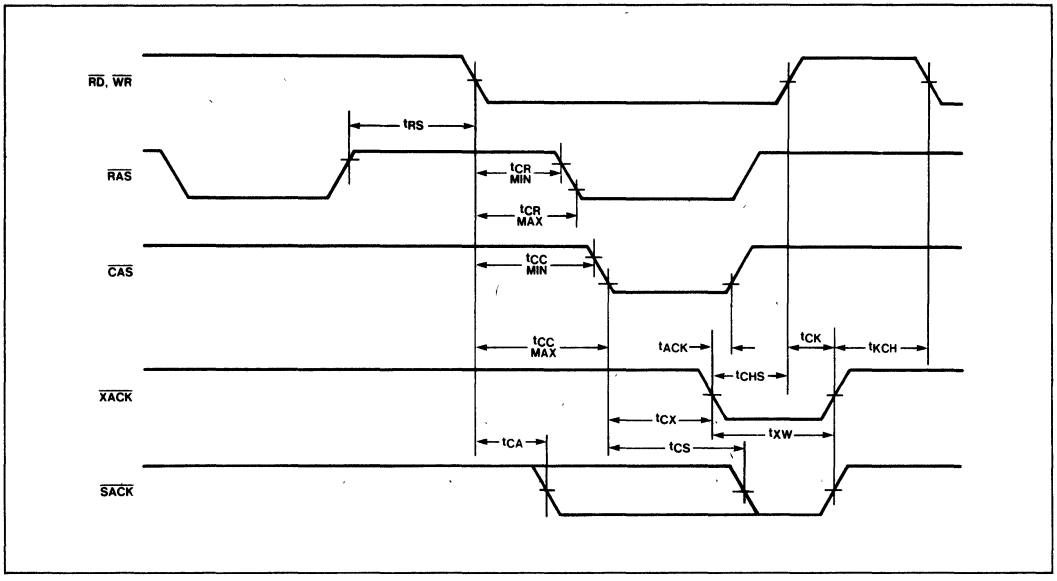
T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, V_{CC} = 5V ± 5% for 8202A-3

Measurements made with respect to $\overline{RAS}_0$-$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$,$\overline{OUT}_0$-$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

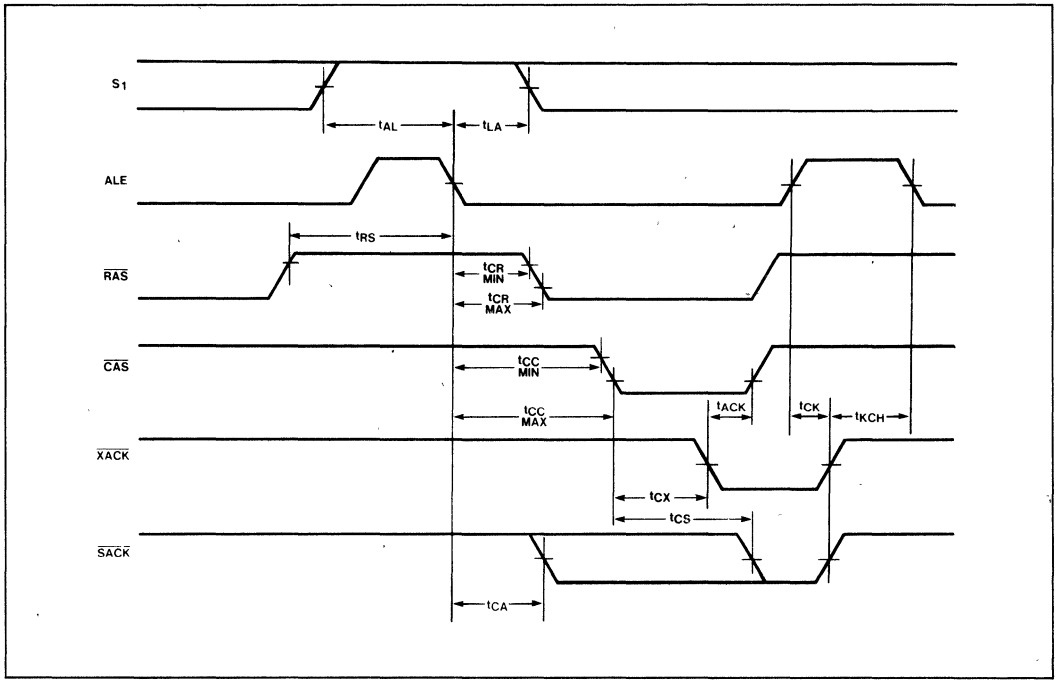
Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	40	54	
t_{PH}	External Clock High Time	20		
t_{PL}	External Clock Low Time—above (>) 20 mHz	17		
t_{PL}	External Clock Low Time—below (<) 20 mHz	20		
t_{RC}	Memory Cycle Time	10tp − 30	12tp	4, 5
t_{REF}	Refresh Time (128 cycles—16K mode)	264tp	288tp	
t_{RP}	$\overline{RAS}$ Precharge Time	4tp − 30		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	5tp − 30		3
t_{ASR}	Address Setup to $\overline{RAS}$	tp − 30		3
t_{RAH}	Address Hold From $\overline{RAS}$	tp − 10		3
t_{ASC}	Address Setup to $\overline{CAS}$	tp − 30		3
t_{CAH}	Address Hold from $\overline{CAS}$	5tp − 20		3
t_{CAS}	$\overline{CAS}$ Pulse Width	5tp − 10		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	tp − 40		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	5tp − 35		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ delay from $\overline{RAS}$	5tp		
t_{MRP}	$\overline{RD}$, $\overline{WR}$ setup to $\overline{RAS}$	0		5
t_{RMS}	REFRQ setup to $\overline{RD}$, $\overline{WR}$	2tp		
t_{RMP}	REFRQ setup to $\overline{RAS}$	2tp		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold from ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	tp + 30	2tp + 70	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	3tp + 25	4tp + 85	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ setup to REFRQ	5		
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay		2tp + 47	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	5tp − 25	5tp + 20	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	5tp − 25	5tp + 40	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	tp − 25		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ turn-off Delay		35	
t_{KCH}	CMD Inactive Hold after $\overline{SACK}$, $\overline{XACK}$	10		
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		4tp + 100	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	40	3

WAVEFORMS

Normal Read or Write Cycle

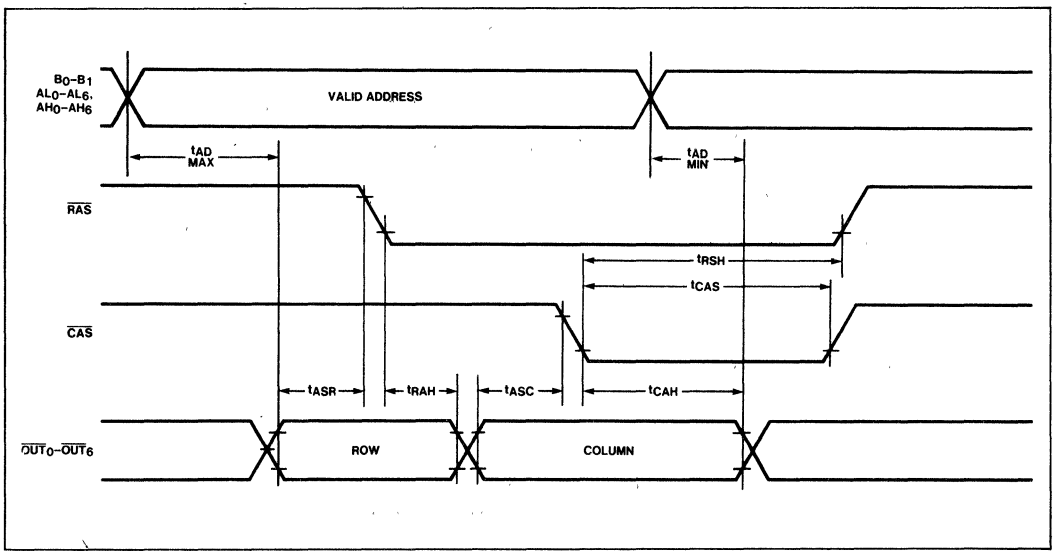


Advanced Read Mode

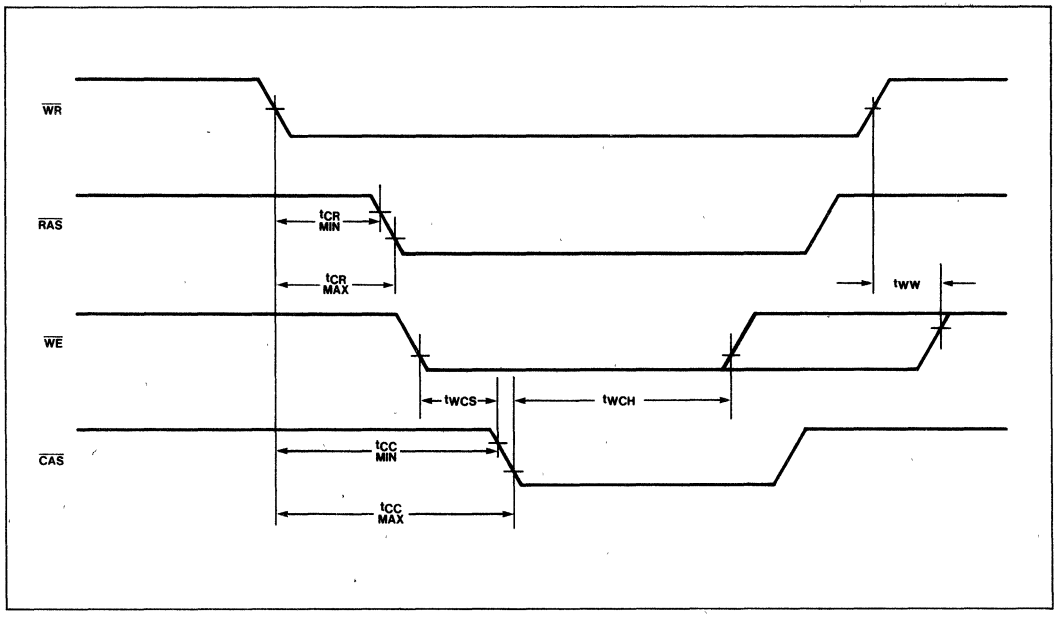


WAVEFORMS (cont'd)

Memory Compatibility Timing

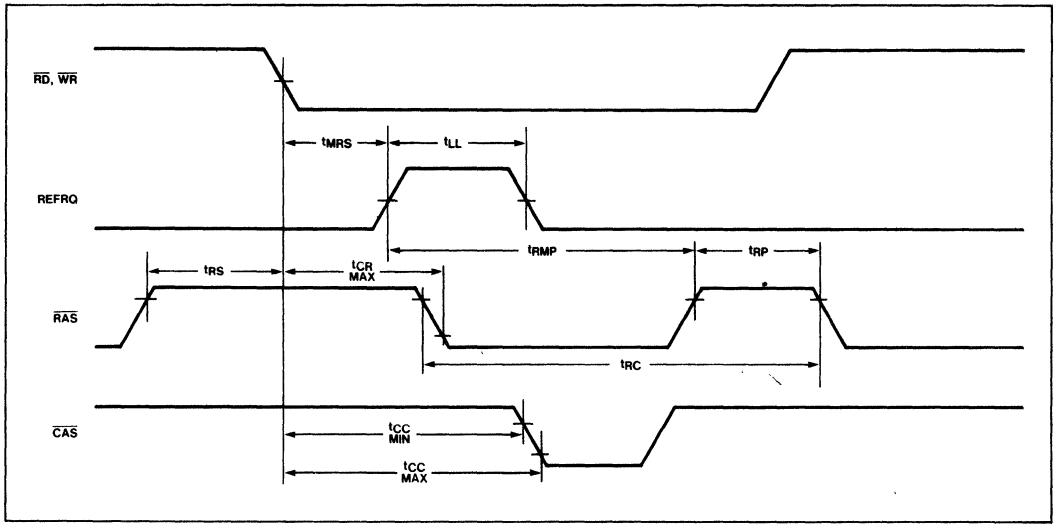


Write Cycle Timing

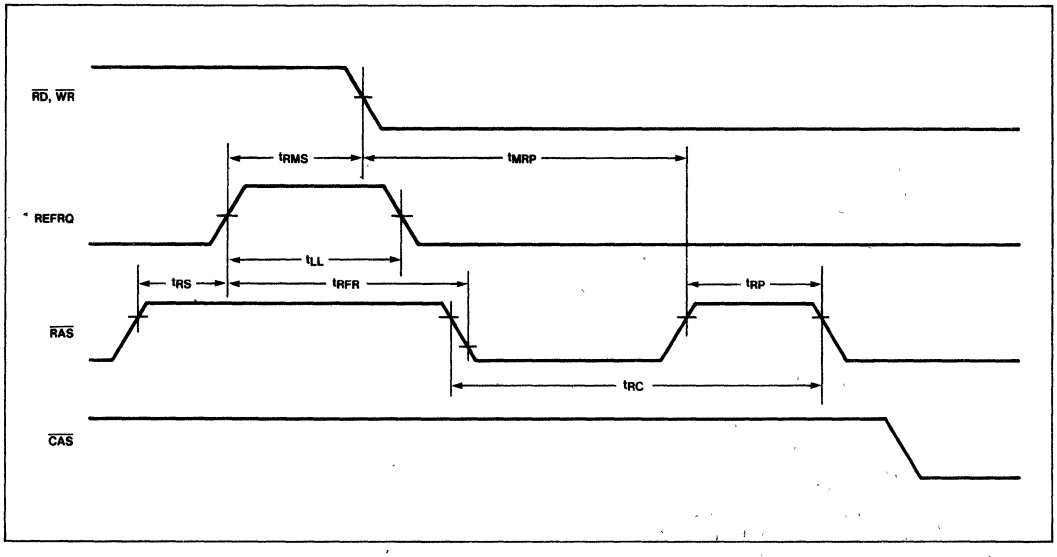


WAVEFORMS (cont'd)

Read or Write Followed By External Refresh



External Refresh Followed By Read or Write



AFN 01838A

WAVEFORMS (cont'd)

Clock And System Timing

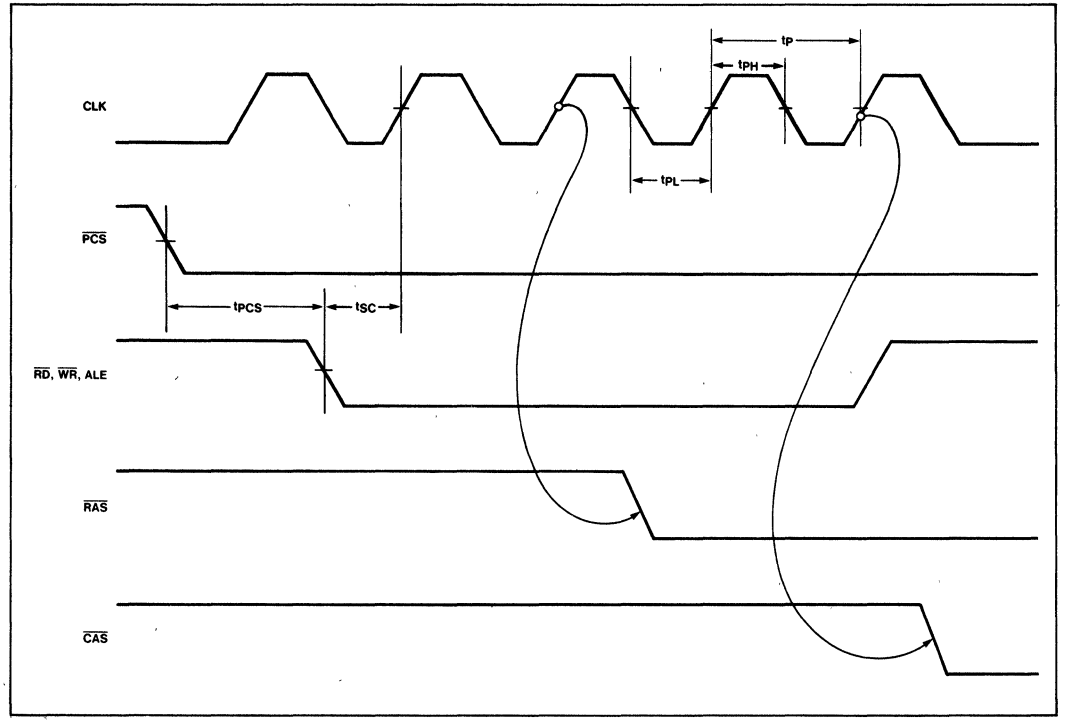


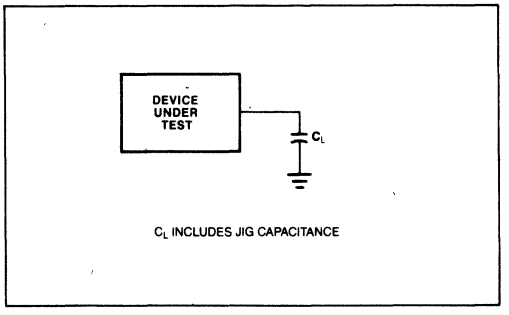
Table 2 8202A Output Test Loading.

Pin	Test Load
SACK, XACK	C_L = 30 pF
OUT_0-OUT_6	C_L = 160 pF
RAS_0-RAS_3	C_L = 60 pF
WE	C_L = 224 pF
CAS	C_L = 320 pF

NOTES:

1. t_{SC} is a reference point only. ALE, RD, WR, and REFRQ inputs do not have to be externally synchronized to 8202A clock.
2. If t_{RS} min and t_{MRS} min are met then, t_{CA}, t_{CR}, and t_{CC} are valid, otherwise t_{CS} is valid.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, and t_{RSH} depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8202A.
4. For back-to-back refresh cycles, t_{RC} max = 13tp
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when RD, WR has already gone high. Otherwise XACK follows RD, WR.
8. WE goes high according to t_{WCH} or t_{WW}, whichever occurs first.

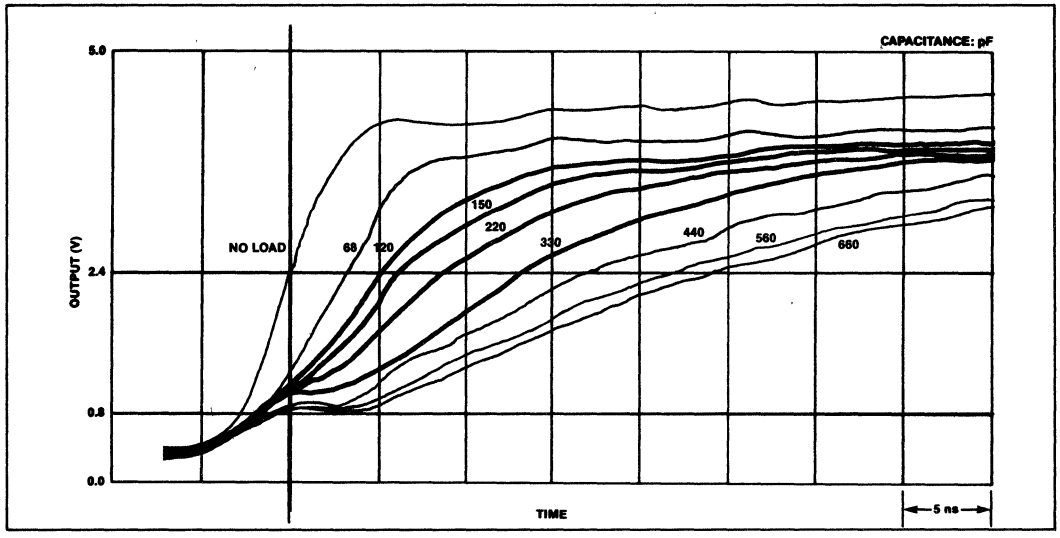
A.C. TESTING LOAD CIRCUIT

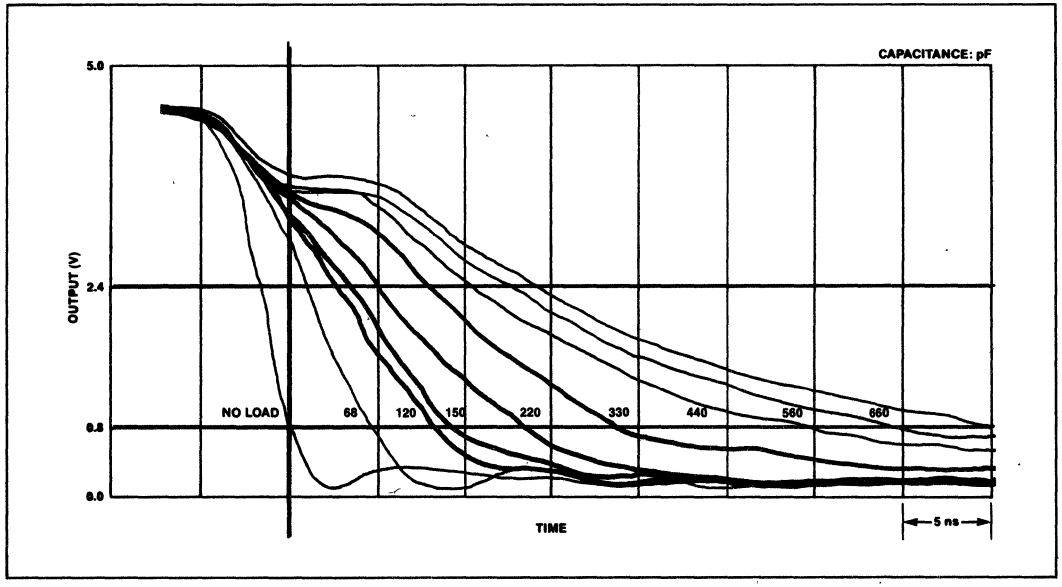


9. t_{CA} applies only when in normal SACK mode.
10. t_{CS} applies only when in delayed SACK mode.
11. t_{CHS} must be met only to ensure a SACK active pulse when in delayed SACK mode. XACK will always be activated for at least t_{XW} (t_P − 25 nS). Violating t_{CHS} min does not otherwise affect device operation.

The typical rising and falling characteristic curves for the $\overline{OUT}$, $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS





NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS:

$T_A = 25°C$
$V_{CC} = +5V$
$t_p = 50$ ns

Pins not measured are loaded with the Test Load capacitance.

AFN-01838A

Example: Find the effect on t_{CR} and t_{CC} using 64 2118 Dynamic RAMs configured in 4 banks.

1. Determine the typical RAS and CAS capacitance: From the data sheet RAS = 4 pF and CAS = 4 pF.
   ∴ RAS load = 64 pF + board capacitance.
   CAS load = 256 pF + board capacitance.
   Assume 2 pF/in (trace length) for board capacitance.
2. From the waveform diagrams, we determine that the falling edge timing is needed for t_{CR} and t_{CC}. Next find the curve that *best* approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.
3. If we use 72 pF for RAS loading, then the t_{CR} (max.) spec should be increased by *about* 1 ns. Similarly if we use 288 pF for CAS, then t_{CC} (min.) and (max.) should decrease about 1 ns.

AFN-01838A

8203
64K DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 64K (2164) and 16K (2117, 2118) Dynamic Memories**
- **Directly Addresses and Drives Up to 64 Devices Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Provides Refresh/Access Arbitration**
- **Internal Clock Capability with the 8203-1 and the 8203-3**
- **Fully Compatible with Intel® 8080A, 8085A, iAPX 88, and iAPX 86 Family Microprocessors**
- **Decodes CPU Status for Advanced Read Capability in 16K mode with the 8203-1 and the 8203-3.**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Refresh Cycles May be Internally or Externally Requested (For Transparent Refresh)**
- **Internal Series Damping Resistors on All RAM Outputs**

The Intel® 8203 is a Dynamic Ram System Controller designed to provide all signals necessary to use 2164, 2118 or 2117 Dynamic RAMs in microcomputer systems. The 8203 provides multiplexed addresses and address strobes, refresh logic, refresh/access arbitration. Refresh cycles can be started internally or externally. The 8203-1 and the 8203-3 support an internal crystal oscillator and Advanced Read Capability. The 8203-3 is a ±5% V_{CC} part.

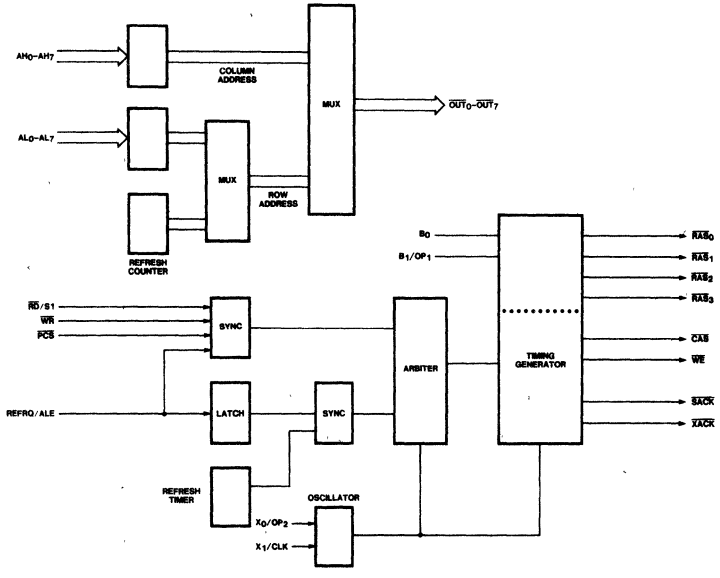


Figure 1. 8203 Block Diagram

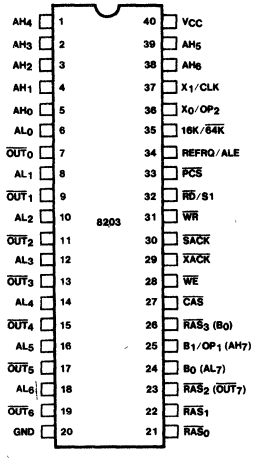


Figure 2. Pin Configuration

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6	6 8 10 12 14 16 18	I I I I I I I	**Address Low:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**Address High:** CPU address inputs used to generate memory column address.
B_0/AL_7 $B_1/OP_1/AH_7$	24 25	I I	**Bank Select Inputs:** Used to gate the appropriate RAS output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode. (Not available in 64K mode.) See Figure 5. When in 64K RAM Mode, pins 24 and 25 operate as the AL_7 and AH_7 address inputs.
$\overline{PCS}$	33	I	**Protected Chip Select:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**Memory Write Request.**
$\overline{RD}/S1$	32	I	**Memory Read Request:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ ALE	34	I	**External Refresh Request:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT_0}$ $\overline{OUT_1}$ $\overline{OUT_2}$ $\overline{OUT_3}$ $\overline{OUT_4}$ $\overline{OUT_5}$ $\overline{OUT_6}$	7 9 11 13 15 17 19	O O O O O O O	**Output of the Multiplexer:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0-7}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**Write Enable:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**Column Address Strobe:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS_0}$ $\overline{RAS_1}$ $\overline{RAS_2}/\overline{OUT_7}$ $\overline{RAS_3}/B_0$	21 22 23 26	O O O I/O	**Row Address Strobe:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8203 Bank Select pins (B_0, B_1/OP_1). In 64K mode, only $\overline{RAS_0}$ and $\overline{RAS_1}$ are available; pin 23 operates as $\overline{OUT_7}$ and pin 26 operates as the B_0 bank select input.
$\overline{XACK}$	29	O	**Transfer Acknowledge:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.
$\overline{SACK}$	30	O	**System Acknowledge:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
X_0/OP_2 X_1/CLK	36 37	I/O I/O	**Oscillator Inputs:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is shorted to pin 40 (V_{CC}) or if X_0/OP_2 is connected to +12V through a 1KΩ resistor then X_1/CLK becomes a TTL input for an external clock. (Note: Crystal mode for the 8203-1 and the 8203-3 only).
$16K/\overline{64K}$	35	I	**Mode Select:** This input selects 16K mode (2117, 2118) or 64K mode (2164). Pins 23-26 change function based on the mode of operation.
V_{CC}	40		**Power Supply:** +5V.
GND	20		**Ground.**

Functional Description

The 8203 provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2164, 2118 and 2117 dynamic RAMs.

The 8203 has two modes, one for 16K dynamic RAMs and one for 64Ks, controlled by pin 35.

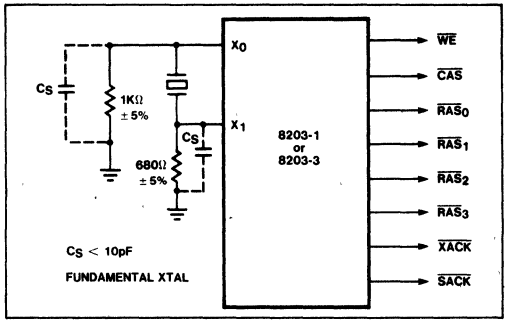


Figure 3. Crystal Operation for the 8203-1 and 8203-3

All 8203 timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the trailing edges of the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$.

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The Advanced-Read mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal timer which generates refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 8203 clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

16K/$\overline{64K}$ Option Selection

Pin 35 is a strap input that controls the two 8203 modes. Figure 4 shows the four pins that are multiplexed. In 16K mode (pin 35 tied to V_{CC} or left open), the 8203 has two Bank Select inputs to select one of four $\overline{RAS}$ outputs. In this mode, the 8203 is exactly compatible with the Intel 8202A Dynamic RAM Controller. In 64K mode (pin 35 tied to GND), there is only one Bank Select input (pin 26) to select the two $\overline{RAS}$ outputs. More than two banks of 64K dynamic RAM's can be used with external logic.

Other Option Selections

The 8203 has three strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a $\overline{RD}$ input to an S1 input, and pin 34 changes from a REFRQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor on the 8203-1 or 8203-3 only.

When OP_2 is selected, the internal oscillator is disabled and pin 37 changes from a crystal input (X_1) to a CLK input for an external TTL clock. OP_2 is selected by shorting pin 36 (X_0/OP_2) directly to pin 40 (V_{CC}). No current limiting resistor should be used. OP_2 may also be selected by tying pin 36 to +12V through a 1KΩ resistor.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Pin #	16K Function	64K Function
23	$\overline{RAS}_2$	Address Output ($\overline{OUT}_7$)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS}_3$	Bank Select (B_0)

Figure 4. 16K/$\overline{64K}$ Mode Selection

	Inputs		Outputs			
	B_1	B_0	$\overline{RAS}_0$	$\overline{RAS}_1$	$\overline{RAS}_2$	$\overline{RAS}_3$
16K Mode	0	0	0	1	1	1
	0	1	1	0	1	1
	1	0	1	1	0	1
	1	1	1	1	1	0
64K Mode	—	0	0	1	—	—
	—	1	1	0	—	—

Figure 5. Bank Selection

Description	Pin #	Normal Function	Option Function
B1/OP1 (16K only)/AH_7	25	Bank (RAS) Select	Advanced-Read Mode (8203-1, -3)
X_0/OP_2	36	Crystal Oscillator (8203-1 and 8203-3)	External Oscillator

Figure 6. 8203 Option Selection

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_7 are gated to $\overline{OUT}_0$–$\overline{OUT}_7$, then AH_0–AH_7 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_7$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 8203 has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE input allows refresh requests to be requested external to the 8203.

All three of these inputs may be asynchronous with respect to the 8203's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 8203 is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 8203 is normally in the IDLE state. Whenever one of the other cycles is requested, the 8203 will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 8203 will return to the IDLE state.

Test Cycle

The TEST Cycle is used to check operation of several 8203 internal functions. TEST cycles are requested by activating the $\overline{PCS}$, $\overline{RD}$ and $\overline{WR}$ inputs. The TEST Cycle will reset the refresh address counter and perform a WRITE Cycle. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 8203 has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 8203 refresh cycles activate all of the $\overline{RAS}$ outputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 8203 clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds (128 cycles) or every 4 milliseconds (256 cycles). If REFRQ is inactive, the refresh timer will request a refresh cycle every 10-16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 8203 clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 8203 is not in the middle of a cycle.

When the 8203 is in the idle state a simultaneous memory request and external refresh request will result in the memory request being honored first. This 8203 characteristic can be used to "hide" refresh cycles during system operation. A circuit similar to Figure 7 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 8203 performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 8203 can complete the refresh cycle before the next memory request is generated.

If the 8203 is not in the idle state then a simultaneous memory request and an external refresh request may result in the refresh request being honored first.

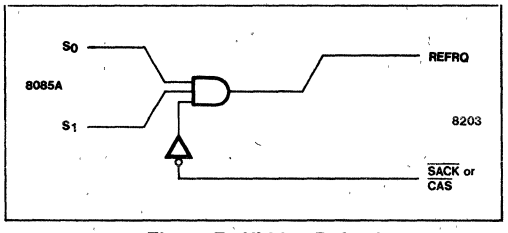


Figure 7. Hidden Refresh

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

Read Cycles

The 8203 can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs (16K mode only)

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B1 input	OP_1 (+12V)
Pin 32	RD input	S1 input
Pin 34	REFRQ input	ALE input
# RAM banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh	Yes	No

Figure 8. 8203 Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 8203 responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similiar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 8203 will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 8203 will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-SACK latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similiar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 8203 Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests.

In order to minimize propagation delay, the 8203 uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The t_{AD} AC parameter should be used for this system calculation.

The B_0-B_1 inputs are similiar to the address inputs in that they are not latched. B_0 and B_1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 8203 uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 8203 synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 8203.

A microprocessor system is concerned when the data is valid after $\overline{RD}$ goes low. See Figure 9. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (t_{RAC}) and the CAS-access time (t_{CAC}). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable $t_{CC,max}$ (8203) + t_{CAC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 8203 normally performs "early-write" cycles, the data must be stable at the RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the $\overline{WR}$ input signal or delay the 8203 $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 8203 timing, including the READY handshake signals, $\overline{SACK}$ and $\overline{XACK}$, which

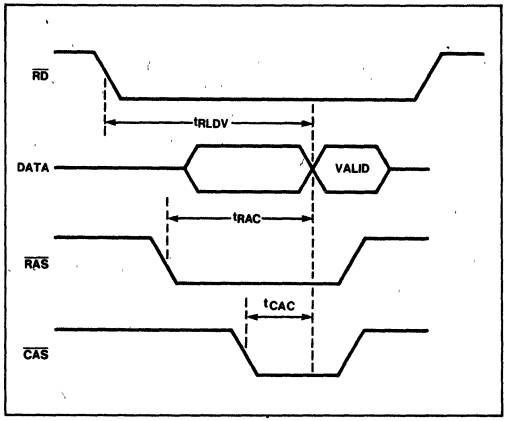


Figure 9. Read Access Time

may increase the number of WAIT states generated by the CPU.

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the CAS active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

The $\overline{RAS}_{0-3}$, $\overline{CAS}$, $\overline{OUT}_{0-7}$, and $\overline{WE}$ outputs contain on-chip series damping resistors (typically 20Ω) to minimize overshoot.

Some dynamic RAMs require more than 2.4V V_{IH}. Noise immunity may be improved for these RAMs by adding pull-up resistors to the 8203's outputs. Intel RAMs do not require pull-up resistors.

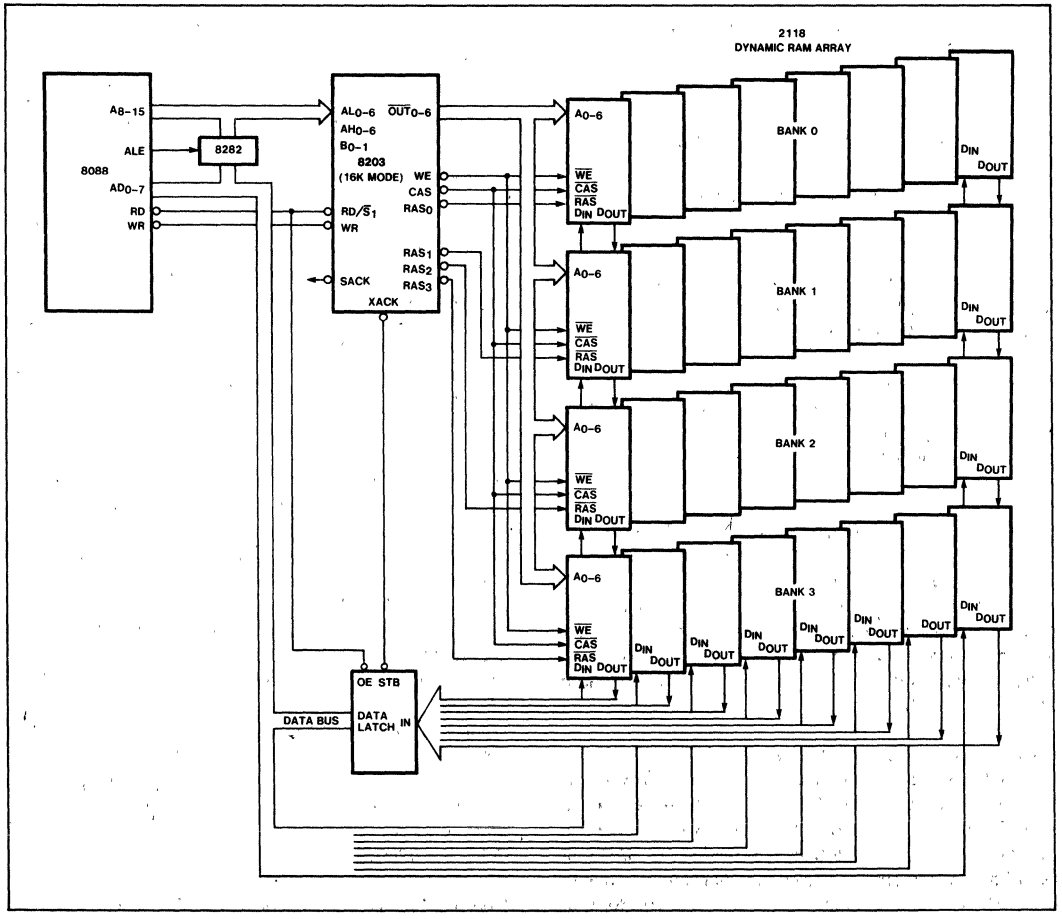


Figure 10. Typical 8088 System

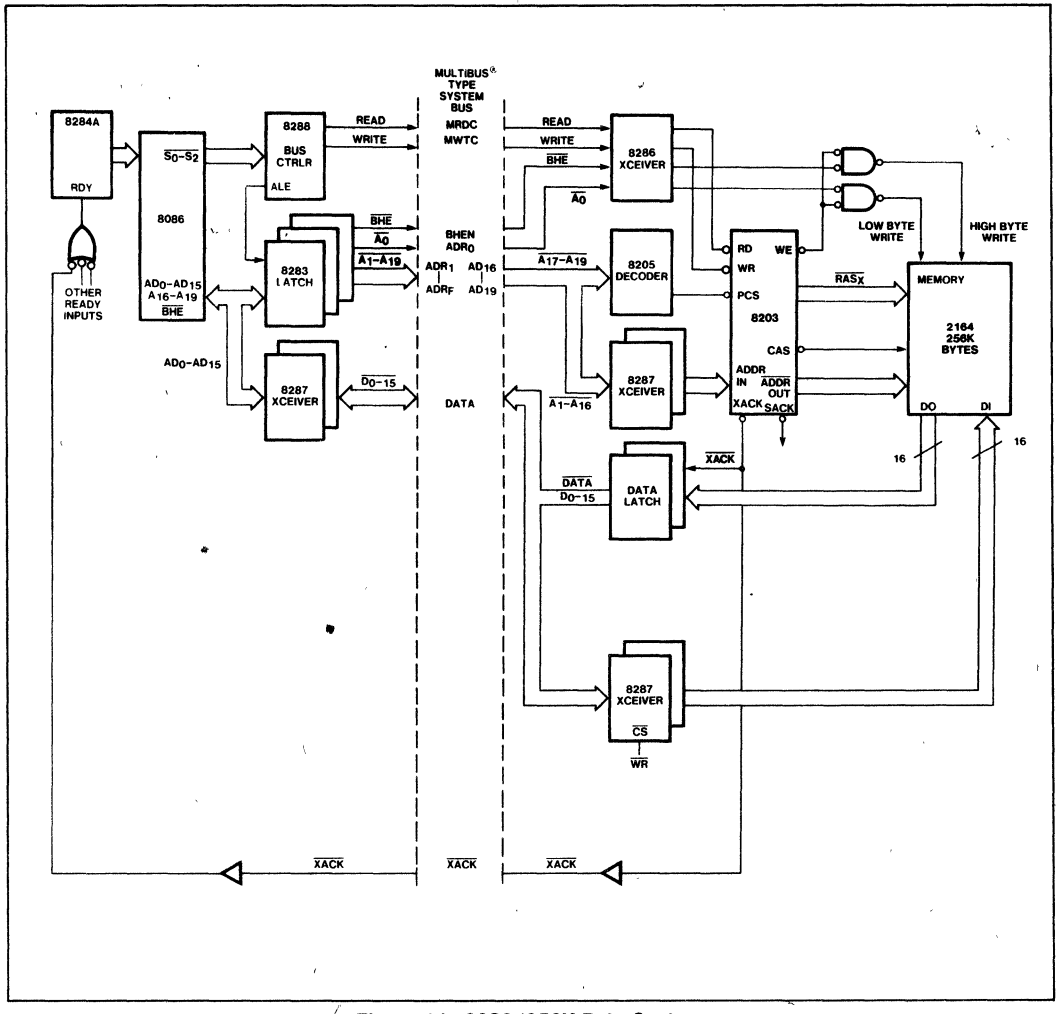


Figure 11. 8086/256K Byte System

ABSOLUTE MAXIMUM RATINGS*

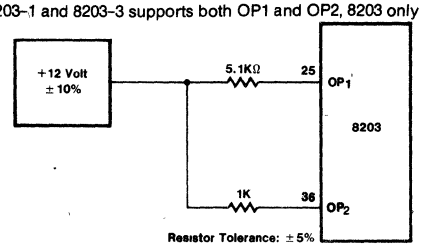
Ambient Temperature Under Bias .....0°C to 70°C
Storage Temperature .....−65°C to +150°C
Voltage On any Pin
With Respect to Ground .....−0.5V to +7V[4]
Power Dissipation .....1.6 Watts

**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C; V_{CC} = 5.0V ± 10% (5.0V ± 5% for 8203-3); GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		290	mA	
I_F	Forward Input Current				
	CLK, 64K/16K Mode select		−2.0	mA	V_F = 0.45V
	All Other Inputs[3]		−320	μA	V_F = 0.45V
I_R	Reverse Input Current[3]		40	μA	$V_R = V_{CC}$; Note 1
V_{OL}	Output Low Voltage				
	$\overline{SACK}$, $\overline{XACK}$		0.45	V	I_{OL} = 5 mA
	All Other Outputs		0.45	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage				V_{IL} = 0.65 V
	$\overline{SACK}$, $\overline{XACK}$	2.4		V	I_{OH} = −1 mA
	All Other Outputs	2.6		V	I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	V_{CC} = 5.0V (Note 2)
V_{IH1}	Input High Voltage	2.0	V_{CC}	V	V_{CC} = 5.0V
V_{IH2}	Option Voltage		V_{CC}	V	(Note 4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTES:
1. I_R = 200 μA for pin 37 (CLK).
2. For test mode $\overline{RD}$ & $\overline{WR}$ must be held at GND.
3. Except for pin 36 in XTAL mode.
4. 8203-1 and 8203-3 supports both OP1 and OP2, 8203 only supports OP2.



A.C. CHARACTERISTICS

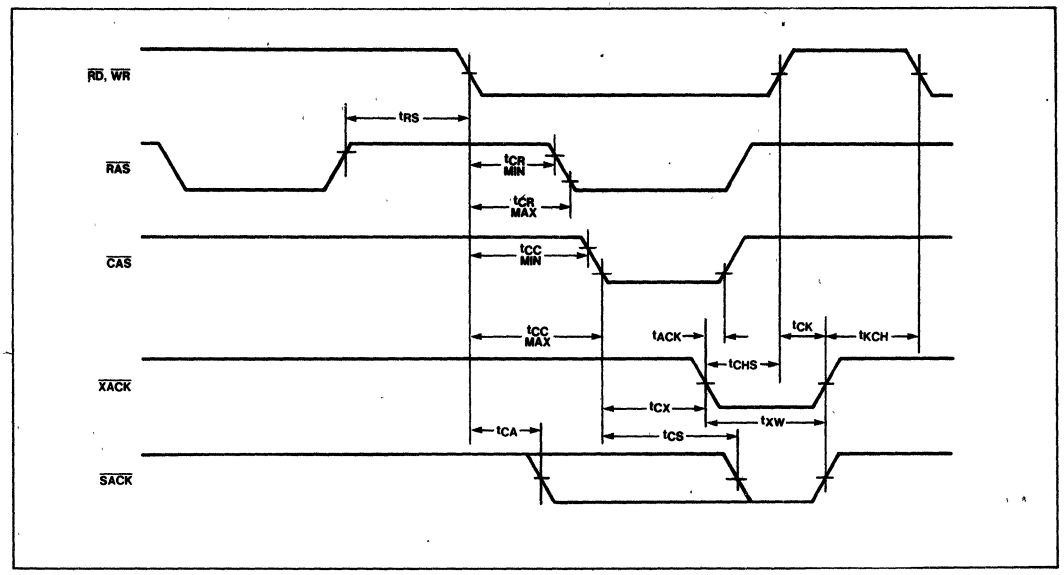
T_J = 0°C to 70°C; V_{CC} = 5V ± 10% (5.0V ± 5% for 8203-3); GND = 0V

Measurements made with respect to $\overline{RAS}_0$-$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$-$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

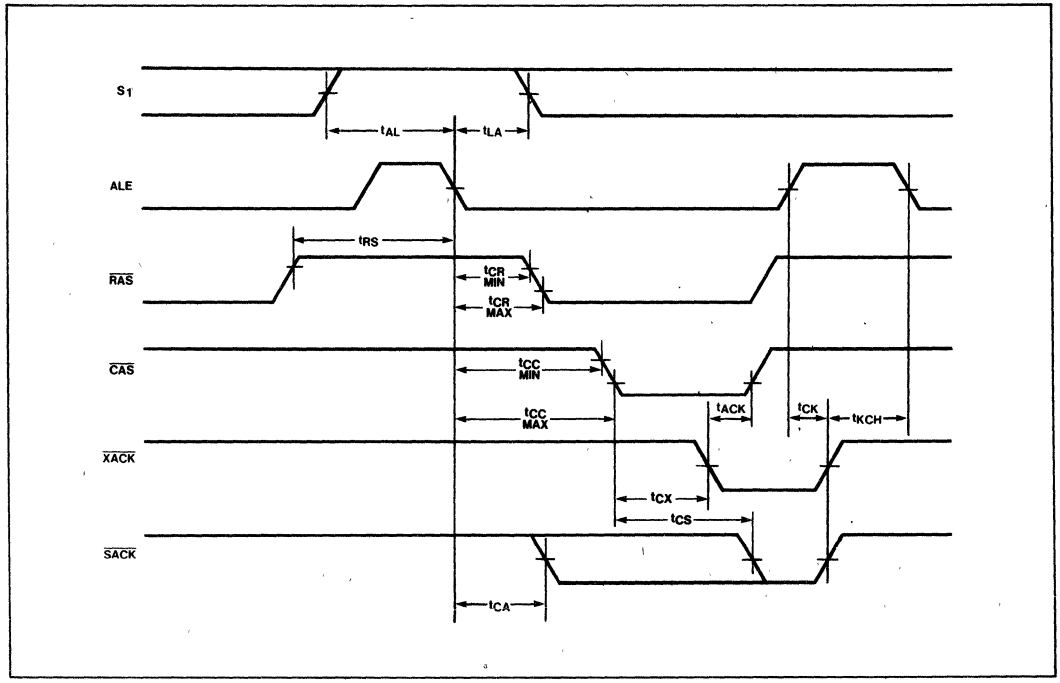
Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	40	54	
t_{PH}	External Clock High Time	20		
t_{PL}	External Clock Low Time—above (>) 20 mHz	17		
t_{PL}	External Clock Low Time—below (≤) 20 mHz	20		
t_{RC}	Memory Cycle Time	10tp − 30	12tp	4, 5
t_{REF}	Refresh Time (128 cycles)	264tp	288tp	
t_{RP}	$\overline{RAS}$ Precharge Time	4tp − 30		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	5tp − 30		3
t_{ASR}	Address Setup to $\overline{RAS}$	tp − 30		3
t_{RAH}	Address Hold From $\overline{RAS}$	tp − 10		3
t_{ASC}	Address Setup to $\overline{CAS}$	tp − 30		3
t_{CAH}	Address Hold from $\overline{CAS}$	5tp − 20		3
t_{CAS}	$\overline{CAS}$ Pulse Width	5tp − 10		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	tp − 40		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	5tp − 35		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ delay from $\overline{RAS}$	5tp		2, 6
t_{MRP}	$\overline{RD}$, $\overline{WR}$ setup to $\overline{RAS}$	0		5
t_{RMS}	REFRQ setup to $\overline{RD}$, $\overline{WR}$	2tp		6
t_{RMP}	REFRQ setup to $\overline{RAS}$	2tp		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold from ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	tp + 30	2tp + 70	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	3tp + 25	4tp + 85	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ setup to REFRQ	5		2
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay		2tp + 47	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	5tp − 25	5tp + 20	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	5tp − 25	5tp + 40	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	tp − 25		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ turn-off Delay		35	
t_{KCH}	CMD Inactive Hold after $\overline{SACK}$, $\overline{XACK}$	10		
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		4tp + 100	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	40	3

WAVEFORMS

Normal Read or Write Cycle

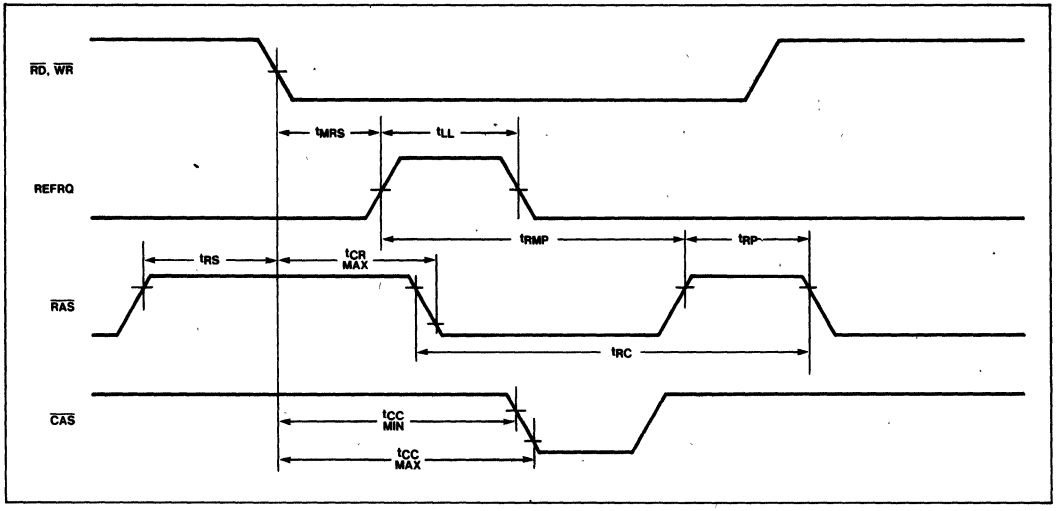


Advanced Read Mode

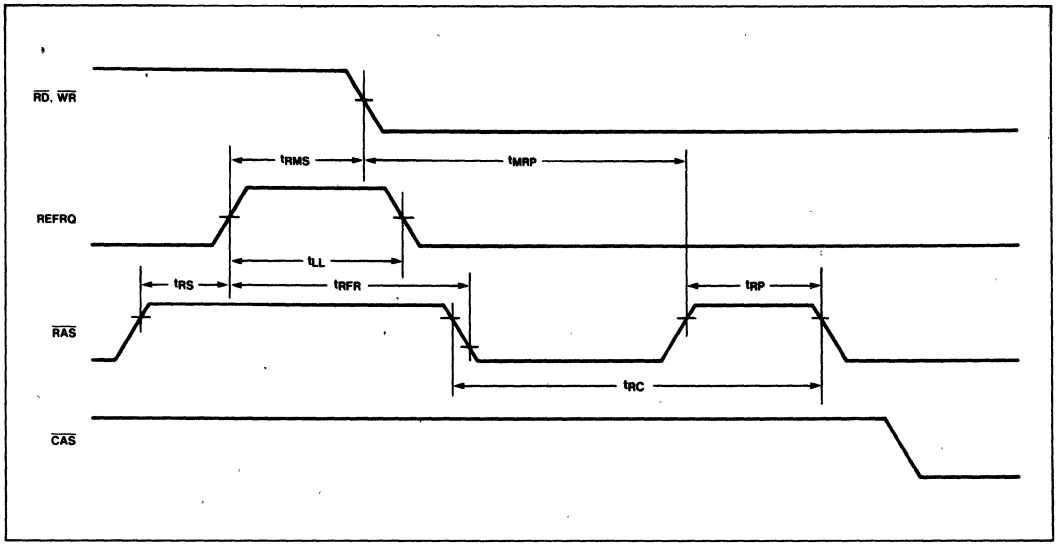


WAVEFORMS (cont'd)

Read or Write Followed By External Refresh



External Refresh Followed By Read or Write



WAVEFORMS (cont'd)

Clock And System Timing

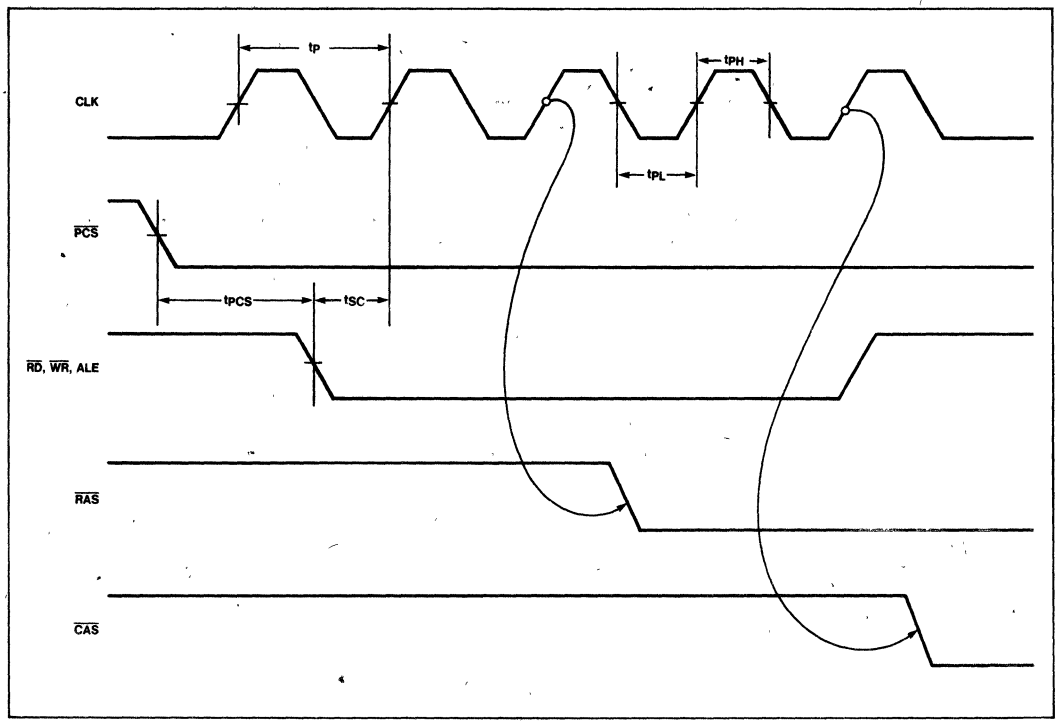


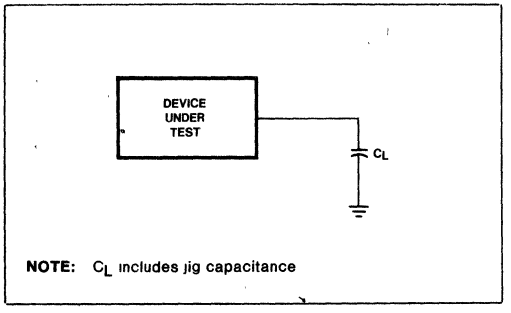
Table 2. 8203 Output Loading. All specifications are for the Test Load unless otherwise noted.

Pin	Test Load
$\overline{SACK}$, $\overline{XACK}$	C_L = 30 pF
$\overline{OUT}_0$-$\overline{OUT}_6$	C_L = 160 pF
$\overline{RAS}_0$-$\overline{RAS}_3$	C_L = 60 pF
$\overline{WE}$	C_L = 224 pF
$\overline{CAS}$	C_L = 320 pF

NOTES:

1. t_{SC} is a reference point only. ALE, $\overline{RD}$, $\overline{WR}$, and REFRQ inputs do not have to be externally synchronized to 8203 clock.
2. If t_{RS} min and t_{MRS} min are met then t_{CA}, t_{CR}, and t_{CC} are valid, otherwise t_{CS} is valid.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, and t_{RSH} depend upon B0-B1 and CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 8203.
4. For back-to-back refresh cycles, t_{RC} max = 13tp
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when $\overline{RD}$, $\overline{WR}$ has already gone high. Otherwise $\overline{XACK}$ follows $\overline{RD}$, $\overline{WR}$.
8. $\overline{WE}$ goes high according to t_{WCH} or t_{WW}, whichever occurs first.

A.C. TESTING LOAD CIRCUIT

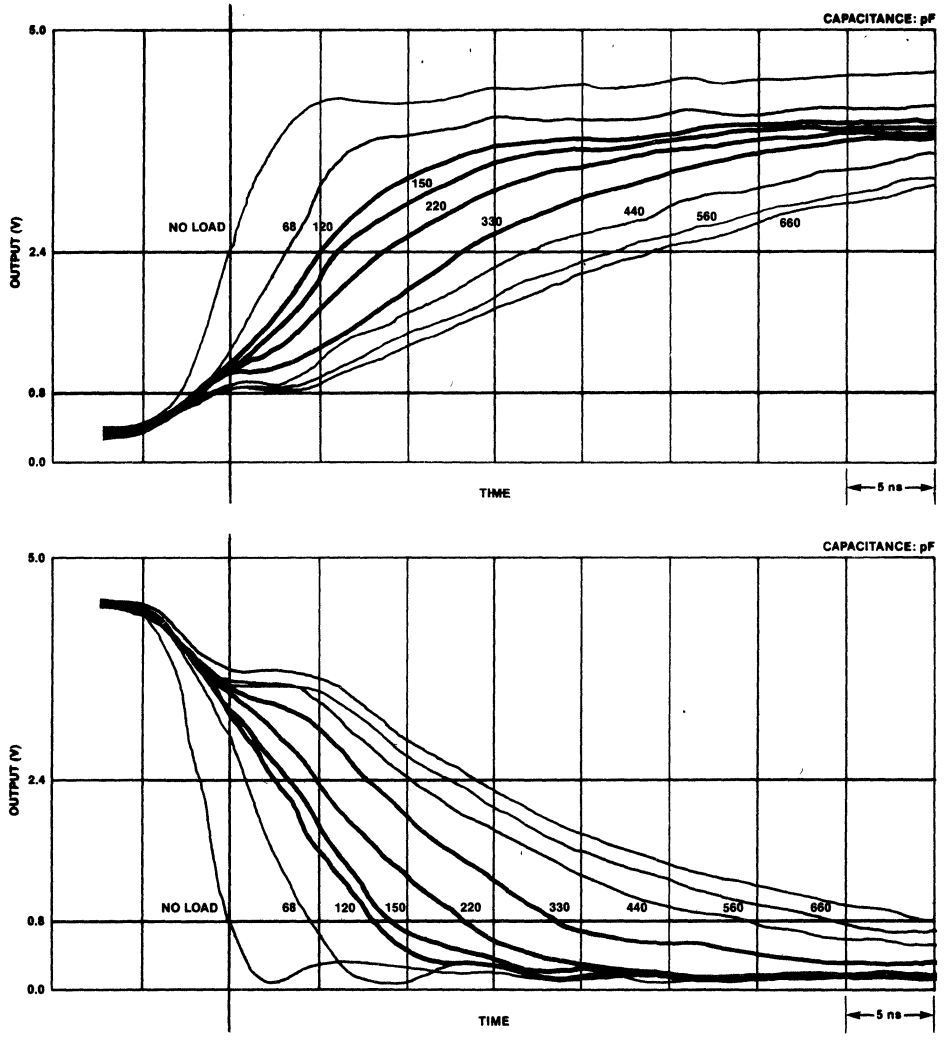


NOTE: C_L includes jig capacitance

9. t_{CA} applies only when in normal $\overline{SACK}$ mode.de.
10. t_{CS} applies only when in delayed $\overline{SACK}$ mode.
11. t_{CHS} must be be met only to ensure a $\overline{SACK}$ active pulse when in delayed $\overline{SACK}$ mode. $\overline{XACK}$ will always be activated for at least t_{XW} (t_P−25 nS). Violating t_{CHS} min does not otherwise affect device operation.

The typical rising and falling characteristic curves for the $\overline{OUT}$, $\overline{RAS}$, $\overline{CAS}$ and $\overline{WE}$ output buffers can be used to determine the effects of capacitive loading on the A.C. Timing Parameters. Using this design tool in conjunction with the timing waveforms, the designer can determine typical timing shifts based on system capacitive load.

A.C. CHARACTERISTICS FOR DIFFERENT CAPACITIVE LOADS



NOTE:
Use the Test Load as the base capacitance for estimating timing shifts for system critical timing parameters.

MEASUREMENT CONDITIONS:

T_A = 25°C
V_{CC} = +5V
t_p = 50 ns

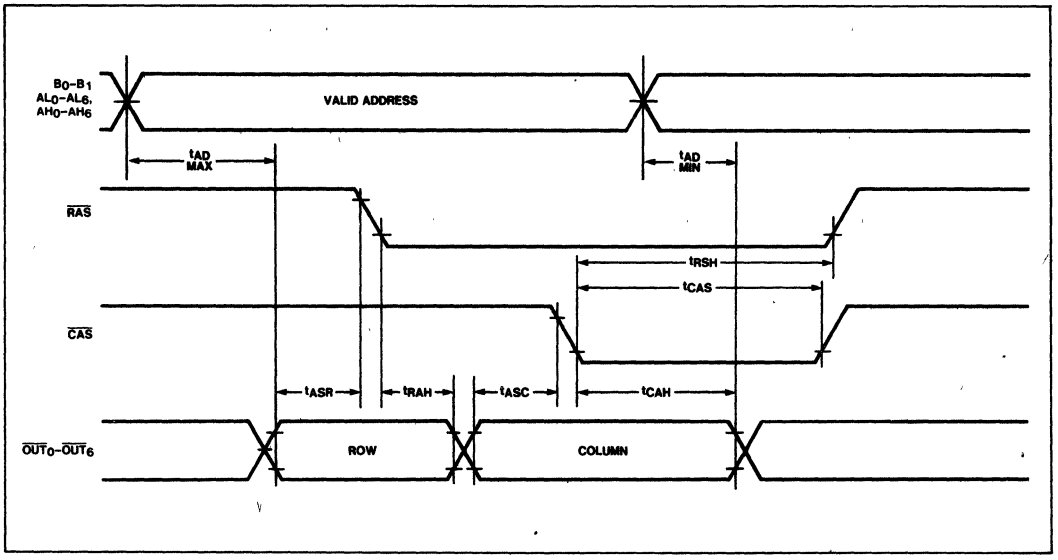
Pins not measured are loaded with the Test Load capacitance

Example: Find the effect on t_{CR} and t_{CC} using 32 2164 Dynamic RAMs configured in 2 banks.

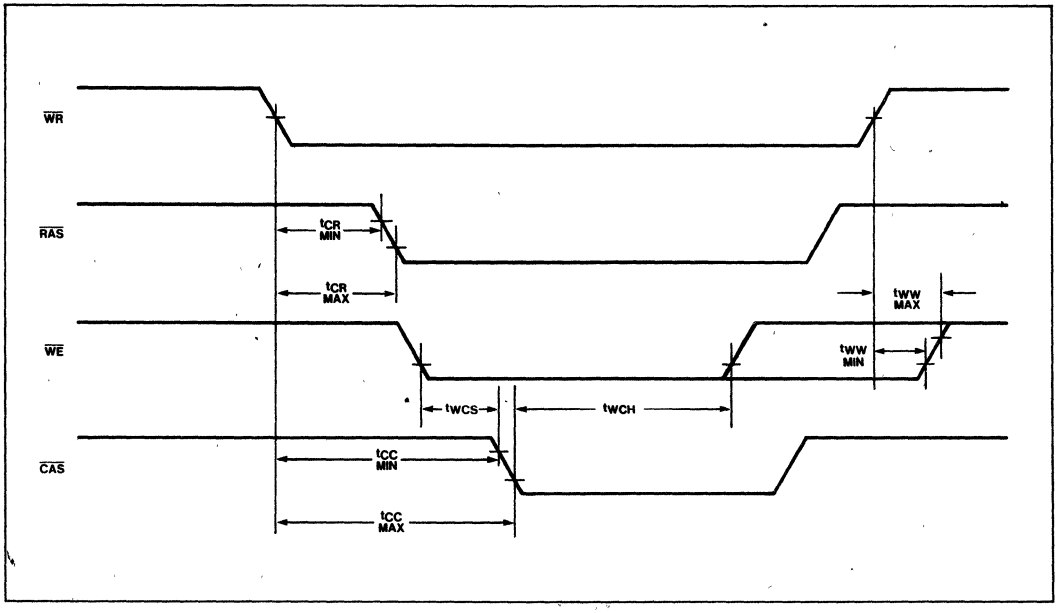
1. Determine the typical RAS and CAS capacitance:
   From the data sheet RAS = 5 pF and CAS = 5 pF.
   ∴ RAS load = 80 pF + board capacitance.
   CAS load = 160 pF + board capacitance.
   Assume 2 pF/in (trace length) for board capacitance and for this example 4 inches for RAS and 8 inches for CAS.
2. From the waveform diagrams, we determine that the falling edge timing is needed for t_{CR} and t_{CC}. Next find the curve that *best* approximates the test load; i.e., 68 pF for RAS and 330 pF for CAS.
3. If we use 88 pF for RAS loading, then t_{CR} (min.) spec should be increased by about 1 ns, and t_{CR} (max.) spec should be increased by *about* 2 ns. Similarly if we use 176 pF for CAS, then t_{CC} (min.) should decrease by 3 ns and t_{CC} (max.) should decrease by about 7 ns.

WAVEFORMS (cont'd)

Memory Compatibility Timing



Write Cycle Timing



82C03
CMOS 64K DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to NMOS (2164A) and CMOS Control (51C64) 64K Dynamic Memories**
- **Directly Addresses and Drives Up to 64 Devices Without External Drivers**
- **Provides Address Multiplexing and Strobes**
- **Provides a Refresh Timer and a Refresh Counter**
- **Provides Refresh/Access Arbitration**
- **Internal Clock Capability**
- **Provides System Acknowledge and Transfer Acknowledge Signals**
- **Refresh Cycles May be Internally or Externally Requested (For Transparent Refresh)**
- **Internal Series Damping Resistors on All RAM Outputs**

The Intel® 82C03 is a CMOS Dynamic Ram System Controller designed to provide all signals necessary to use 51C64 CMOS Dynamic RAMs in microcomputer systems. The 82C03 provides multiplexed addresses and address strobes, refresh logic, refresh/access arbitration. Refresh cycles can be started internally or externally. The 82C03 supports an internal crystal oscillator.

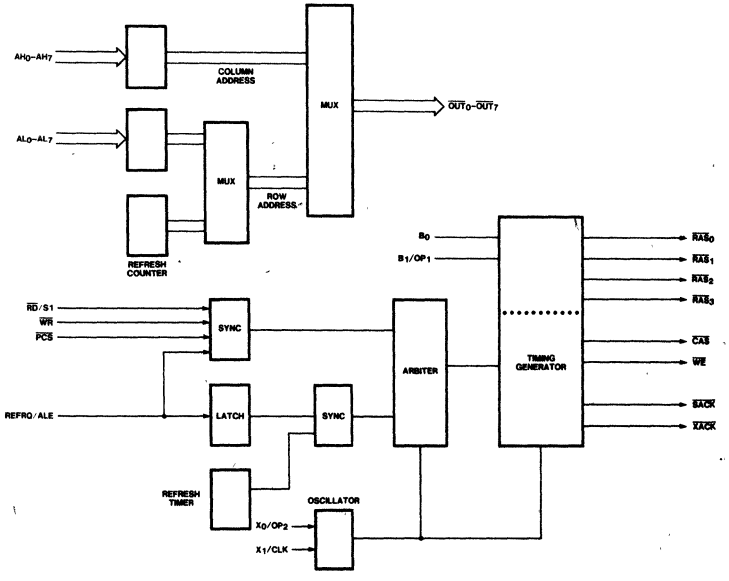


Figure 1. 82C03 Block Diagram

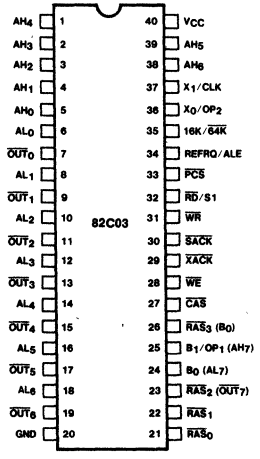


Figure 2. Pin Configuration

Table 1. Pin Descriptions

Symbol	Pin No.	Type	Name and Function
AL_0 AL_1 AL_2 AL_3 AL_4 AL_5 AL_6	6 8 10 12 14 16 18	I I I I I I I	**Address Low:** CPU address inputs used to generate memory row address.
AH_0 AH_1 AH_2 AH_3 AH_4 AH_5 AH_6	5 4 3 2 1 39 38	I I I I I I I	**Address High:** CPU address inputs used to generate memory column address.
B_0/AL_7 $B_1/OP_1/AH_7$	24 25	I I	**Bank Select Inputs:** Used to gate the appropriate RAS output for a memory cycle. B_1/OP_1 option used to select the Advanced Read Mode. (Not available in 64K mode.) See Figure 5. When in 64K RAM Mode, pins 24 and 25 operate as the AL_7 and AH_7 address inputs.
$\overline{PCS}$	33	I	**Protected Chip Select:** Used to enable the memory read and write inputs. Once a cycle is started, it will not abort even if $\overline{PCS}$ goes inactive before cycle completion.
$\overline{WR}$	31	I	**Memory Write Request.**
$\overline{RD}/S1$	32	I	**Memory Read Request:** S1 function used in Advanced Read mode selected by OP_1 (pin 25).
REFRQ/ ALE	34	I	**External Refresh Request:** ALE function used in Advanced Read mode, selected by OP_1 (pin 25).
$\overline{OUT_0}$ $\overline{OUT_1}$ $\overline{OUT_2}$ $\overline{OUT_3}$ $\overline{OUT_4}$ $\overline{OUT_5}$ $\overline{OUT_6}$	7 9 11 13 15 17 19	O O O O O O O	**Output of the Multiplexer:** These outputs are designed to drive the addresses of the Dynamic RAM array. (Note that the $\overline{OUT}_{0-7}$ pins do not require inverters or drivers for proper operation.)
$\overline{WE}$	28	O	**Write Enable:** Drives the Write Enable inputs of the Dynamic RAM array.
$\overline{CAS}$	27	O	**Column Address Strobe:** This output is used to latch the Column Address into the Dynamic RAM array.
$\overline{RAS_0}$ $\overline{RAS_1}$ $\overline{RAS_2}/\overline{OUT_7}$ $\overline{RAS_3}/B_0$	21 22 23 26	O O O I/O	**Row Address Strobe:** Used to latch the Row Address into the bank of dynamic RAMs, selected by the 8203 Bank Select pins (B_0, B_1/OP_1). In 64K mode, only $\overline{RAS_0}$ and $\overline{RAS_1}$ are available; pin 23 operates as $\overline{OUT_7}$ and pin 26 operates as the B_0 bank select input.
$\overline{XACK}$	29	O	**Transfer Acknowledge:** This output is a strobe indicating valid data during a read cycle or data written during a write cycle. $\overline{XACK}$ can be used to latch valid data from the RAM array.
$\overline{SACK}$	30	O	**System Acknowledge:** This output indicates the beginning of a memory access cycle. It can be used as an advanced transfer acknowledge to eliminate wait states. (Note: If a memory access request is made during a refresh cycle, $\overline{SACK}$ is delayed until $\overline{XACK}$ in the memory access cycle).
X_0/OP_2 X_1/CLK	36 37	I/O I/O	**Oscillator Inputs:** These inputs are designed for a quartz crystal to control the frequency of the oscillator. If X_0/OP_2 is left open then X_1/CLK becomes a TTL input for an external clock. (Note: Crystal mode for the 82C03-1 only).
$16K/\overline{64K}$	35	I	**Mode Select:** This input selects 16K mode (2117, 2118) or 64K mode (2164). Pins 23-26 change function based on the mode of operation.
V_{CC}	40		**Power Supply:** +5V.
GND	20		**Ground.**

Functional Description

The 82C03 provides a complete dynamic RAM controller for microprocessor systems as well as expansion memory boards. All of the necessary control signals are provided for 2164A and 51C64 64K dynamic RAMs. As well as 16K dynamic RAMs.

The 82C03 has two modes, one for 16K dynamic RAMs and one for 64Ks, controlled by pin 35.

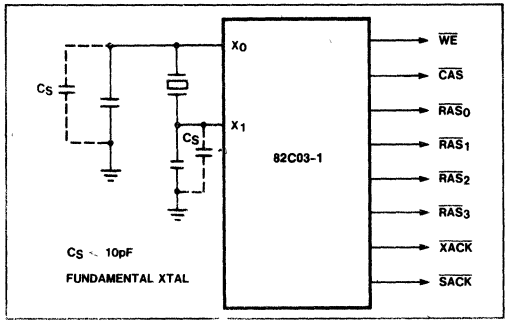


Figure 3. Crystal Operation for the 82C03-1

All 8203 timing is generated from a single reference clock. This clock is provided via an external oscillator or an on-chip crystal oscillator. All output signal transitions are synchronous with respect to this clock reference, except for the trailing edges of the CPU handshake signals $\overline{SACK}$ and $\overline{XACK}$.

CPU memory requests normally use the $\overline{RD}$ and $\overline{WR}$ inputs. The Advanced-Read mode allows ALE and S1 to be used in place of the $\overline{RD}$ input.

Failsafe refresh is provided via an internal timer which generates refresh requests. Refresh requests can also be generated via the REFRQ input.

An on-chip synchronizer/arbiter prevents memory and refresh requests from affecting a cycle in progress. The READ, WRITE, and external REFRESH requests may be asynchronous to the 82C03 clock; on-chip logic will synchronize the requests, and the arbiter will decide if the requests should be delayed, pending completion of a cycle in progress.

16K/$\overline{64K}$ Option Selection

Pin 35 is a strap input that controls the two 82C03 modes. Figure 4 shows the four pins that are multiplexed. In 16K mode (pin 35 tied to V_{CC} or left open), the 82C03 has two Bank Select inputs to select one of four $\overline{RAS}$ outputs. In this mode, the 82C03 is exactly compatible with the Intel 8202A Dynamic RAM Controller. In 64K mode (pin 35 tied to GND), there is only one Bank Select input (pin 26) to select the two $\overline{RAS}$ outputs. More than two banks of 64K dynamic RAM's can be used with external logic.

Other Option Selections

The 82C03 has three strapping options. When OP_1 is selected (16K mode only), pin 32 changes from a $\overline{RD}$ input to an S1 input, and pin 34 changes from a REFRQ input to an ALE input. See "Refresh Cycles" and "Read Cycles" for more detail. OP_1 is selected by tying pin 25 to +12V through a 5.1K ohm resistor.

When OP_2 is selected, the internal oscillator is disabled and pin 37 changes from a crystal input (X_1) to a CLK input for an external TTL clock. OP_2 is selected by leaving pin 36 (X_0/OP_2) open.

Refresh Timer

The refresh timer is used to monitor the time since the last refresh cycle occurred. When the appropriate amount of time has elapsed, the refresh timer will request a refresh cycle. External refresh requests will reset the refresh timer.

Refresh Counter

The refresh counter is used to sequentially refresh all of the memory's rows. The 8-bit counter is incremented after every refresh cycle.

Pin #	16K Function	64K Function
23	$\overline{RAS_2}$	Address Output ($\overline{OUT_7}$)
24	Bank Select (B_0)	Address Input (AL_7)
25	Bank Select (B_1)	Address Input (AH_7)
26	$\overline{RAS_3}$	Bank Select (B_0)

Figure 4. 16K/$\overline{64K}$ Mode Selection

	Inputs		Outputs			
	B_1	B_0	$\overline{RAS_0}$	$\overline{RAS_1}$	$\overline{RAS_2}$	$\overline{RAS_3}$
16K Mode	0	0	0	1	1	1
	0	1	1	0	1	1
	1	0	1	1	0	1
	1	1	1	1	1	0
64K Mode	—	0	0	1	—	—
	—	1	1	0	—	—

Figure 5. Bank Selection

Description	Pin #	Normal Function	Option Function
B1/OP1 (16K only)/AH_7	25	Bank (RAS) Select	Advanced-Read Mode
X_0/OP_2	36	Crystal Oscillator	External Oscillator

Figure 6. 8203 Option Selection

Address Multiplexer

The address multiplexer takes the address inputs and the refresh counter outputs, and gates them onto the address outputs at the appropriate time. The address outputs, in conjunction with the $\overline{RAS}$ and $\overline{CAS}$ outputs, determine the address used by the dynamic RAMs for read, write, and refresh cycles. During the first part of a read or write cycle, AL_0–AL_7 are gated to $\overline{OUT}_0$–$\overline{OUT}_7$, then AH_0–AH_7 are gated to the address outputs.

During a refresh cycle, the refresh counter is gated onto the address outputs. All refresh cycles are RAS-only refresh ($\overline{CAS}$ inactive, $\overline{RAS}$ active).

To minimize buffer delay, the information on the address outputs is inverted from that on the address inputs.

$\overline{OUT}_0$–$\overline{OUT}_7$ do not need inverters or buffers unless additional drive is required.

Synchronizer/Arbiter

The 82C03 has three inputs, REFRQ/ALE (pin 34), $\overline{RD}$ (pin 32) and $\overline{WR}$ (pin 31). The $\overline{RD}$ and $\overline{WR}$ inputs allow an external CPU to request a memory read or write cycle, respectively. The REFRQ/ALE input allows refresh requests to be requested external to the 82C03.

All three of these inputs may be asynchronous with respect to the 82C03's clock. The arbiter will resolve conflicts between refresh and memory requests, for both pending cycles and cycles in progress. Read and write requests will be given priority over refresh requests.

System Operation

The 82C03 is always in one of the following states:

a) IDLE
b) TEST Cycle
c) REFRESH Cycle
d) READ Cycle
e) WRITE Cycle

The 82C03 is normally in the IDLE state. Whenever one of the other cycles is requested, the 82C03 will leave the IDLE state to perform the desired cycle. If no other cycles are pending, the 82C03 will return to the IDLE state.

Test Cycle

The TEST Cycle is used to check operation of several 82C03 internal functions. TEST cycles are requested by activating the $\overline{PCS}$, $\overline{RD}$ and $\overline{WR}$ inputs. The TEST Cycle will reset the refresh address counter and perform a WRITE Cycle. The TEST Cycle should not be used in normal system operation, since it would affect the dynamic RAM refresh.

Refresh Cycles

The 82C03 has two ways of providing dynamic RAM refresh:

1) Internal (failsafe) refresh
2) External (hidden) refresh

Both types of 82C03 refresh cycles activate all of the $\overline{RAS}$ ouputs, while $\overline{CAS}$, $\overline{WE}$, $\overline{SACK}$, and $\overline{XACK}$ remain inactive.

Internal refresh is generated by the on-chip refresh timer. The timer uses the 82C03 clock to ensure that refresh of all rows of the dynamic RAM occurs every 2 milliseconds (128 cycles) or every 4 milliseconds (256 cycles). If REFRQ is inactive, the refresh timer will request a refresh cycle every 10–16 microseconds.

External refresh is requested via the REFRQ input (pin 34). External refresh control is not available when the Advanced-Read mode is selected. External refresh requests are latched, then synchronized to the 82C03 clock.

The arbiter will allow the refresh request to start a refresh cycle only if the 82C03 is not in the middle of a cycle.

When the 82C03 is in the idle state a simultaneous memory request and external refresh request will result in the memory request being honored first. This 82C03 characteristic can be used to "hide" refresh cycles during system operation. A circuit similiar to Figure 7 can be used to decode the CPU's instruction fetch status to generate an external refresh request. The refresh request is latched while the 82C03 performs the instruction fetch; the refresh cycle will start immediately after the memory cycle is completed, even if the $\overline{RD}$ input has not gone inactive. If the CPU's instruction decode time is long enough, the 82C03 can complete the refresh cycle before the next memory request is generated.

If the 82C03 is not in the idle state then a simultaneous memory request and an external refresh request may result in the refresh request being honored first.

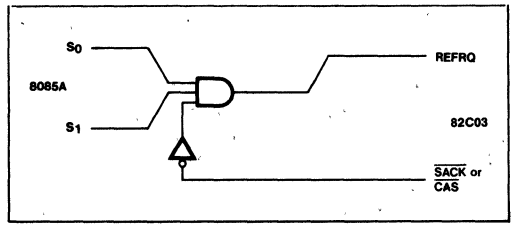


Figure 7. Hidden Refresh

Certain system configurations require complete external refresh requests. If external refresh is requested faster than the minimum internal refresh timer (t_{REF}), then, in effect, all refresh cycles will be caused by the external refresh request, and the internal refresh timer will never generate a refresh request.

Read Cycles

The 82C03 can accept two different types of memory Read requests:

1) Normal Read, via the $\overline{RD}$ input
2) Advanced Read, using the S1 and ALE inputs (16K mode only)

The user can select the desired Read request configuration via the B1/OP1 hardware strapping option on pin 25.

	Normal Read	Advanced Read
Pin 25	B1 input	OP_1 (+12V)
Pin 32	RD input	S1 input
Pin 34	REFRQ input	ALE input
# RAM banks	4 ($\overline{RAS}_{0-3}$)	2 ($\overline{RAS}_{2-3}$)
Ext. Refresh	Yes	No

Figure 8. 82C03 Read Options

Normal Reads are requested by activating the $\overline{RD}$ input, and keeping it active until the 82C03 responds with an $\overline{XACK}$ pulse. The $\overline{RD}$ input can go inactive as soon as the command hold time (t_{CHS}) is met.

Advanced Read cycles are requested by pulsing ALE while S1 is active; if S1 is inactive (low) ALE is ignored. Advanced Read timing is similiar to Normal Read timing, except the falling edge of ALE is used as the cycle start reference.

If a Read cycle is requested while a refresh cycle is in progress, then the 82C03 will set the internal delayed-SACK latch. When the Read cycle is eventually started, the 82C03 will delay the active $\overline{SACK}$ transition until $\overline{XACK}$ goes active, as shown in the AC timing diagrams. This delay was designed to compensate for the CPU's READY setup and hold times. The delayed-SACK latch is cleared after every READ cycle.

Based on system requirements, either $\overline{SACK}$ or $\overline{XACK}$ can be used to generate the CPU READY signal. $\overline{XACK}$ will normally be used; if the CPU can tolerate an advanced READY, then $\overline{SACK}$ can be used, but only if the CPU can tolerate the amount of advance provided by $\overline{SACK}$. If $\overline{SACK}$ arrives too early to provide the appropriate number of WAIT states, then either $\overline{XACK}$ or a delayed form of $\overline{SACK}$ should be used.

Write Cycles

Write cycles are similiar to Normal Read cycles, except for the $\overline{WE}$ output. $\overline{WE}$ is held inactive for Read cycles, but goes active for Write cycles. All 82C03 Write cycles are "early-write" cycles; $\overline{WE}$ goes active before $\overline{CAS}$ goes active by an amount of time sufficient to keep the dynamic RAM output buffers turned off.

General System Considerations

All memory requests (Normal Reads, Advanced Reads, Writes) are qualified by the $\overline{PCS}$ input. $\overline{PCS}$ should be stable, either active or inactive, prior to the leading edge of $\overline{RD}$, $\overline{WR}$, or ALE. Systems which use battery backup should pullup $\overline{PCS}$ to prevent erroneous memory requests.

In order to minimize propagation delay, the 82C03 uses an inverting address multiplexer without latches. The system must provide adequate address setup and hold times to guarantee $\overline{RAS}$ and $\overline{CAS}$ setup and hold times for the RAM. The t_{ADA} Cparacmeter should be used for this system calculation.

The B_0-B_1 inputs are similiar to the address inputs in that they are not latched. B_0 and B_1 should not be changed during a memory cycle, since they directly control which $\overline{RAS}$ output is activated.

The 82C03 uses a two-stage synchronizer for the memory request inputs ($\overline{RD}$, $\overline{WR}$, ALE), and a separate two stage synchronizer for the external refresh input (REFRQ). As with any synchronizer, there is always a finite probability of metastable states inducing system errors. The 82C03 synchronizer was designed to have a system error rate less than 1 memory cycle every three years based on the full operating range of the 82C03.

A microprocessor system is concerned when the data is valid after $\overline{RD}$ goes low. See Figure 9. In order to calculate memory read access times, the dynamic RAM's A.C. specifications must be examined, especially the RAS-access time (t_{RAC}) and the CAS-access time (t_{CAC}). Most configurations will be CAS-access limited; i.e., the data from the RAM will be stable t_{CC}, max (82C03) + t_{CC} (RAM) after a memory read cycle is started. Be sure to add any delays (due to buffers, data latches, etc.) to calculate the overall read access time.

Since the 82C03 normally performs "early-write" cycles, the data must be stable at RAM data inputs by the time $\overline{CAS}$ goes active, including the RAM's data setup time. If the system does not normally guarantee sufficient write data setup, you must either delay the WR input signal or delay the 82C03 $\overline{WE}$ output.

Delaying the $\overline{WR}$ input will delay all 82C03 timing, including the READY handshake signals, $\overline{SACK}$ and

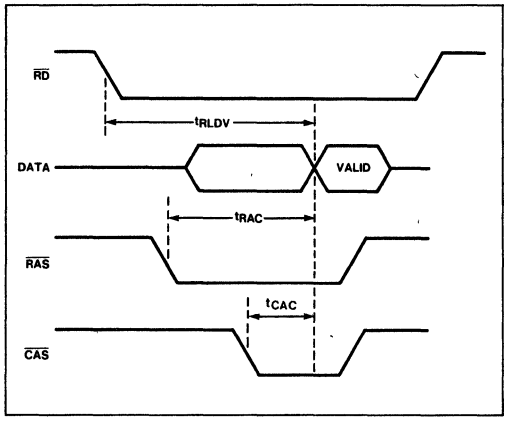


Figure 9. Read Access Time

XACK, which may increase the number of WAIT states generated by the CPU.

If the $\overline{WE}$ output is externally delayed beyond the $\overline{CAS}$ active transition, then the RAM will use the falling edge of $\overline{WE}$ to strobe the write data into the RAM. This $\overline{WE}$ transition should not occur too late during the CAS active transition, or else the $\overline{WE}$ to $\overline{CAS}$ requirements of the RAM will not be met.

The $\overline{RAS}_{0-3}$, $\overline{CAS}$, $\overline{OUT}_{0-7}$, and $\overline{WE}$ outputs contain on-chip series damping resistors (typically 20Ω) to minimize overshoot.

Some dynamic RAMs require more than 2.4V V_{IH}. Noise immunity may be improved for these RAMs by adding pull-up resistors to the 82C03's output. Intel RAMs do not require pull-up resistors.

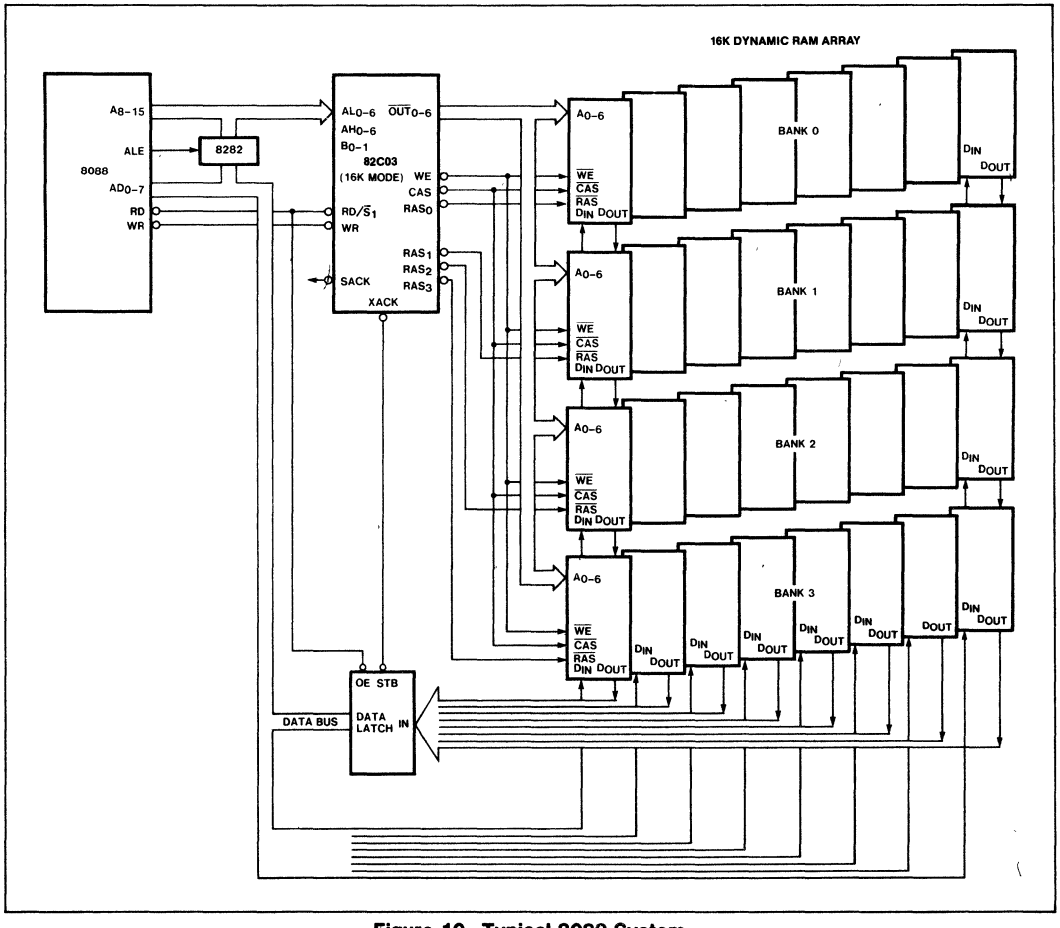


Figure 10. Typical 8088 System

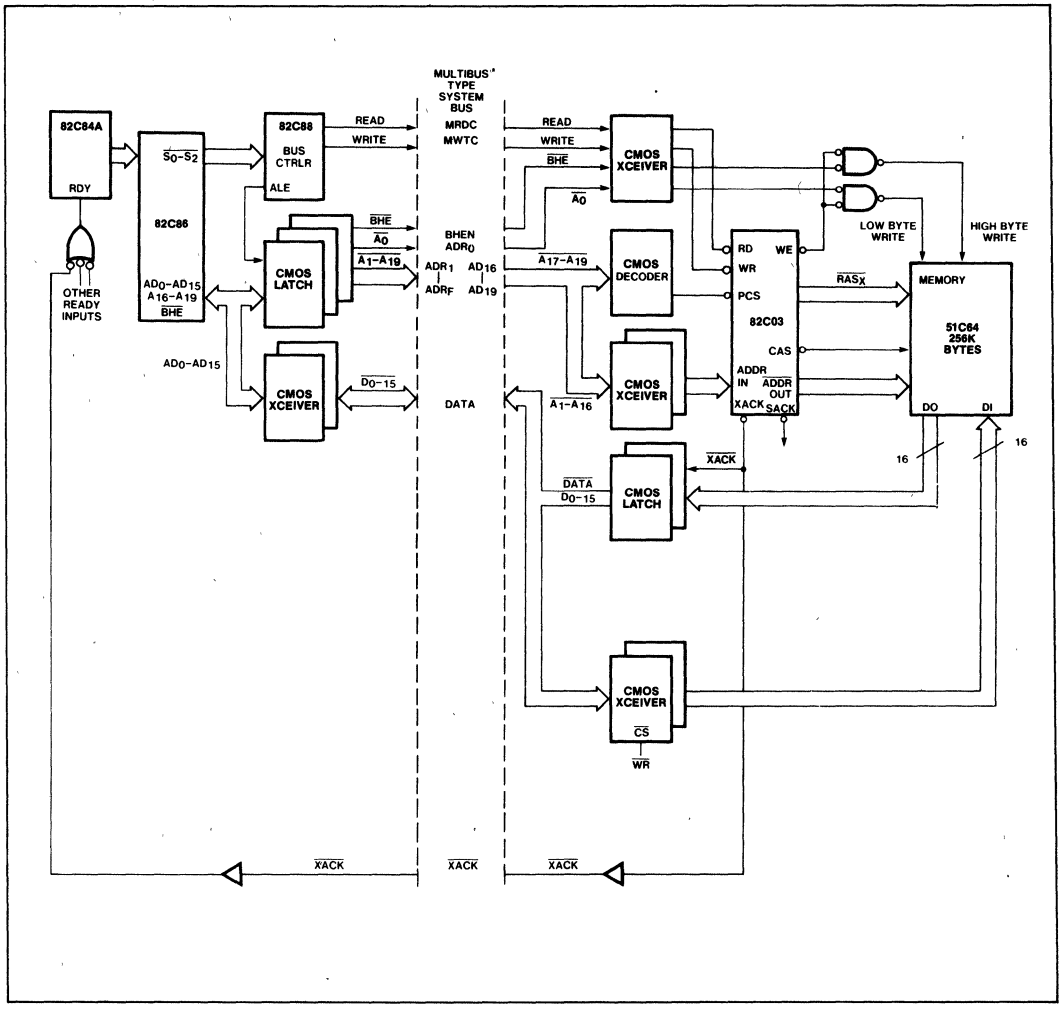


Figure 11. 80C86/256K Byte System

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
Voltage On any Pin With Respect to Ground	−0.5V to +7V4
Power Dissipation	0.2 Watts

**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

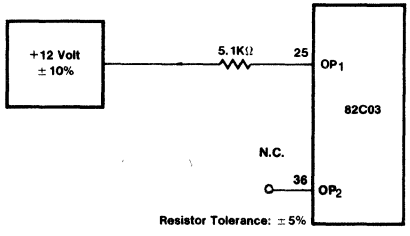
D.C. CHARACTERISTICS

T_A = 0°C to 70°C; V_{CC} = 5.0V ± 10%; GND = 0V

Symbol	Parameter	Min	Max	Units	Test Conditions
V_C	Input Clamp Voltage		−1.0	V	I_C = −5 mA
I_{CC}	Power Supply Current		25	mA	
I_{LI}	Input Leakage Current		±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
V_{OL}	Output Low Voltage $\overline{SACK}$, $\overline{XACK}$ All Other Outputs		 0.45 0.45	 V V	 I_{OL} = 5 mA I_{OL} = 3 mA
V_{OH}	Output High Voltage $\overline{SACK}$, $\overline{XACK}$ All Other Outputs	 2.4 2.6		 V V	 I_{OH} = −1 mA I_{OH} = −1 mA
V_{IL}	Input Low Voltage		0.8	V	
V_{IH1}	Input High Voltage	2.0	V_{CC}	V	
V_{IH2}	Option Voltage		V_{CC}	V	(Note 4)
C_{IN}	Input Capacitance		30	pF	F = 1 MHz V_{BIAS} = 2.5V, V_{CC} = 5V T_A = 25°C

NOTES:

1. For test mode $\overline{RD}$ & $\overline{WR}$ must be held at GND.
2. Except for pin 36 in XTAL mode.
3



A.C. CHARACTERISTICS

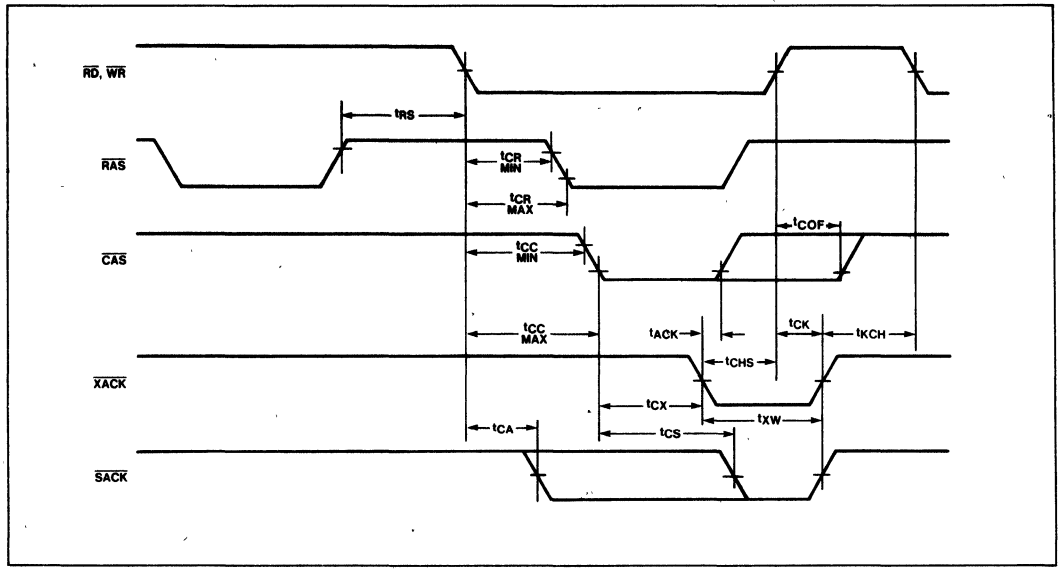
T_J = 0°C to 70°C; V_{CC} = 5V ± 10%; GND = 0V

Measurements made with respect to $\overline{RAS}_0$-$\overline{RAS}_3$, $\overline{CAS}$, $\overline{WE}$, $\overline{OUT}_0$-$\overline{OUT}_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V. All times are in nsec.

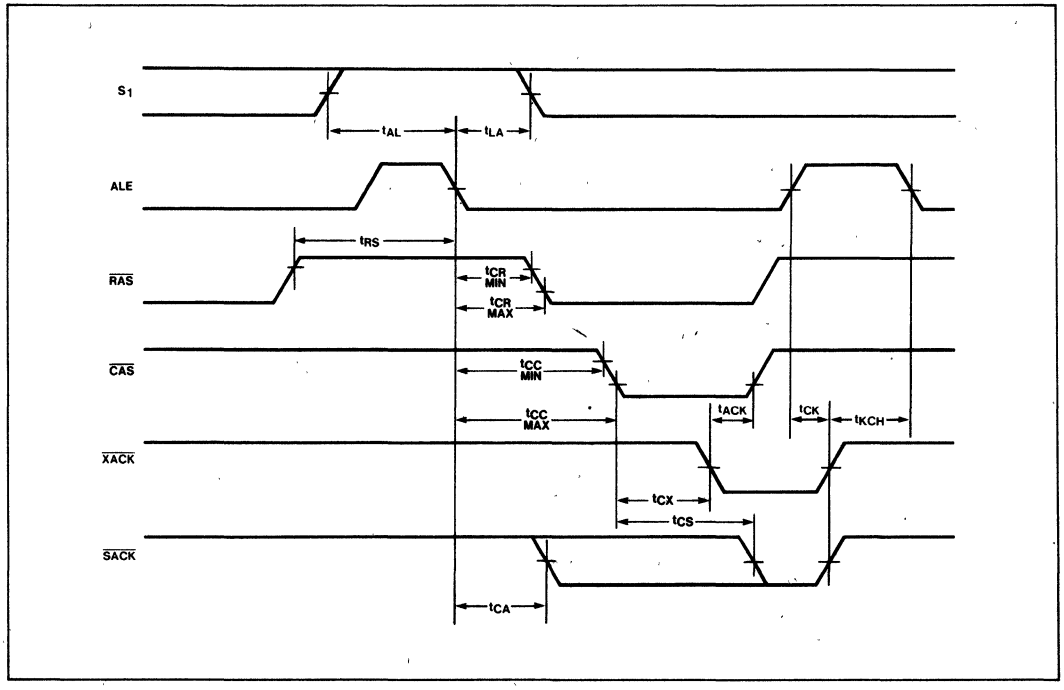
Symbol	Parameter	Min	Max	Notes
t_P	Clock Period	33	54	
t_{PH}	External Clock High Time	15		
t_{PL}	External Clock Low Time—above (>) 20 mHz	15		
t_{RC}	Memory Cycle Time	11tp - 20	11tp + 20	4, 5
t_{REF}	Refresh Time (128 cycles)	273tp	288tp	
t_{RP}	$\overline{RAS}$ Precharge Time	4tp + 3		
t_{RSH}	$\overline{RAS}$ Hold After $\overline{CAS}$	5tp − 30		3
t_{ASR}	Address Setup to $\overline{RAS}$	tp - 25		3
t_{RAH}	Address Hold From $\overline{RAS}$	tp - 8		3
t_{ASC}	Address Setup to $\overline{CAS}$	tp − 30		3
t_{CAH}	Address Hold from $\overline{CAS}$	5tp − 20		3
t_{CAS}	$\overline{CAS}$ Pulse Width	5tp − 10		
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	tp − 40		
t_{WCH}	$\overline{WE}$ Hold After $\overline{CAS}$	5tp − 35		8
t_{RS}	$\overline{RD}$, $\overline{WR}$, ALE, REFRQ delay from $\overline{RAS}$	3tp		2, 6
t_{MRP}	$\overline{RD}$, $\overline{WR}$ setup to $\overline{RAS}$	-1tp		5
t_{RMS}	REFRQ setup to $\overline{RD}$, $\overline{WR}$	2tp		6
t_{RMP}	REFRQ setup to $\overline{RAS}$	2tp		5
t_{PCS}	$\overline{PCS}$ Setup to $\overline{RD}$, $\overline{WR}$, ALE	20		
t_{AL}	S1 Setup to ALE	15		
t_{LA}	S1 Hold from ALE	30		
t_{CR}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{RAS}$ Delay	tp + 30	2tp + 70	2
t_{CC}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{CAS}$ Delay	3tp + 25	4tp + 80	2
t_{SC}	CMD Setup to Clock	15		1
t_{MRS}	$\overline{RD}$, $\overline{WR}$ setup to REFRQ	5		2
t_{CA}	$\overline{RD}$, $\overline{WR}$, ALE to $\overline{SACK}$ Delay	1tp	2tp + 47	2, 9
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Delay	5tp − 25	5tp + 20	
t_{CS}	$\overline{CAS}$ to $\overline{SACK}$ Delay	5tp − 25	5tp + 40	2, 10
t_{ACK}	$\overline{XACK}$ to $\overline{CAS}$ Setup	10		
t_{XW}	$\overline{XACK}$ Pulse Width	tp − 25		7
t_{CK}	$\overline{SACK}$, $\overline{XACK}$ turn-off Delay		35	
t_{KCH}	CMD Inactive Hold after $\overline{SACK}$, $\overline{XACK}$	10		
t_{LL}	REFRQ Pulse Width	20		
t_{CHS}	CMD Hold Time	30		11
t_{RFR}	REFRQ to $\overline{RAS}$ Delay		4tp + 100	6
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Delay	0	50	8
t_{AD}	CPU Address Delay	0	35	3
t_{COF}	$\overline{CAS}$ Turn-Off Delay		70	
t_{PCH}	$\overline{PCS}$ Hold From $\overline{RD}$, $\overline{WR}$, $\overline{ALE}$	30		
t_{BRH}	B_0, B_1 Hold From $\overline{RAS}$	0		
t_{BS}	B_0, B_1 Setup to $\overline{RD}$, $\overline{WR}$, $\overline{ALE}$	0		

WAVEFORMS

Normal Read or Write Cycle

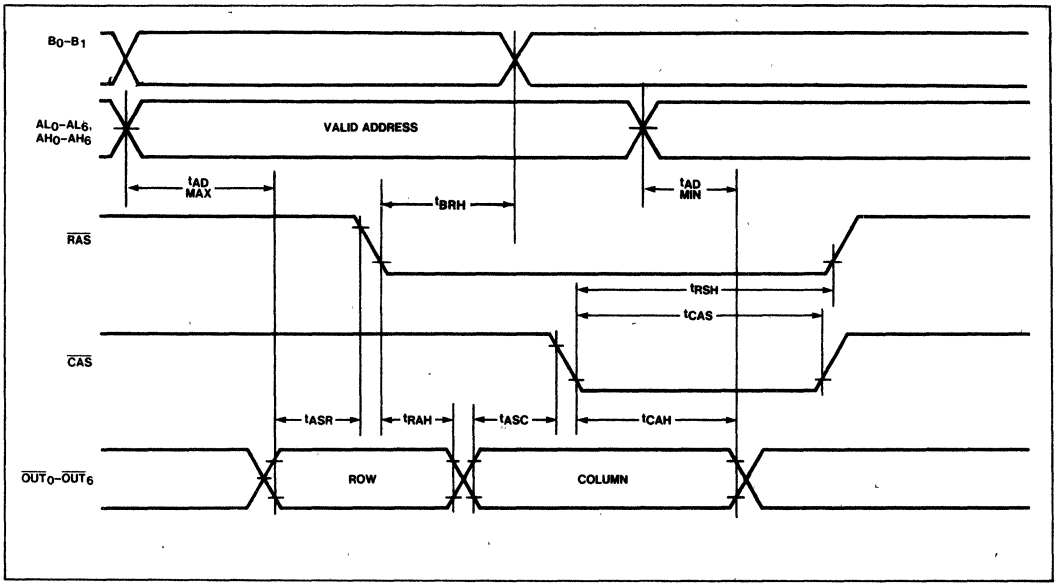


Advanced Read Mode

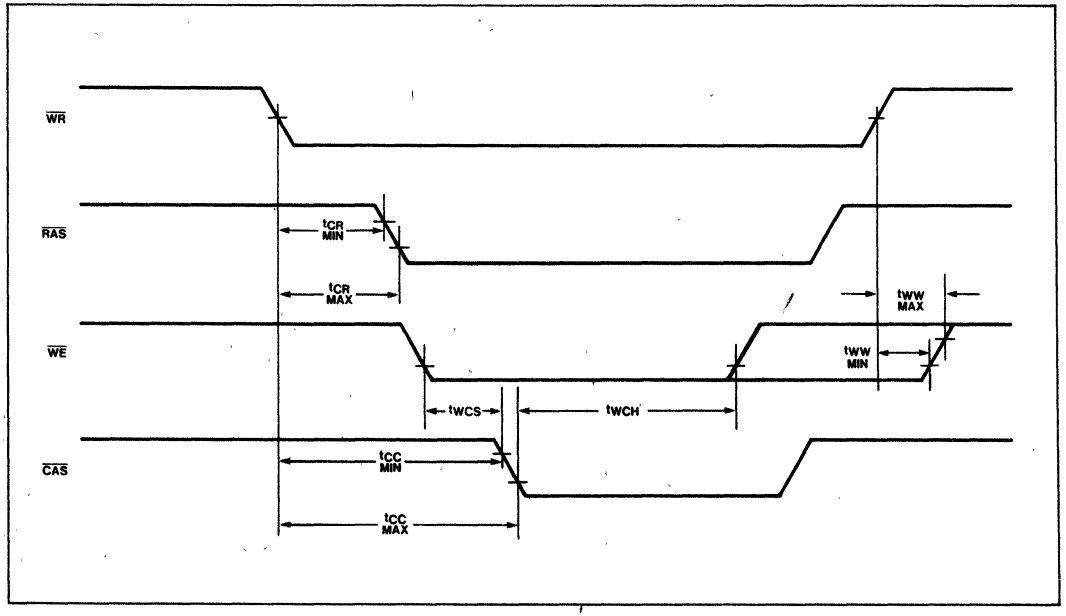


WAVEFORMS (cont'd)

Memory Compatibility Timing

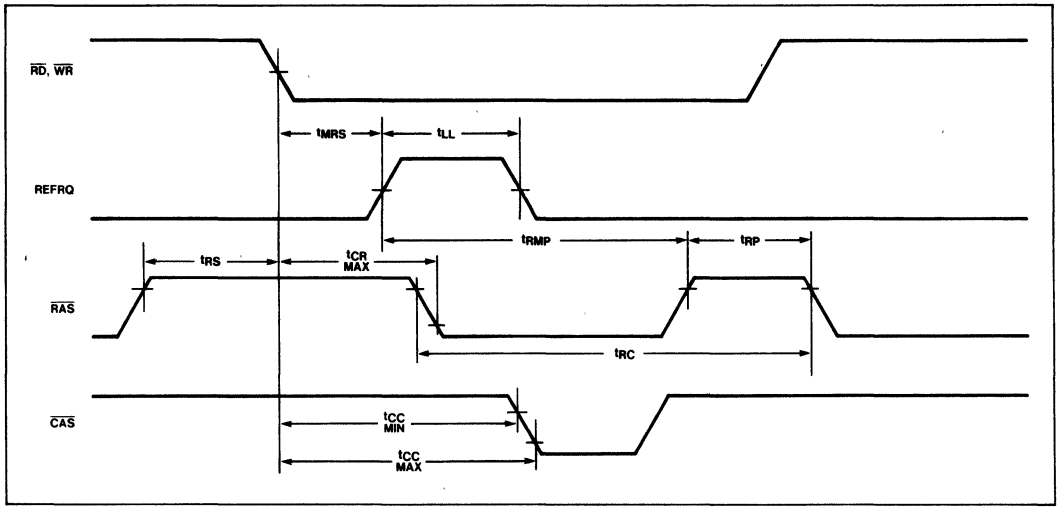


Write Cycle Timing

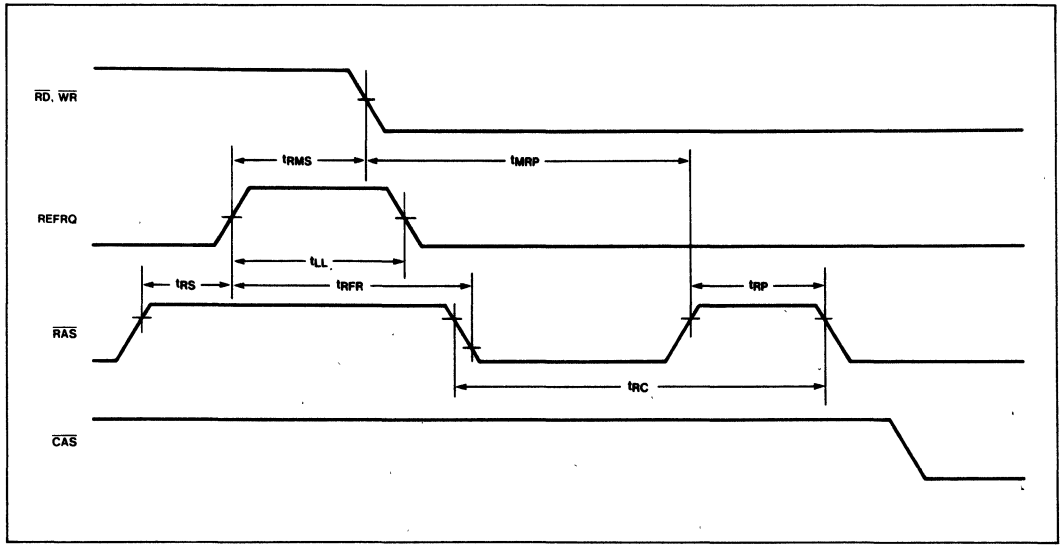


WAVEFORMS (cont'd)

Read or Write Followed By External Refresh



External Refresh Followed By Read or Write



WAVEFORMS (cont'd)

Clock And System Timing

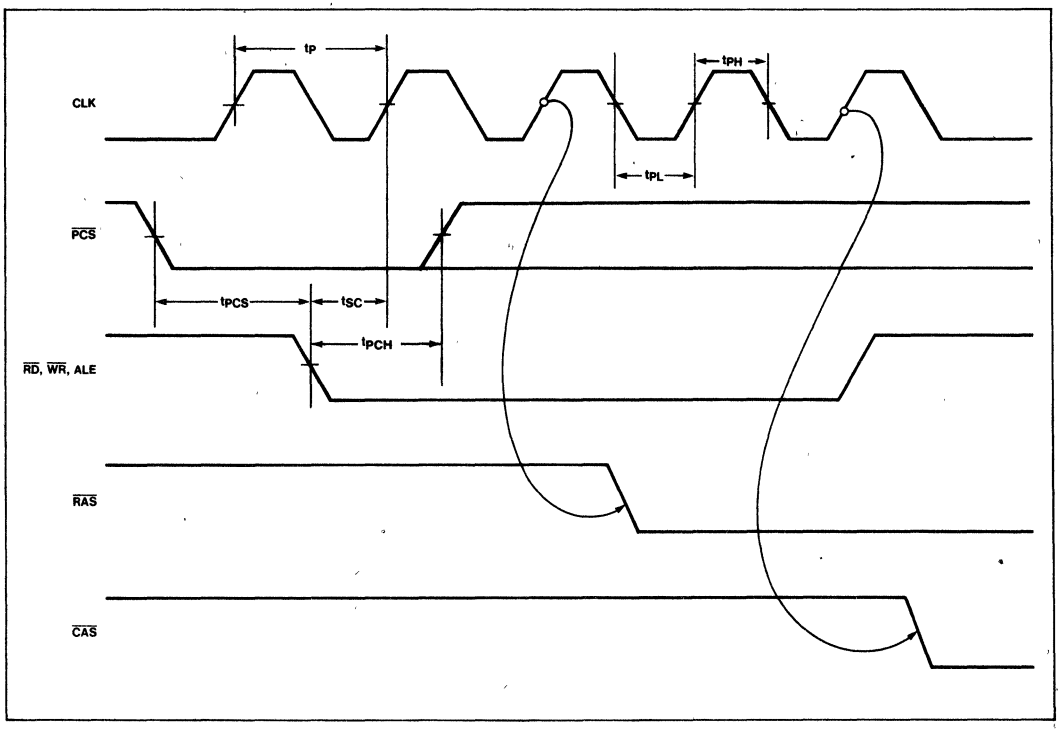


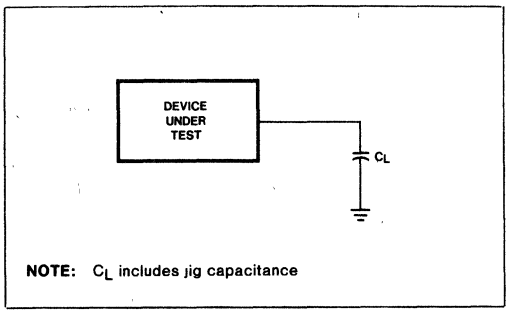
Table 2. 82C03 Ouptut Loading. All specifications are for the Test Load unless otherwise noted.

Pin	Test Load
$\overline{SACK}$, $\overline{XACK}$	C_L = 30 pF
$\overline{OUT}_0$-$\overline{OUT}_6$	C_L = 160 pF
$\overline{RAS}_0$-$\overline{RAS}_3$	C_L = 60 pF
$\overline{WE}$	C_L = 224 pF
$\overline{CAS}$	C_L = 320 pF

NOTES:

1. t_{SC} is a reference point only. ALE, $\overline{RD}$, $\overline{WR}$, and REFRQ inputs do not have to be externally synchronized to 82C03 clock
2. If t_{RS} min and t_{MRS} min are met then t_{CA}, t_{CR}, and t_{CC} are valid, T_{CS} is valid when delayed SACK is generated.
3. t_{ASR}, t_{RAH}, t_{ASC}, t_{CAH}, depend upon CPU address remaining stable throughout the memory cycle. The address inputs are not latched by the 82C03.
4. For back-to-back refresh cycles, t_{RC} max = 12tp
5. t_{RC} max is valid only if t_{RMP} min is met (READ, WRITE followed by REFRESH) or t_{MRP} min is met (REFRESH followed by READ, WRITE).
6. t_{RFR} is valid only if t_{RS} min and t_{RMS} min are met.
7. t_{XW} min applies when $\overline{RD}$, $\overline{WR}$ has already gone high. Otherwise $\overline{XACK}$ follows $\overline{RD}$, $\overline{WR}$.
8. $\overline{WE}$ goes high according to t_{WCH} or t_{WW}, whichever occurs first.

A.C. TESTING LOAD CIRCUIT



NOTE: C_L includes jig capacitance

8206/8206-2
ERROR DETECTION AND CORRECTION UNIT

- **Detects and Corrects All Single Bit Errors**
- **Detects All Double Bit and Most Multiple Bit Errors**
- **52 ns Maximum for Detection; 67 ns Maximum for Correction (16 Bit System)**
- **Syndrome Outputs for Error Logging**
- **8206-2 Timing Optimized for 8MHz iAPX 186, 188, 86, 88 and 8207-2 Systems**
- **Separate Input and Output Busses—No Timing Strobes Required**
- **Expandable to Handle 80 Bit Memories**
- **Supports Reads With and Without Correction, Writes, Partial (Byte) Writes, and Read-Modify-Writes**
- **HMOS Technology for Low Power**
- **68 Pin Leadless JEDEC Package**
- **Single +5V Supply**

The HMOS 8206 Error Detection and Correction Unit is a high-speed device that provides error detection and correction for memory systems (static and dynamic) requiring high reliability and performance. Each 8206 handles 8 or 16 data bits and up to 8 check bits. 8206's can be cascaded to provide correction and detection for up to 80 bits of data. Other 8206 features include the ability to handle byte writes, memory initialization, and error logging.

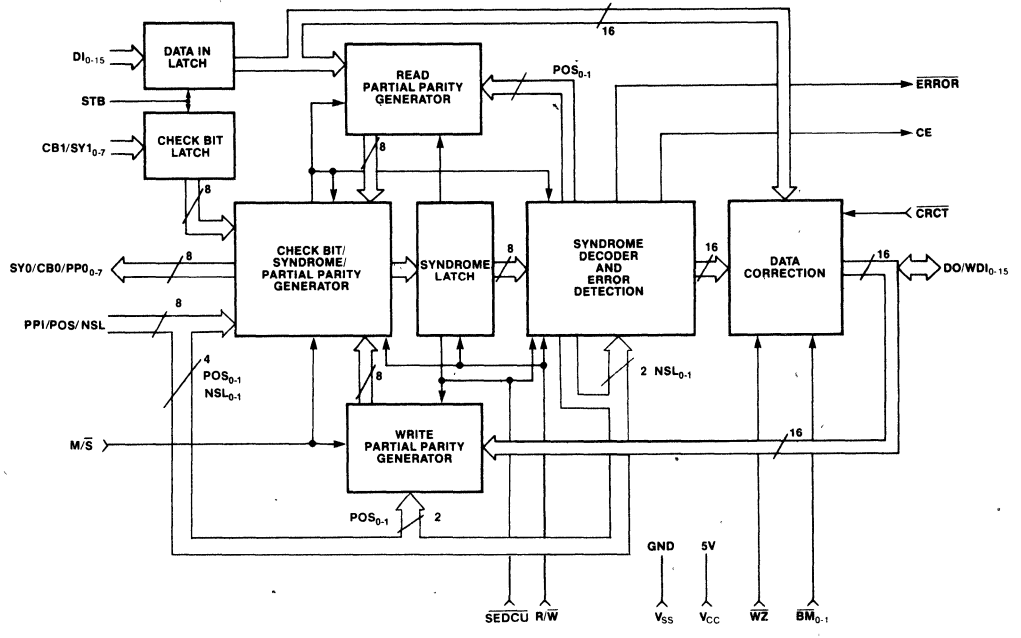


Figure 1. 8206 Block Diagram

MARCH 1982
Order Number 205220-004

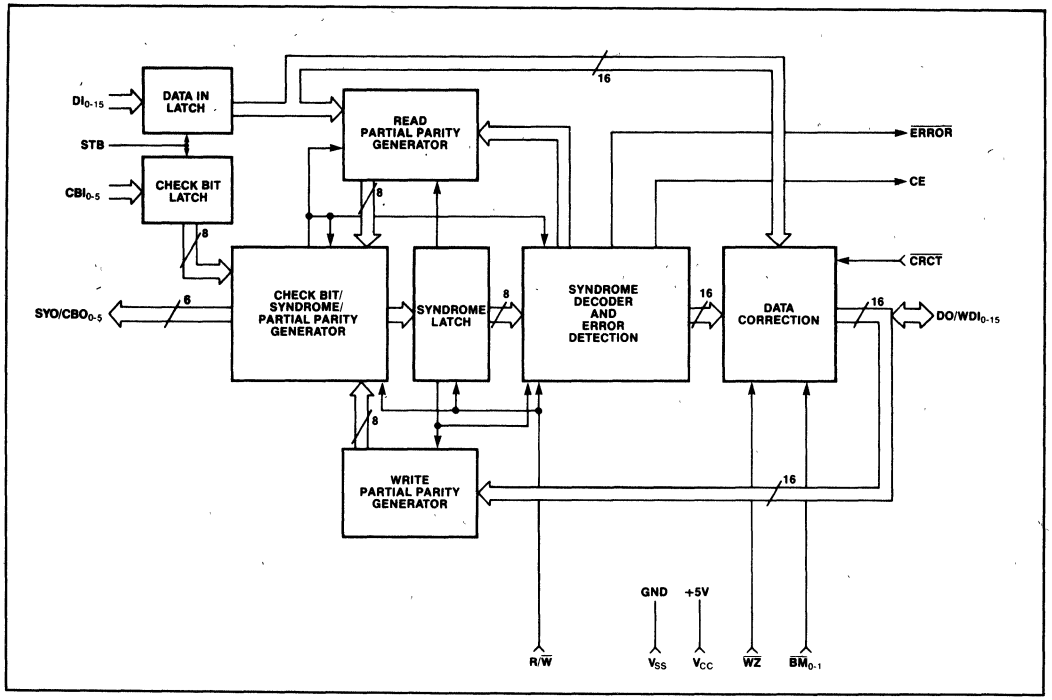


Figure 2. 8206-2 Block Diagram

Table 1. 8206 Pin Description

Symbol	Pin No.	Type	Name and Function
DI_{0-15}	1, 68-61, 59-53	I	**Data In:** These inputs accept a 16 bit data word from RAM for error detection and/or correction.
CBI/SYI_0 CBI/SYI_1 CBI/SYI_2 CBI/SYI_3 CBI/SYI_4 CBI/SYI_5 CBI/SYI_6 CBI/SYI_7	5 6 7 8 9 10 11 12	I I I I I I I I	**Check Bits In/Syndrome In:** In a single 8206 system, or in the master in a multi-8206 system, these inputs accept the check bits (5 to 8) from the RAM. In a single 8206 16 bit system, CBI_{0-5} are used. In slave 8206's these inputs accept the syndrome from the master.
DO/WDI_0 DO/WDI_1 DO/WDI_2 DO/WDI_3 DO/WDI_4 DO/WDI_5 DO/WDI_6 DO/WDI_7 DO/WDI_8 DO/WDI_9 DO/WDI_{10} DO/WDI_{11} DO/WDI_{12} DO/WDI_{13} DO/WDI_{14} DO/WDI_{15}	51 50 49 48 47 46 45 44 42 41 40 39 38 37 36 35	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	**Data Out/Write Data In:** In a read cycle, data accepted by DI_{0-15} appears at these outputs corrected if $\overline{CRCT}$ is low, or uncorrected if $\overline{CRCT}$ is high. The $\overline{BM}$ inputs must be high to enable the output buffers during the read cycle. In a write cycle, data to be written into the RAM is accepted by these inputs for computing the write check bits. In a partial-write cycle, the byte not to be modified appears at either DO_{0-7} if $\overline{BM}_0$ is high, or DO_{8-15} if $\overline{BM}_1$ is high, for writing to the RAM. When $\overline{WZ}$ is active, it causes the 8206 to output all zeros at DO_{0-15}, with the proper write check bits on CBO.

Table 1. 8206 Pin Description (Continued)

Symbol	Pin No.	Type	Name and Function
$SYO/CBO/PPO_0$ $SYO/CBO/PPO_1$ $SYO/CBO/PPO_2$ $SYO/CBO/PPO_3$ $SYO/CBO/PPO_4$ $SYO/CBO/PPO_5$ $SYO/CBO/PPO_6$ $SYO/CBO/PPO_7$	23 24 25 27 28 29 30 31	O O O O O O O O	**Syndrome Out/Check Bits Out/Partial Parity Out:** In a single 8206 system, or in the master in a multi-8206 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. In slave 8206's the partial parity bits used by the master appear at these outputs. The syndrome is latched (during read-modify-writes) by R/$\overline{W}$ going low.
PPI_0/POS_0 PPI_1/POS_1	13 14	I I	**Partial Parity In/Position:** In the master in a multi-8206 system, these inputs accept partial parity bits 0 and 1 from the slaves. In a slave 8206 these inputs inform it of its position within the system (1 to 4). Not used in a single 8206 system.
PPI_2/NSL_0 PPI_3/NSL_1	15 16	I I	**Partial Parity In/Number of Slaves:** In the master in a multi-8206 system, these inputs accept partial parity bits 2 and 3 from the slaves. In a multi-8206 system these inputs are used in slave number 1 to tell it the total number of slaves in the system (1 to 4). Not used in other slaves or in a single 8206 system.
PPI_4/CE	17	I/O	**Partial Parity In/Correctable Error:** In the master in a multi-8206 system this pin accepts partial parity bit 4. In slave number 1 only, or in a single 8206 system, this pin outputs the correctable error flag. CE is latched by R/$\overline{W}$ going low. Not used in other slaves.
PPI_5 PPI_6 PPI_7	18 19 20	I I I	**Partial Parity In:** In the master in a multi-8206 system these pins accept partial parity bits 5 to 7. The number of partial parity bits equals the number of check bits. Not used in single 8206 systems or in slaves.
$\overline{ERROR}$	22	O	**Error:** This pin outputs the error flag in a single 8206 system or in the master of a multi-8206 system. It is latched by R/$\overline{W}$ going low. Not used in slaves.
$\overline{CRCT}$	52	I	**Correct:** When low this pin causes data correction during a read or read-modify-write cycle. When high, it causes error correction to be disabled, although error checking is still enabled.
STB	2	I	**Strobe:** STB is an input control used to strobe data at the DI inputs and check-bits at the CBI/SYI inputs. The signal is active high to admit the inputs. The signals are latched by the high-to-low transition of STB.
$\overline{BM}_0$ $\overline{BM}_1$	33 32	I I	**Byte Marks:** When high, the Data Out pins are enabled for a read cycle. When low, the Data Out buffers are tristated for a write cycle. $\overline{BM}_0$ controls DO_{0-7}, while $\overline{BM}_1$ controls DO_{8-15}. In partial (byte) writes, the byte mark input is low for the new byte to be written.
R/$\overline{W}$	21	I	**Read/Write:** When high this pin causes the 8206 to perform detection and correction (if $\overline{CRCT}$ is low). When low, it causes the 8206 to generate check bits. On the high-to-low transition the syndrome is latched internally for read-modify-write cycles.
$\overline{WZ}$	34	I	**Write Zero:** When low this input overrides the $\overline{BM}_{0-1}$ and R/$\overline{W}$ inputs to cause the 8206 to output all zeros at DO_{0-15} with the corresponding check bits at CBO_{0-7}. Used for memory initialization.
M/$\overline{S}$	4	I	**Master/Slave:** Input tells the 8206 whether it is a master (high) or a slave (low).
$\overline{SEDCU}$	3	I	**Single EDC Unit:** Input tells the master whether it is operating as a single 8206 (low) or as the master in a multi-8206 system (high). Not used in slaves.
V_{CC}	60	I	**Power Supply:** +5V
V_{SS}	26	I	**Logic Ground**
V_{SS}	43	I	**Output Driver Ground**

Table 2. 8206-2 Pin Description Differences over the 8206.

Symbol	Pin	Type	Name and Function
CBI_{0-5}	5-10	I	**Check Bits In:** In an 8206-2 system, these inputs accept the check bits (5 to 6) from the RAM.
SYO/CBO_0 SYO/CBO_1 SYO/CBO_2 SYO/CBO_3 SYO/CBO_4 SYO/CBO_5	23 24 25 27 28 29	O O O O O O	**Syndrome Out/Check Bits Out:** In an 8206-2 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. The syndrome is latched (during read-modify-writes) by $R/\overline{W}$ going low.
CE	17	O	**Correctable Error:** In an 8206-2 system, this pin outputs the correctable error flag. CE is latched by $R/\overline{W}$ going low.
$\overline{WZ}$	34	I	**Write Zero:** When low this input overrides the BM_{0-1} and R/W inputs to cause the 8206-2 to output all zeros at DO_{0-15} with the corresponding check bits at CBO_{0-5}. Used for memory initialization.
Strap High	4	I	Must be tied High.
Strap Low	3	I	Must be tied Low.
N.C.	11-16 18-20	I	**Note:** These pins have internal pull-up resistors but if possible should be tied high or low.
N.C.	30, 31	O	**Note:** These are no connect pins and should be left open.

FUNCTIONAL DESCRIPTION

The 8206 Error Detection and Correction Unit provides greater memory system reliability through its ability to detect and correct memory errors. It is a single chip device that can detect and correct all single bit errors and detect all double bit and some higher multiple bit errors. Some other odd multiple bit errors (e.g., 5 bits in error) are interpreted as single bit errors, and the CE flag is raised. While some even multiple bit errors (e.g., 4 bits in error) are interpreted as no error, most are detected as double bit errors. This error handling is a function of the number of check bits used by the 8206 (see Figure 2) and the specific Hamming code used. Errors in check bits are not distinguished from errors in a word.

For more information on error correction codes, see Intel Application Notes AP-46 and AP-73.

DATA WORD BITS	CHECK BITS
8	5
16	6
24	6
32	7
40	7
48	8
56	8
64	8
72	8
80	8

Figure 3. Number of Check Bits Used by 8206

A single 8206 or 8206-2 handles 8 or 16 bits of data, and up to 5 8206's can be cascaded in order to handle data paths of 80 bits. For a single 8206 8 bit system, the DI_{8-15}, DO/WDI_{8-15} and BM_1 inputs are grounded. See the Multi-Chip systems section for information on 24-80 bit systems.

The 8206 has a "flow through" architecture. It supports two kinds of error correction architecture: 1) Flow-through, or correct-always; and 2) Parallel, or check-only. There are two separate 16-pin busses, one to accept data from the RAM (DI) and the other to deliver corrected data to the system bus ($\overline{DO}$/WDI). The logic is entirely combinatorial during a read cycle. This is in contrast to an architecture with only one bus, with bidirectional bus drivers that must first read the data and then be turned around to output the corrected data. The latter architecture typically requires additional hardware (latches and/or transceivers) and may be slower in a system due to timing skews of control signals.

READ CYCLE

With the R/$\overline{W}$ pin high, data is received from the RAM outputs into the DI pins where it is optionally latched by the STB signal. Check bits are generated from the data bits and compared to the check bits read from the RAM into the CBI pins. If an error is detected the $\overline{ERROR}$ flag is activated and the correctable error flag (CE) is used to inform the system whether the error was correctable or not. With the $\overline{BM}$ inputs high, the word appears corrected at the DO pins if the error was correctable, or unmodified if the error was uncorrectable.

If more than one 8206 is being used, then the check bits are read by the master. The slaves generate a partial parity output (PPO) and pass it to the master. The master 8206 then generates and returns the syndrome to the slaves (SYO) for correction of the data.

The 8206 may alternatively be used in a "check-only" mode with the $\overline{CRCT}$ pin left high. With the correction facility turned off, the propagation delay from memory outputs to 8206 outputs is significantly shortened. In this mode the 8206 issues an $\overline{ERROR}$ flag to the CPU, which can then perform one of several options: lengthen the current cycle for correction, restart the instruction, perform a diagnostic routine, etc.

A syndrome word, five to eight bits in length and containing all necessary information about the existence and location of an error, is made available to the system at the SYO_{0-7} pins. Error logging may be accomplished by latching the syndrome and the memory address of the word in error.

WRITE CYCLE

For a full write, in which an entire word is written to memory, the data is written directly to the RAM, bypassing the 8206. The same data enters the 8206 through the WDI pins where check bits are generated. The Byte Mark inputs must be low to tristate the DO drivers. The check bits, 5 to 8 in number, are then written to the RAM through the CBO pins for storage along with the data word. In a multi-chip system, the master writes the check bits using partial parity information from the slaves.

In a partial write, part of the data word is overwritten, and part is retained in memory. This is accomplished by performing a read-modify-write cycle. The complete old word is read into the 8206 and corrected, with the syndrome internally latched by R/$\overline{W}$ going low. Only that part of the word not to be modified is output onto the DO pins, as controlled by the Byte Mark inputs. That portion of the word to be overwritten is supplied by the system bus. The 8206 then calculates check bits for the new word, using the byte from the previous read and the new byte from the system bus, and writes them to the memory.

READ-MODIFY-WRITE CYCLES

Upon detection of an error the 8206 may be used to correct the bit in error in memory. This reduces the probability of getting multiple-bit errors in subsequent read cycles. This correction is handled by executing read-modify-write cycles.

The read-modify-write cycle is controlled by the R/$\overline{W}$ input. After (during) the read cycle, the system dynamic RAM controller or CPU examines the 8206 $\overline{ERROR}$ and CE outputs to determine if a correctable error occurred. If it did, the dynamic RAM controller or CPU forces R/$\overline{W}$ low, telling the 8206 to latch the generated syndrome and drive the corrected check bits onto the CBO outputs. The corrected data is available on the DO pins. The DRAM controller then writes the corrected data and corresponding check bits into memory.

The 8206 may be used to perform read-modify-writes in one or two RAM cycles. If it is done in two cycles, the 8206 latches are used to hold the data and check bits from the read cycle to be used in the following write cycle. The Intel 8207 Advanced Dynamic RAM controller allows read-modify-write cycles in one memory cycle. See the System Environment section.

INITIALIZATION

A memory system operating with ECC requires some form of initialization at system power-up in order to set valid data and check bit information in memory. The 8206 supports memory initialization by the write zero function. By activating the $\overline{WZ}$ pin, the 8206 will write a data pattern of zeros and the associated check bits in the current write cycle. By thus writing to all memory at power-up, a controller can set memory to valid data and check bits. Massive memory failure, as signified by both data and check bits all ones or zeros, will be detected as an uncorrectable error.

MULTI-CHIP SYSTEMS

A single 8206 handles 8 or 16 bits of data and 5 or 6 check bits, respectively. Up to 5 8206's can be cascaded for 80 bit memories with 8 check bits.

When cascaded, one 8206 operates as a master, and all others as slaves. As an example, during a read cycle in a 32 bit system with one master and one slave, the slave calculates parity on its portion of the word—"partial parity"—and presents it to the master through the PPO pins. The master combines the partial parity from the slave with the parity it calculated from its own portion of the word to generate the syndrome. The syndrome is then returned by the master to the slave for error correction. In systems with more than one slave the above description continues to apply, except that the partial parity outputs of the slaves must be XOR'd externally. Figure 4 shows the necessary external logic for multi-chip systems. Write and read-modify-write cycles are carried out analogously. See the System Operation section for multi-chip wiring diagrams.

There are several pins used to define whether the 8206 will operate as a master or a slave. Tables 3 and 4 illustrate how these pins are tied.

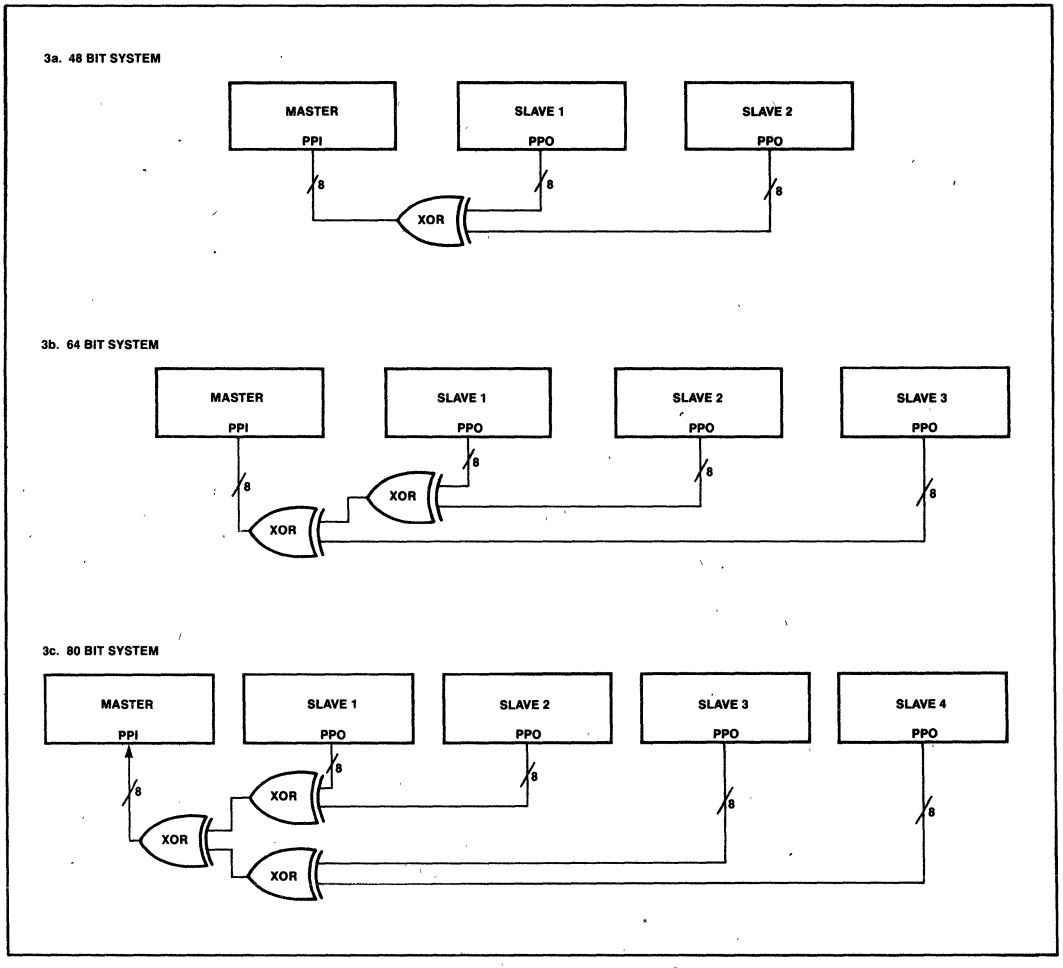


Figure 4. External Logic For Mult-Chip Systems

Table 3. Master/Slave Pin Assignments

Pin No.	Pin Name	Master	Slave 1	Slave 2	Slave 3	Slave 4
4	$M/\overline{S}$	+5V	Gnd	Gnd	Gnd	Gnd
3	$\overline{SEDCU}$	+5V	+5V	+5V	+5V	+5V
13	PPI_0/POS_0	PPI	Gnd	+5V	Gnd	+5V
14	PPI_1/POS_1	PPI	Gnd	Gnd	+5V	+5V
15	PPI_2/NSL_0	PPI	*	+5V	+5V	+5V
16	PPI_3/NSL_1	PPI	*	+5V	+5V	+5V

*See Table 3.

NOTE:
Pins 13, 14, 15, 16 have internal pull-up resistors and may be left as N.C. where specified as connecting to +5V.

Table 4. NSL Pin Assignments for Slave 1

	Number of Slaves			
Pin	1	2	3	4
PPI_2/NSL_0	Gnd	+5V	Gnd	+5V
PPI_3/NSL_1	Gnd	Gnd	+5V	+5V

The timing specifications for multi-chip systems must be calculated to take account of the external XOR gating in 3, 4, and 5-chip systems. Let tXOR be the delay for a single external TTL XOR gate. Then the following equations show how to calculate the relevant timing parameters for 2-chip (n=0), 3-chip (n=1), 4-chip (n=2), and 5-chip (n=2) systems:

Data-in to corrected data-out (read cycle) =
TDVSV + TPVSV + TSVQV + ntXOR

Data-in to error flag (read cycle) =
TDVSV + TPVEV + ntXOR

Data-in to correctable error flag (read cycle) =
TDVSV + TPVSV + TSVCV + ntXOR

Write data to check-bits valid (full write cycle) =
TQVQV + TPVSV + ntXOR

Data-in to check-bits valid (read-mod-write cycle) =
TDVSV + TPVSV + TSVQV + TQVQV + TPVSV + 2ntXOR

Data-in to check-bits valid (non-correcting read-modify-write cycle) =
TDVQU + TQVQV + TPVSV + ntXOR

HAMMING CODE

The 8206 uses a modified Hamming code which was optimized for multi-chip EDCU systems. The code is such that partial parity is computed by all 8206's in parallel. No 8206 requires more time for propagation through logic levels than any other one, and hence no one device becomes a bottleneck in the parity operation. However, one or two levels of external TTL XOR gates are required in systems with three to five chips. The code appears in Table 5. The check bits are derived from the table by XORing or XNORing together the bits indicated by 'X's in each row corresponding to a check bit. For example, check bit 0 in the MASTER for data word 1000110101101011 will be "0." It should be noted that the 8206 will detect the gross-error condition of all lows or all highs.

Error correction is accomplished by identifying the bad bit and inverting it. Table 5 can also be used as an error syndrome table by replacing the 'X's with '1's. Each column then represents a different syndrome word, and by locating the column corresponding to a particular syndrome the bit to be corrected may be identified. If the syndrome cannot be located then the error cannot be corrected. For example, if the syndrome word is 00110111, the bit to be corrected is bit 5 in the slave one data word (bit 21).

The syndrome decoding is also summarized in Tables 6 and 7 which can be used for error logging. By finding the appropriate syndrome word (starting with bit zero, the least significant bit), the result is either: 1) no error; 2) an identified (correctable) single bit error; 3) a double bit error; or 4) a multi-bit uncorrectable error.

Table 5. Modified Hamming Code Check Bit Generation

Check bits are generated by XOR'ing (except for the CB0 and CB1 data bits, which are XNOR'ed in the Master) the data bits in the rows corresponding to the check bits. Note there are 6 check bits in a 16-bit system, 7 in a 32-bit system, and 8 in 48-or-more-bit systems.

BYTE NUMBER		0								1								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	XNOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	XNOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
	CB5 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
	CB6 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
	CB7 =	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
DATA BITS		0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	
		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	

16 BIT OR MASTER

2								3								OPERATION
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
-	x	x	x	-	x	x	-	-	x	x	-	-	x	-	-	XOR
x	x	x	-	-	x	-	x	x	x	-	-	-	-	-	x	XOR
-	x	x	x	-	x	x	x	-	-	x	x	-	-	-	-	XOR
x	x	-	-	x	-	x	x	x	-	-	-	x	x	-	-	XOR
x	x	-	-	x	x	x	x	-	-	-	-	x	-	x	-	XOR
-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	XOR
1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	
6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	

SLAVE #1

BYTE NUMBER		4								5								6								7								8								9								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	x	-	x	-	x	x	-	-	x	-	x	x	-	-	x	-	-	x	-	x	-	x	x	-	-	x	x	-	-	x	-	-	XOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	-	x	x	-	-	-	x	x	-	x	x	-	x	-	-	x	-	x	x	-	x	x	x	-	-	x	x	-	-	-	-	-	XOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	-	x	x	-	-	x	-	-	-	x	x	-	-	x	x	x	-	x	-	-	x	-	x	-	-	x	x	-	-	x	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	x	-	x	-	-	x	x	-	x	x	-	-	x	x	-	-	x	x	x	x	-	-	x	x	x	x	-	-	-	x	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	-	x	x	-	-	-	x	x	x	x	-	-	-	x	-	-	XOR
	CB5 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	x	-	x	x	x	x	x	-	-	-	-	x	-	-	x	-	XOR
	CB6 =	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	-	-	-	x	x	-	-	-	-	x	-	x	-	-	XOR
	CB7 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
DATA BITS		3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7	7	7	
		2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	

SLAVE #2 | SLAVE #3 | SLAVE #4

Table 6. 8206 Syndrome Decoding

Syndrome Bits 7	6	5	4	0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 2 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 3 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0	0	0	0	N CB0 CB1 D CB2 D D 18 CB3 D D 0 D 1 2 D
0	0	0	1	CB4 D D 5 D 6 7 D D 3 16 D 4 D D 17
0	0	1	0	CB5 D D 11 D 19 12 D D 8 9 D 10 D D 67
0	0	1	1	D 13 14 D 15 D D 21 20 D D 66 D 22 23 D
0	1	0	0	CB6 D D 25 D 26 49 D D 48 24 D 27 D D 50
0	1	0	1	D 52 55 D 51 D D 70 28 D D 65 D 53 54 D
0	1	1	0	D 29 31 D 64 D D 69 68 D D 32 D 33 34 D
0	1	1	1	30 D D 37 D 38 39 D D 35 71 D 36 D D U
1	0	0	0	CB7 D D 43 D 77 44 D D 40 41 D 42 D D U
1	0	0	1	D 45 46 D 47 D D 74 72 D D U D 73 U D
1	0	1	0	D 59 75 D 79 D D 58 60 D D 56 D U 57 D
1	0	1	1	63 D D 62 D U U D D U U D 61 D D U
1	1	0	0	D U U D U D D U 76 D D U D U U D
1	1	0	1	78 D D U D U U D D U U D U D D U
1	1	1	0	U D D U D U U D D U U D U D D U
1	1	1	1	D U U D U D D U U D D U D U U D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error
U = Uncorrectable Multi-Bit Error

SYSTEM ENVIRONMENT

The 8206 interface to a typical 32 bit memory system is illustrated in Figure 5. For larger systems, the partial parity bits from slaves two to four must be

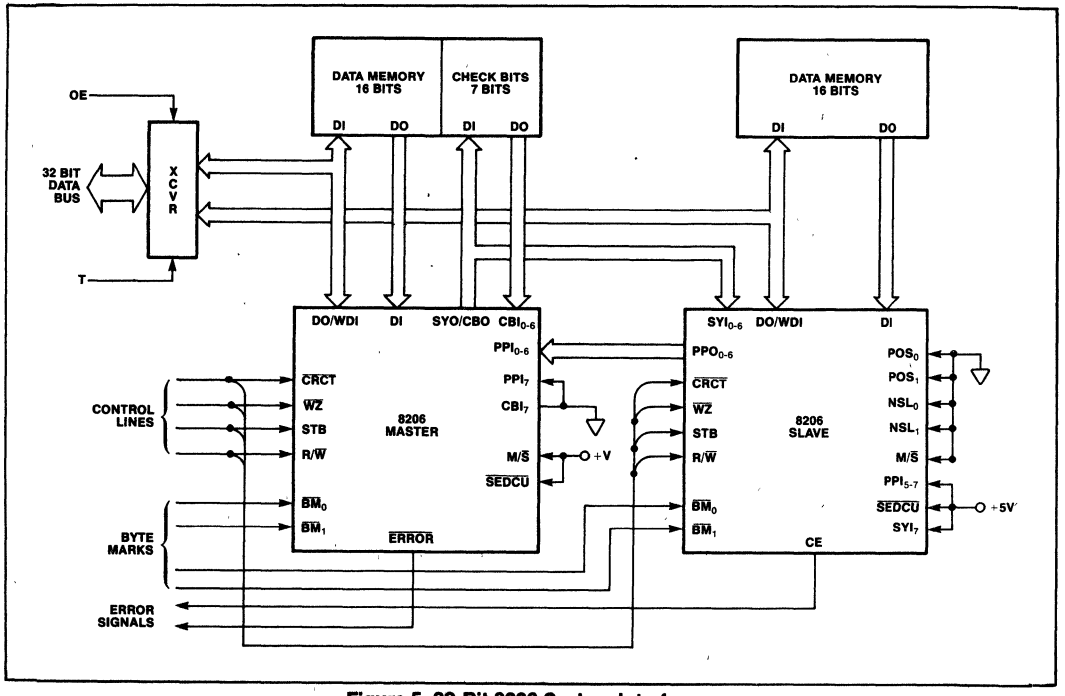


Figure 5. 32-Bit 8206 System Interface

XOR'ed externally, which calls for one level of XOR gating for three 8206's and two levels for four or five 8206's.

The 8206 is designed for direct connection to the Intel 8207 Advanced Dynamic RAM Controller. The 8207 has the ability to perform dual port memory control, and Figure 6 illustrates a highly integrated dual port RAM implementation using the 8206 and 8207. The 8206/8207 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystem.

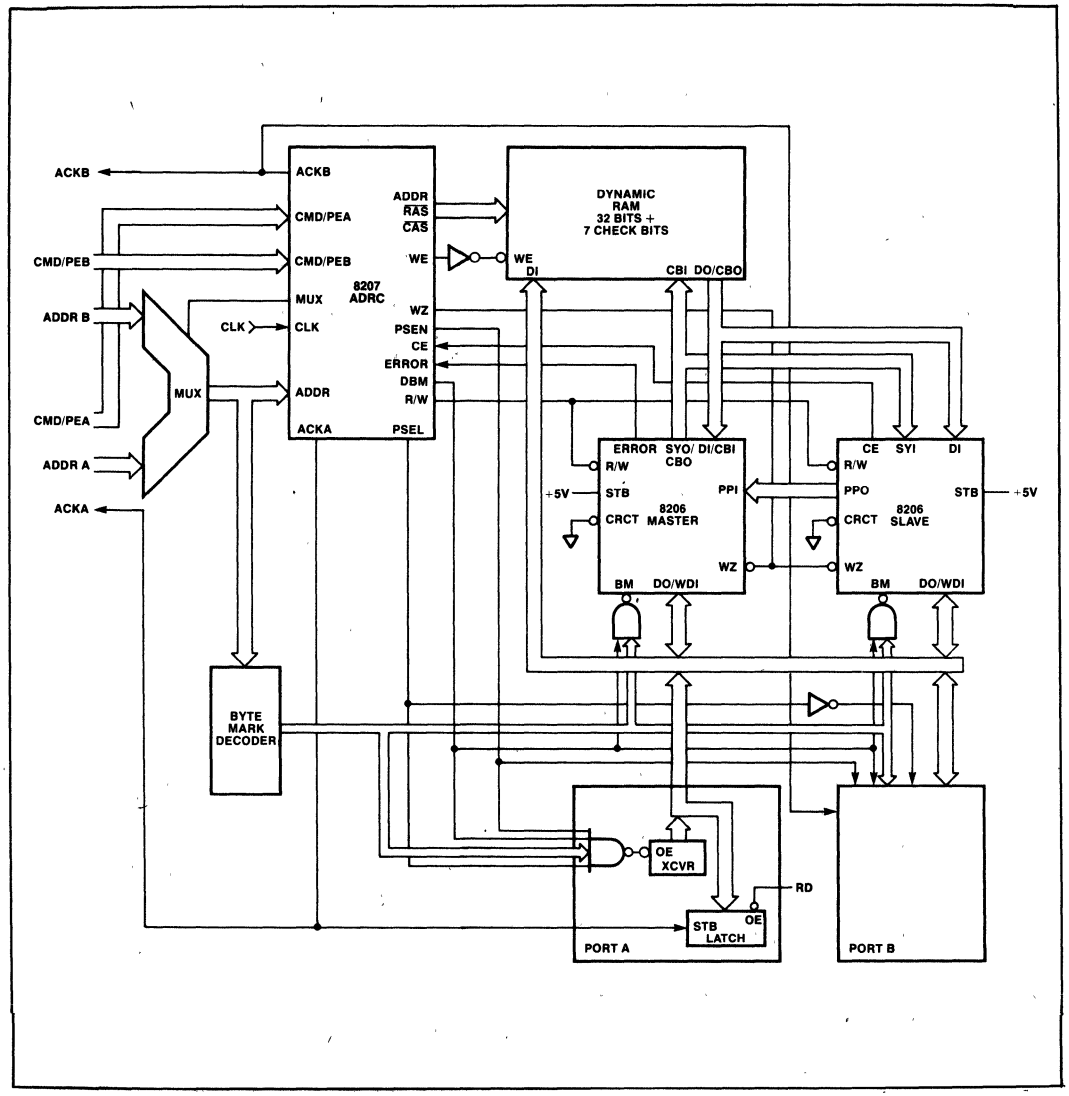


Figure 6. Dual Port RAM Subsystem with 8206/8207 (32-bit bus)

Table 7. 8206-2 Syndrome Decoding

Syndrome Bits 5	4	3	0 0 1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 1 1 1 2 0 0 0 0 1 1 1 1
0	0	0	N CB0 CB1 D CB2 D D D
0	0	1	CB3 D D 0 D 1 2 D
0	1	0	CB4 D D 5 D 6 7 D
0	1	1	D 3 D D 4 D D D
1	0	0	CB5 D D 11 D D 12 D
1	0	1	D 8 9 D 10 D D D
1	1	0	D 13 14 D 15 D D D
1	1	1	D D D D D D D D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error

The 8206-2 handles 8 or 16 bits of data. For 8 bit 8206-2 systems, the DI_{8-15}, DO/WDI_{8-15} and $\overline{BM}_1$ inputs are grounded.

The 8206-2 is designed for direct connection to the Intel 8207-2 Advanced Dynamic RAM Controller. The 8207-2 has the ability to perform dual port memory control, and Figure 7 illustrates a highly integrated iAPX 186 RAM implementation using the 8206-2 and 8207-2. The 8206-2/8207-2 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystems.

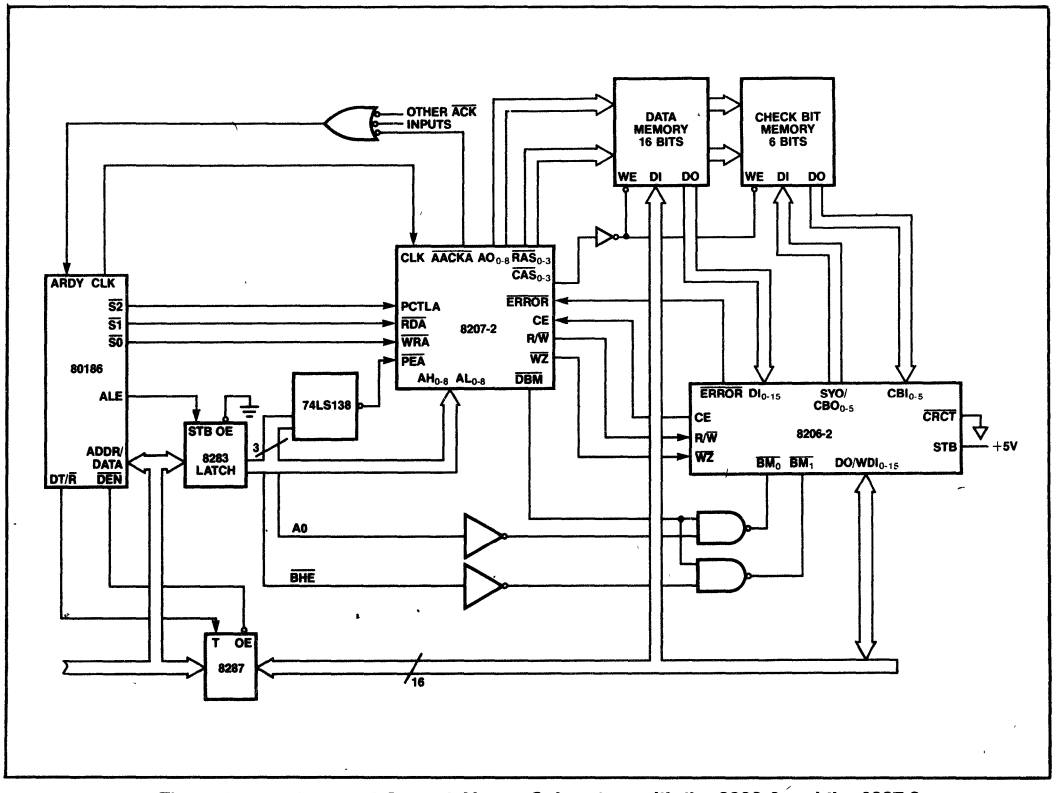


Figure 7. iAPX 186 RAM Correct Always Subsystem with the 8206-2 and the 8207-2

MEMORY BOARD TESTING

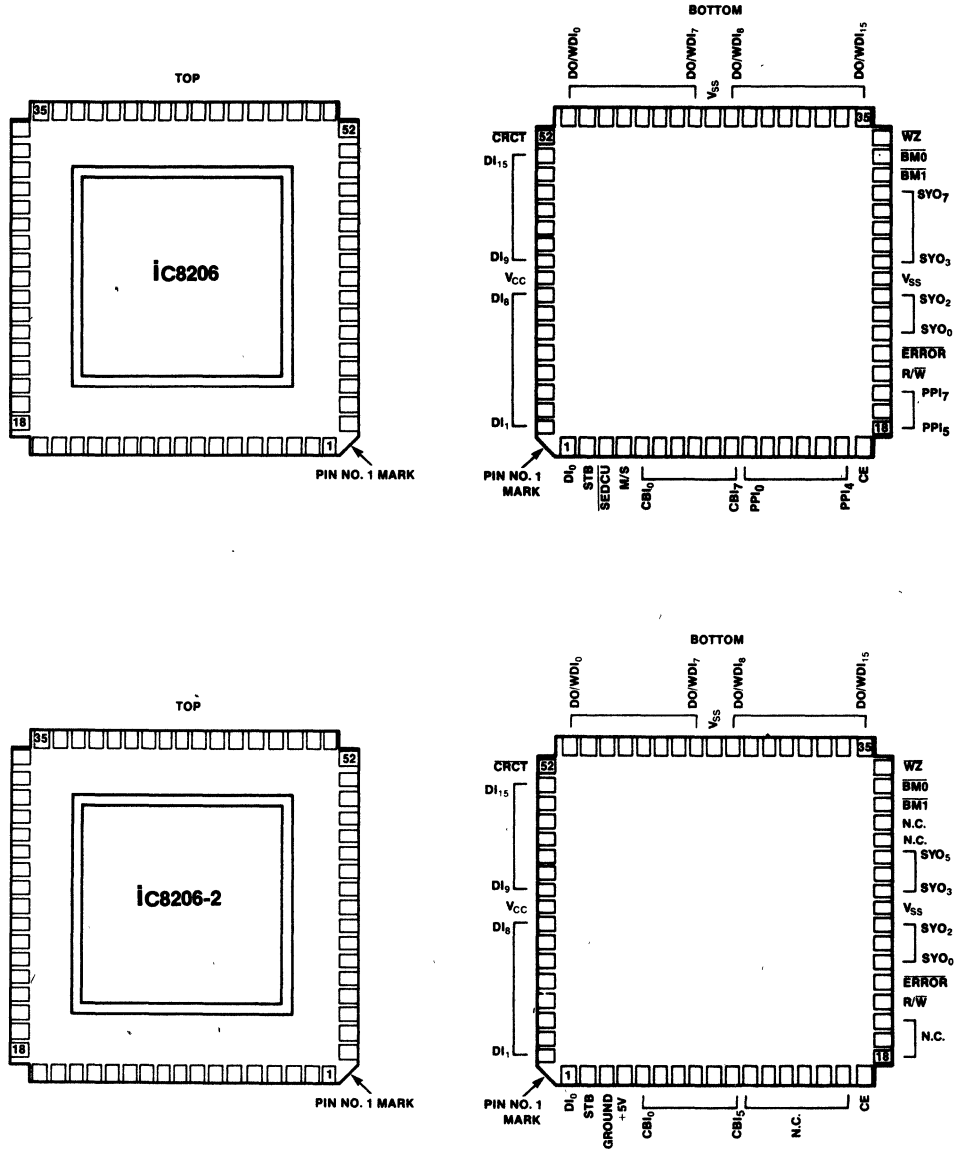
The 8206 lends itself to straightforward memory board testing with a minimum of hardware overhead. The following is a description of four common test modes and their implementation.

Mode 0—Read and write with error correction.
Implementation: This mode is the normal 8206 operating mode.

Mode 1—Read and write data with error correction disabled to allow test of data memory.
Implementation: This mode is performed with $\overline{CRCT}$ deactivated.

Mode 2—Read and write check bits with error correction disabled to allow test of check bits memory.
Implementation: Any pattern may be written into the check bits memory by judiciously choosing the proper data word to generate the desired check bits, through the use of the 8206 Hamming code. To read out the check bits it is first necessary to fill the data memory with all zeros, which may be done by activating $\overline{WZ}$ and incrementing memory addresses with $\overline{WE}$ to the check bits memory held inactive, and then performing ordinary reads. The check bits will then appear directly at the SYO outputs, with bits CB0 and CB1 inverted.

Mode 3—Write data, without altering or writing check bits, to allow the storage of bit combinations to cause error correction and detection.
Implementation: This mode is implemented by writing the desired word to memory with $\overline{WE}$ to the check bits array held inactive.



NOTE:
The 8206 and 8206-2 is packaged in a 68 pin JEDEC TYPE A hermetic chip carrier

Figure 8. 8206 and 8206-2 Pinout Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 1.5 Watts

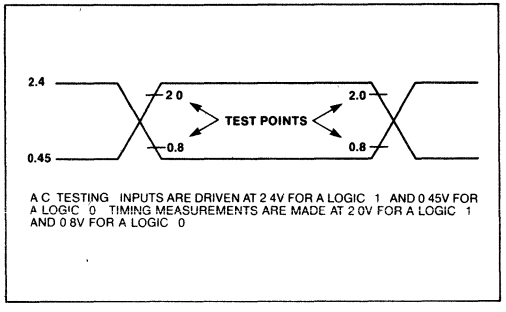
**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, V_{SS} = GND)

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
I_{CC}	Power Supply Current —Single 8206, 8206-2 or Slave #1 —Master in Multi-Chip or Slaves #2, 3, 4		270 230	mA mA	
V_{IL}[1]	Input Low Voltage	−0.5	0.8	V	
V_{IH}[1]	Input High Voltage	2.0	V_{CC} + 0.5V	V	
V_{OL}	Output Low Voltage —DO —All Others		0.45 0.45	V V	I_{OL} = 8mA I_{OL} = 2.0mA
V_{OH}	Output High Voltage —DO, CBO —All Other Outputs	2.6 2.4		V V	I_{OH} = −2mA I_{OH} = −0.4mA
I_{LO}	I/O Leakage Current —PPI_4/CE —DO/WDI_{0-15}		± 20 ± 10	μA μA	$0.45V \leq V_{I/O} \leq V_{CC}$
I_{LI}	Input Leakage Current —$PPI_{0-3, 5-7}$, CBI_{6-7}, $\overline{SEDCU}$[2] —All Other Input Only Pins		± 20 ± 10	μA μA	$0V \leq V_{IN} \leq V_{CC}$

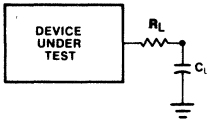
NOTES:
1. $\overline{SEDCU}$ (pin 3) and M/$\overline{S}$ (pin 4) are device strapping options and should be tied to V_{CC} or GND. V_{IH} min = V_{CC} −0.5V and V_{IL} max = 0.5V.
2. PPI_{0-7} (pins 13–20) and CBI_{6-7} (pins 11, 12) have internal pull-up resistors and if left unconnected will be pulled to V_{CC}.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

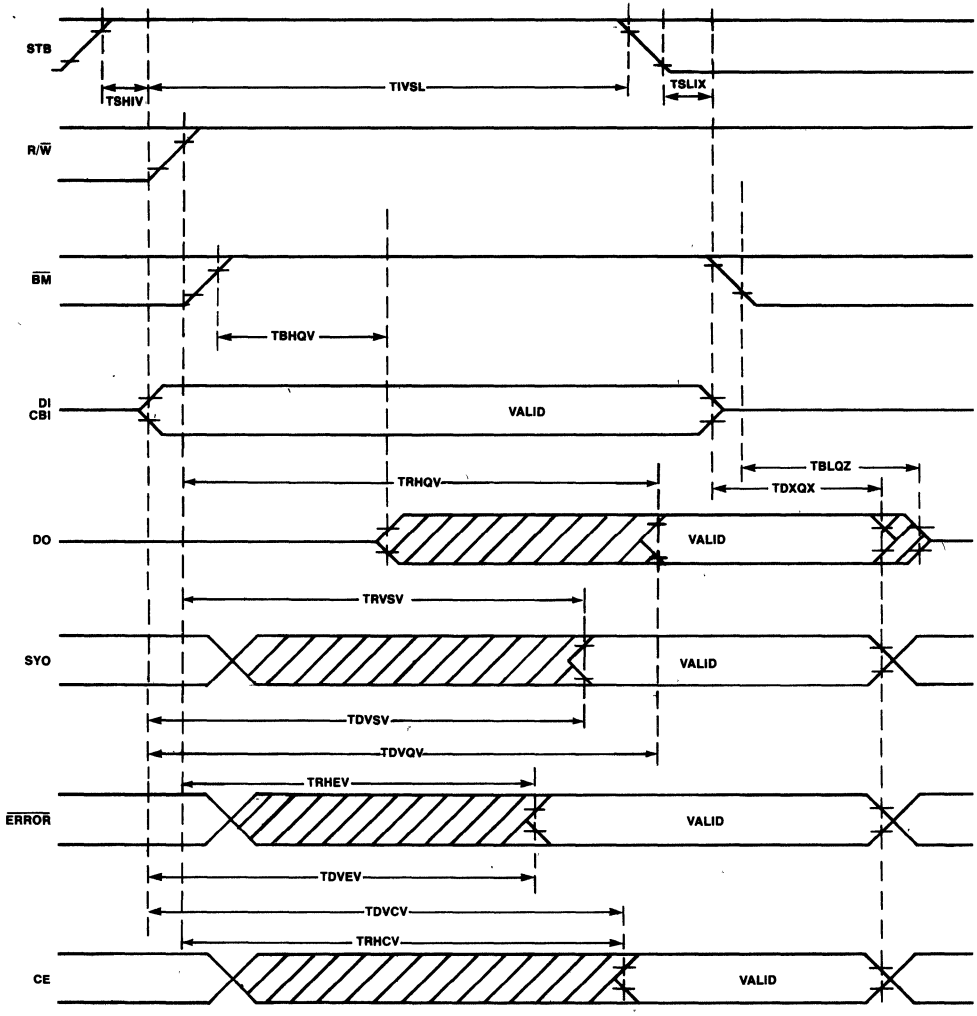
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

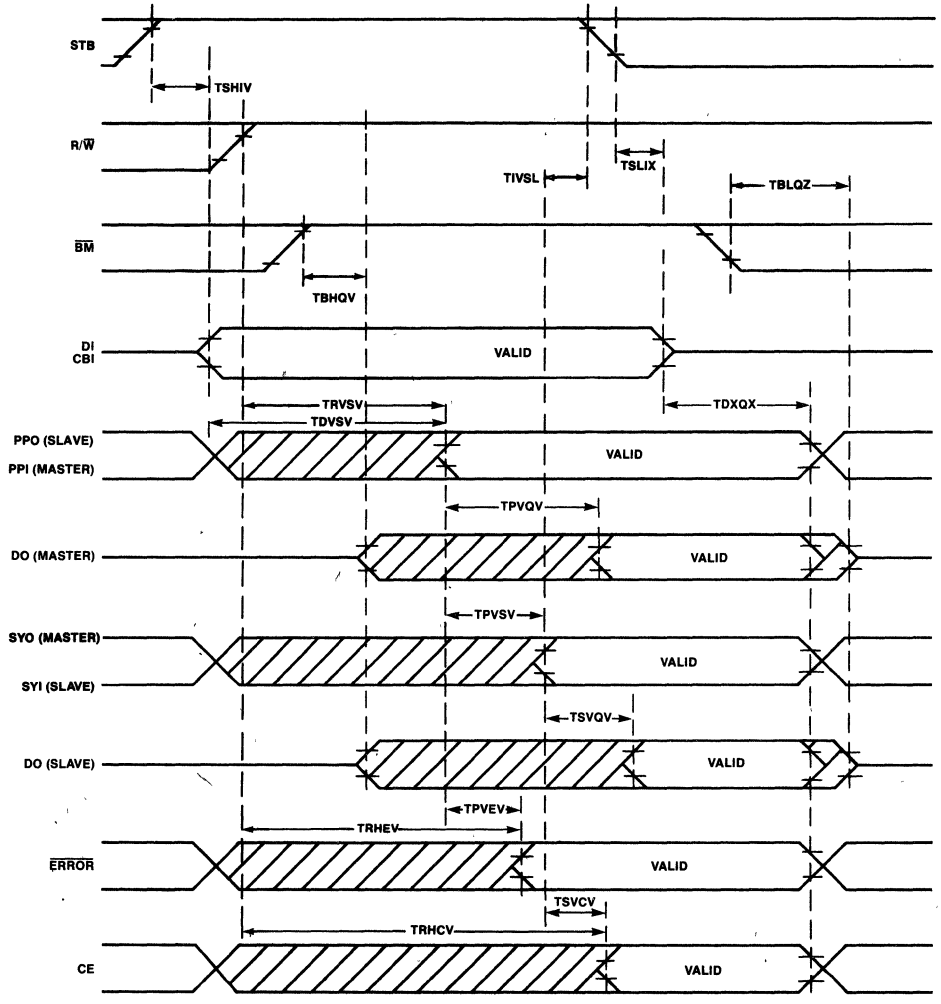
WAVEFORMS

READ



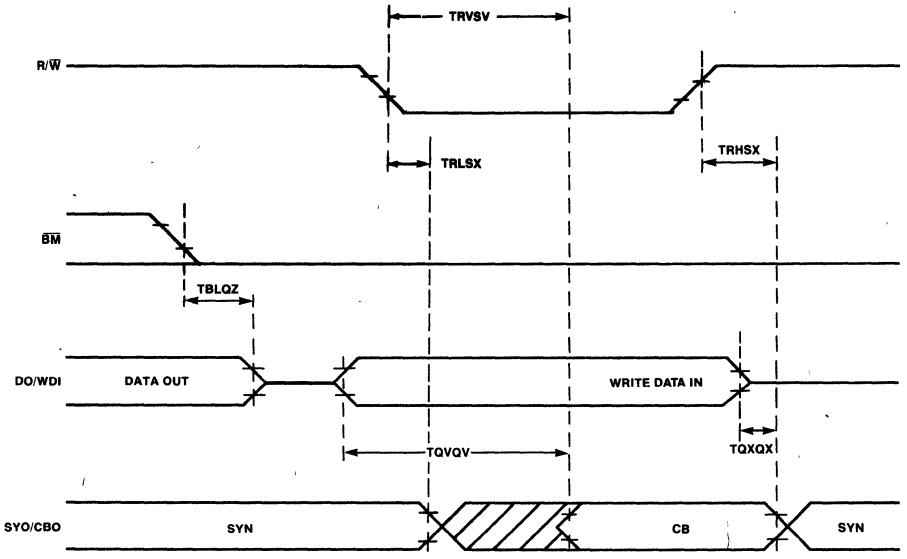
WAVEFORMS (Continued)

READ—MASTER/SLAVE

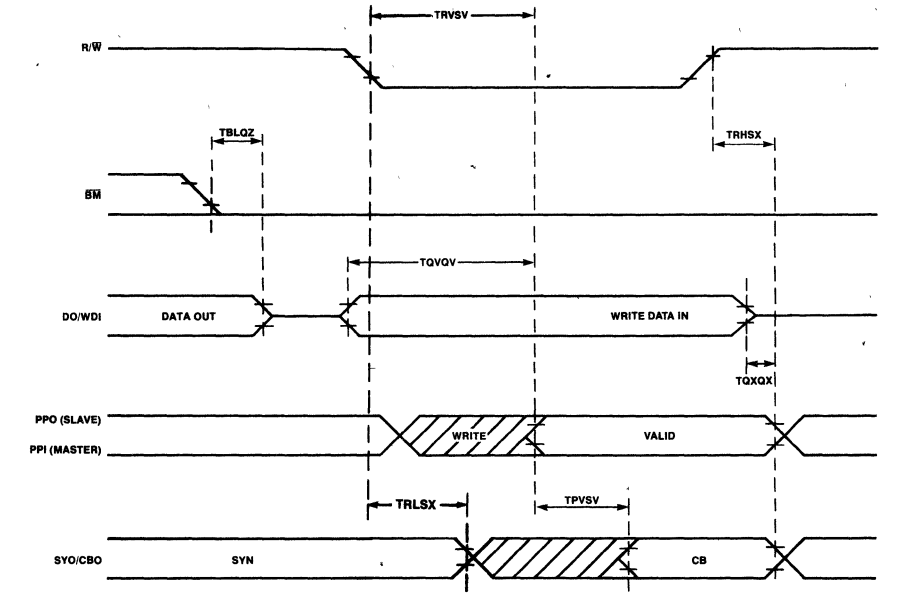


WAVEFORMS (Continued)

FULL WRITE

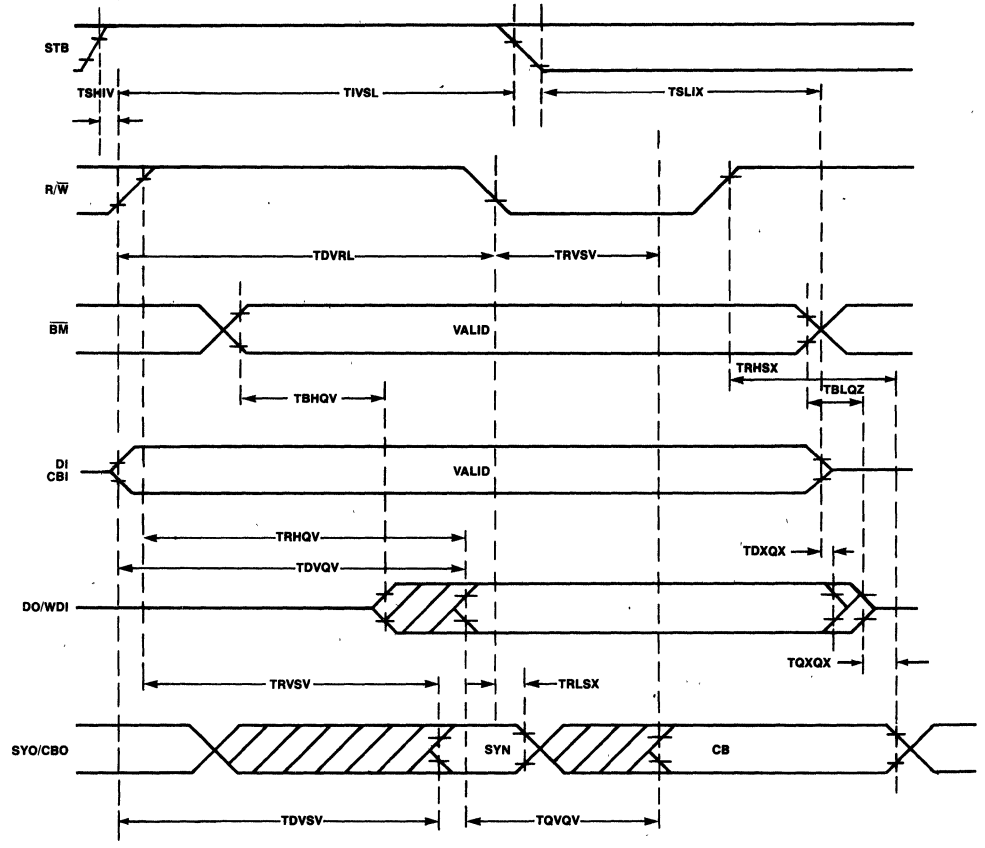


FULL WRITE—MASTER/SLAVE



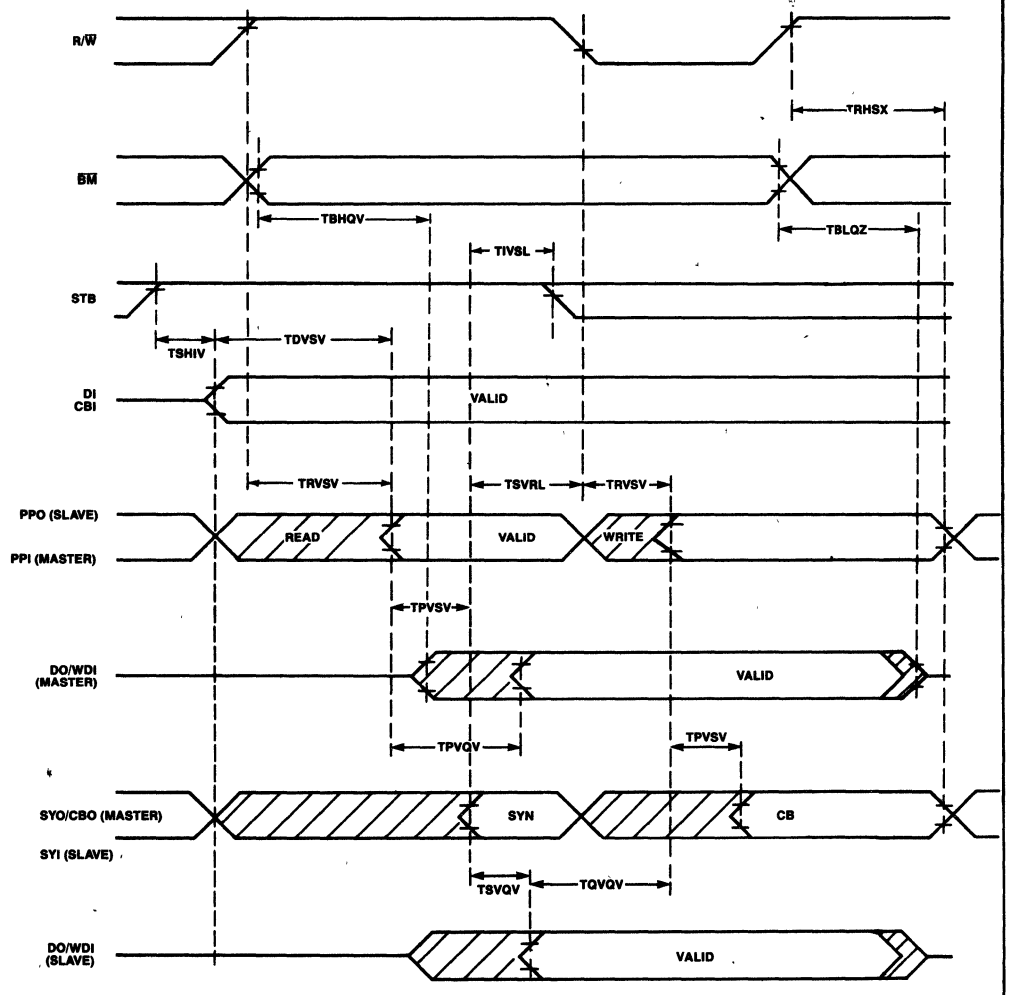
WAVEFORMS (Continued)

READ MODIFY WRITE



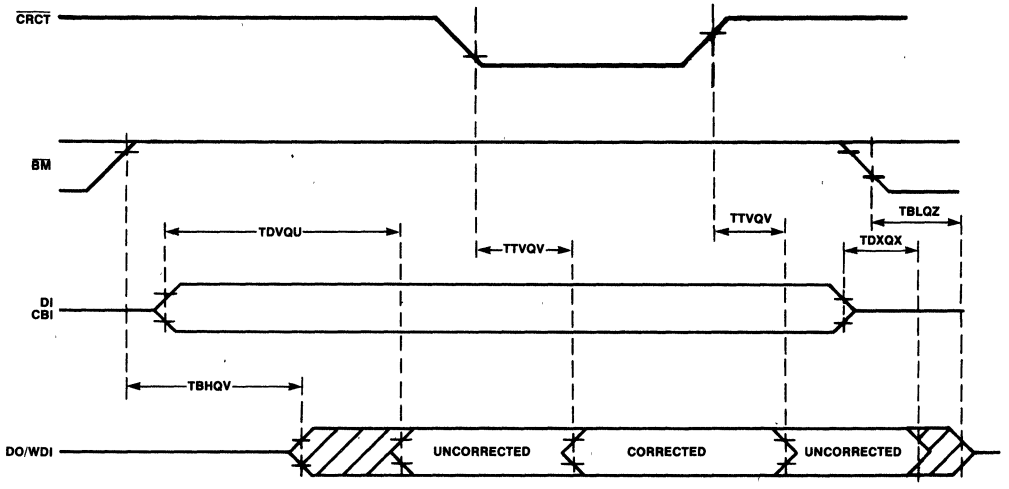
WAVEFORMS (Continued)

READ MODIFY WRITE—MASTER/SLAVE

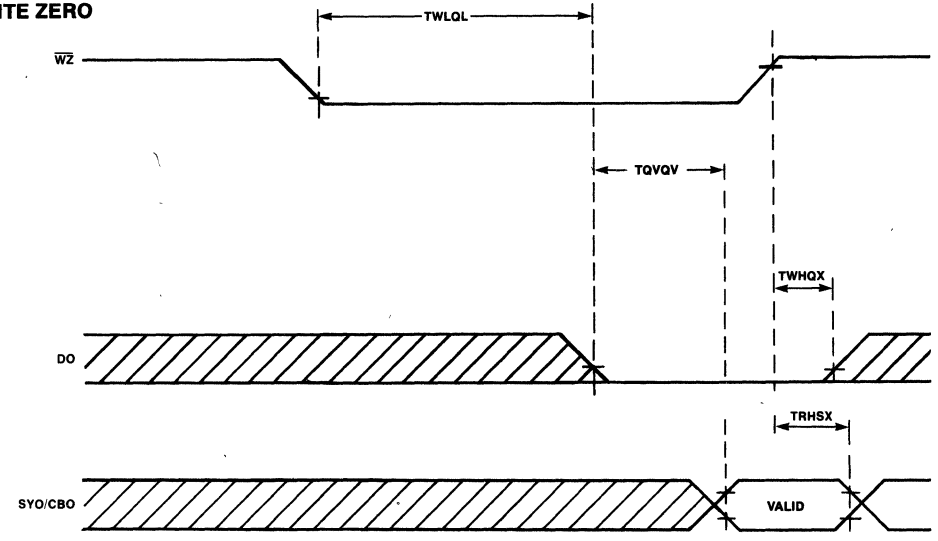


WAVEFORMS (Continued)

NON-CORRECTING READ



WRITE ZERO



A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, V_{SS} = 0V, R_L = 220Ω, C_L = 50 pF; all times are in nsec.)

Symbol	Parameter	8206		8206-2		Notes
		Min.	Max.	Min.	Max.	
TRHEV	$\overline{ERROR}$ Valid from R/$\overline{W}$↑		25		40	
TRHCV	CE Valid from R/$\overline{W}$↑ (Single 8206)		44		49	
TRHQV	Corrected Data Valid from R/$\overline{W}$↑		54		66	1
TRVSV	SYO/CBO/PPO Valid from R/$\overline{W}$		42		46	1
TDVEV	$\overline{ERROR}$ Valid from Data/Check Bits In		52		57	
TDVCV	CE Valid from Data/Check Bits In		70		76	
TDVQV	Corrected Data Valid from Data/Check Bits In		67		74	
TDVSV	SYO/PPO Valid from Data/Check Bits In		55		65	
TBHQV	Corrected Data Access Time		37		37	
TDXQX	Hold Time from Data/check Bits In	0		0		1
TBLQZ	Corrected Data Float Delay	0	28	0	28	1
TSHIV	STB High to Data/Check Bits In Valid	30		30		2
TIVSL	Data/Check Bits In to STB↓ Set-up	5		5		
TSLIX	Data/Check Bits In from STB↓ Hold	25		25		
TPVEV	$\overline{ERROR}$ Valid from Partial Parity In		30			3
TPVQV	Corrected Data (Master) from Partial Parity In		61			1,3
TPVSV	Syndrome/Check Bits Out from Partial Parity In		43			1,3
TSVQV	Corrected Data (Slave) Valid from Syndrome		51			3
TSVCV	CE Valid from Syndrome (Slave number 1)		48			3
TQVQV	Check Bits/Partial Parity Out from Write Data In		64		69	1
TRHSX	Check Bits/Partial Parity Out from R/W, $\overline{WZ}$ Hold	0		0		1
TRLSX	Syndrome Out from R/$\overline{W}$ Hold	0		0		
TQXQX	Hold Time from Write Data In	0		0		1
TSVRL	Syndrome Out to R/$\overline{W}$↓ Set-up	17				3
TDVRL	Data/Check Bits In to R/$\overline{W}$ Set-up	39		41		1
TDVQU	Uncorrected Data Out from Data In		32		38	
TTVQV	Corrected Data Out from $\overline{CRCT}$↓		30		33	
TWLQL	$\overline{WZ}$↓ to Zero Out		30		34	
TWHQX	Zero Out from $\overline{WZ}$↑ Hold	0		0		

NOTES:
1. A.C. Test Levels for CBO and DO are 2.4V and 0.8V.
2. T_{SHIV} is required to guarantee output delay timings: T_{DVEV}, T_{DVCV}, T_{DVQV}, T_{DVSV}. $T_{SHIV} + T_{IVSL}$ guarantees a min STB pulse width of 35 ns (45 ns for the 8206-8).
3. Not required for 8/16 bit systems

8206-2
16 BIT ERROR DETECTION AND CORRECTION UNIT

- **Detects and Corrects All Single Bit Errors**
- **Detects All Double Bit and Most Multiple Bit Errors**
- **Timing Optimized for 8 MHz iAPX 186, 188, 86, 88 and 8207-2 Systems**
- **Syndrome Outputs for Error Logging**
- **Separate Input and Output Busses—No Timing Strobes Required**
- **Supports Reads With and Without Correction, Writes, Partial (Byte) Writes, and Read-Modify-Writes**
- **HMOS Technology for Low Power**
- **68 Pin Leadless JEDEC Package**
- **Single +5V Supply**

The HMOS 8206 Error Detection and Correction Unit is a high-speed device that provides error detection and correction for memory systems (static and dynamic) requiring high reliability and performance. Each 8206-2 handles 8 or 16 data bits and up to 6 check bits. Other 8206-2 features include the ability to handle byte writes, memory initialization, and error logging.

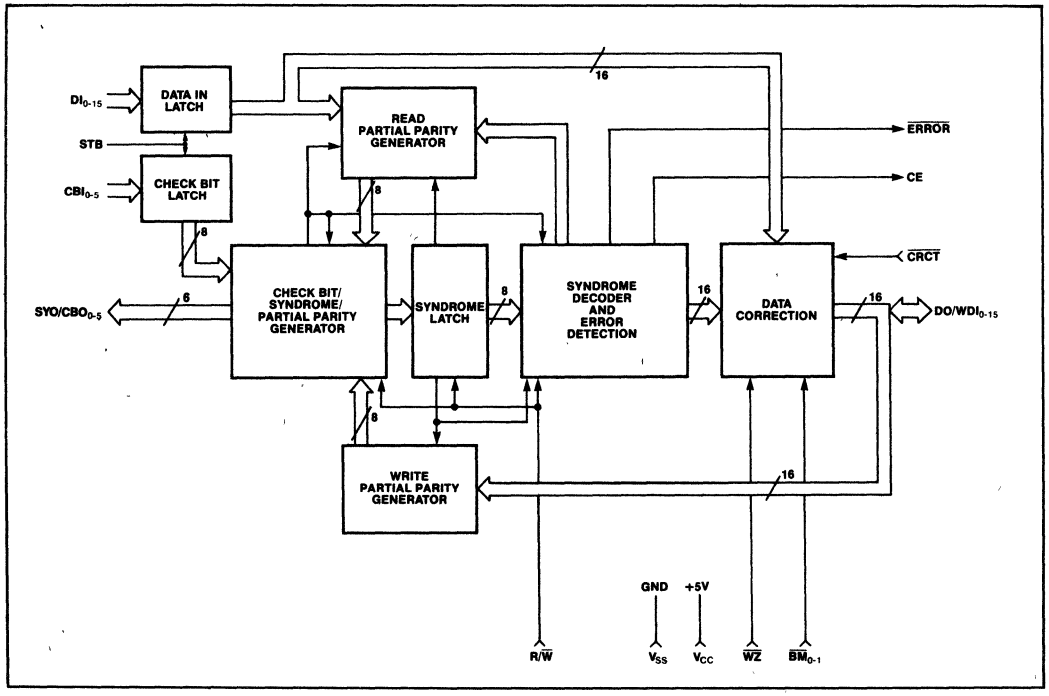


Figure 1. 8206 Block Diagram

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
DI_{0-15}	1, 68-61, 59-53	I	**Data In:** These inputs accept a 16 bit data word from RAM for error detection and/or correction.
CBI_0 CBI_1 CBI_2 CBI_3 CBI_4 CBI_5	5 6 7 8 9 10	I I I I I I	**Check Bits In:** In an 8206-2 system, these inputs accept the check bits (5 to 6) from the RAM.
DO/WDI_0 DO/WDI_1 DO/WDI_2 DO/WDI_3 DO/WDI_4 DO/WDI_5 DO/WDI_6 DO/WDI_7 DO/WDI_8 DO/WDI_9 DO/WDI_{10} DO/WDI_{11} DO/WDI_{12} DO/WDI_{13} DO/WDI_{14} DO/WDI_{15}	51 50 49 48 47 46 45 44 42 41 40 39 38 37 36 35	I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O I/O	**Data Out/Write Data In:** In a read cycle, data accepted by DI_{0-15} appears at these outputs corrected if $\overline{CRCT}$ is low, or uncorrected if $\overline{CRCT}$ is high. The $\overline{BM}$ inputs must be high to enable the output buffers during the read cycle. In a write cycle, data to be written into the RAM is accepted by these inputs for computing the write check bits. In a partial-write cycle, the byte not to be modified appears at either DO_{0-7} if $\overline{BM}_0$ is high, or DO_{8-15} if $\overline{BM}_1$ is high, for writing to the RAM. When $\overline{WZ}$ is active, it causes the 8206 to output all zeros at DO_{0-15}, with the proper write check bits on CBO.
SYO/CBO_0 SYO/CBO_1 SYO/CBO_2 SYO/CBO_3 SYO/CBO_4 SYO/CBO_5	23 24 25 27 28 29	O O O O O O	**Syndrome Out/Check Bits Out:** In an 8206-2 system, the syndrome appears at these outputs during a read. During a write, the write check bits appear. The syndrome is latched (during read-modify-writes) by $R/\overline{W}$ going low.
CE	17	O	**Correctable Error:** In an 8206-2 system, this pin outputs the correctable error flag. CE is latched by $R/\overline{W}$ going low.
$\overline{ERROR}$	22	O	**Error:** This pin outputs the error flag in an 8206-2 system. It is latched by $R/\overline{W}$ going low.
$\overline{CRCT}$	52	I	**Correct:** When low this pin causes data correction during a read or read-modify-write cycle. When high, it causes error correction to be disabled, although error checking is still enabled.
STB	2	I	**Strobe:** STB is an input control used to strobe data at the DI inputs and check-bits at the CBI/SYI inputs. The signal is active high to admit the inputs. The signals are latched by the high-to-low transition of STB.
$\overline{BM}_0$ $\overline{BM}_1$	33 32	I I	**Byte Marks:** When high, the Data Out pins are enabled for a read cycle. When low, the Data Out buffers are tristated for a write cycle. $\overline{BM}_0$ controls DO_{0-7}, while $\overline{BM}_1$ controls DO_{8-15}. In partial (byte) writes, the byte mark input is low for the new byte to be written.
$R/\overline{W}$	21	I	**Read/Write:** When high this pin causes the 8206-2 to perform detection and correction (if $\overline{CRCT}$ is low). When low, it causes the 8206-2 to generate check bits. On the high-to-low transition the syndrome is latched internally for read-modify-write cycles.
$\overline{WZ}$	34	I	**Write Zero:** When low this input overrides the $\overline{BM}_{0-1}$ and $R/\overline{W}$ inputs to cause the 8206 to output all zeros at DO_{0-15} with the corresponding check bits at CBO_{0-5}. Used for memory initialization.
Strap High	4	I	Must be tied High.
Strap Low	3	I	Must be tied Low.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
N.C.	11-16 18-20	I	**Note:** These pins have internal pull-up resistors but if possible should be tied high or low.
N.C.	30, 31	O	**Note:** These are no connect pins and should be left open.
V_{CC}	60	I	**Power Supply:** +5V
V_{SS}	26	I	**Logic Ground**
V_{SS}	43	I	**Output Driver Ground**

FUNCTIONAL DESCRIPTION

The 8206-2 16 Bit Error Detection and Correction Unit provides greater memory system reliability through its ability to detect and correct memory errors. It is a single chip device that can detect and correct all single bit errors and detect all double bit and some higher multiple bit errors. Some other odd multiple bit errors (e.g., 5 bits in error) are interpreted as single bit errors, and the CE flag is raised. While some even multiple bit errors (e.g., 4 bits in error) are interpreted as no error, most are detected as double bit errors. Errors in check bits are not distinguished from errors in a word.

For more information on error correction codes, see Intel Application Notes AP-46 and AP-73.

The 8206-2 has a "flow through" architecture. It supports two kinds of error correction architecture: 1) Flow-through, or correct-always; and 2) Parallel, or check-only. There are two separate 16-pin busses, one to accept data from the RAM (DI) and the other to deliver corrected data to the system bus (DO/WDI). The logic is entirely combinatorial during a read cycle. This is in contrast to an architecture with only one bus, with bidirectional bus drivers that must first read the data and then be turned around to output the corrected data. The latter architecture typically requires additional hardware (latches and/or transceivers) and may be slower in a system due to timing skews of control signals.

READ CYCLE

With the R/$\overline{W}$ pin high, data is received from the RAM outputs into the DI pins where it is optionally latched by the STB signal. Check bits are generated from the data bits and compared to the check bits read from the RAM into the CBI pins. If an error is detected the $\overline{ERROR}$ flag is activated and the correctable error flag (CE) is used to inform the system whether the error was correctable or not. With the $\overline{BM}$ inputs high, the word appears corrected at the DO pins if the error was correctable, or unmodified if the error was uncorrectable.

The 8206-2 may alternatively be used in a "check-only" mode with the $\overline{CRCT}$ pin left high. With the correction facility turned off, the propagation delay from memory outputs to 8206-2 outputs is significantly shortened. In this mode the 8206-2 issues an $\overline{ERROR}$ flag to the CPU, which can then perform one of several options: lengthen the current cycle for correction, restart the instruction, perform a diagnostic routine, etc.

A syndrome word, five to six bits in length and containing all necessary information about the existence and location of an error, is made available to the system at the SYO_{0-5} pins. Error logging may be accomplished by latching the syndrome and the memory address of the word in error.

WRITE CYCLE

For a full write, in which an entire word is written to memory, the data is written directly to the RAM, bypassing the 8206-2. The same data enters the 8206-2 through the WDI pins where check bits are generated. The Byte Mark inputs must be low to tristate the DO drivers. The check bits, 5 to 6 in number, are then written to the RAM through the CBO pins for storage along with the data word.

In a partial write, part of the data word is overwritten, and part is retained in memory. This is accomplished by performing a read-modify-write cycle. The complete old word is read into the 8206-2 and corrected, with the syndrome internally latched by R/$\overline{W}$ going low. Only that part of the word not to be modified is output onto the DO pins, as controlled by the Byte Mark inputs. That portion of the word to be overwritten is supplied by the system bus. The 8206-2 then calculates check bits for the new word, using the byte from the previous read and the new byte from the system bus, and writes them to the memory.

READ-MODIFY-WRITE CYCLES

Upon detection of an error the 8206-2 may be used to correct the bit in error in memory. This reduces the probability of getting multiple-bit errors in subsequent read cycles. This correction is handled by executing read-modify-write cycles.

The read-modify-write cycle is controlled by the R/$\overline{W}$ input. After (during) the read cycle, the system dynamic RAM controller or CPU examines the 8206-2 $\overline{ERROR}$ and CE outputs to determine if a correctable error occurred. If it did, the dynamic RAM controller or CPU forces R/$\overline{W}$ low, telling the 8206-2 to latch the generated syndrome and drive the corrected check bits onto the CBO outputs. The corrected data is available on the DO pins. The DRAM controller then writes the corrected data and corresponding check bits into memory.

The 8206-2 may be used to perform read-modify-writes in one or two RAM cycles. If it is done in two cycles, the 8206-2 latches are used to hold the data and check bits from the read cycle to be used in the following write cycle. The Intel 8207-2 Advanced Dynamic RAM controller allows read-modify-write cycles in one memory cycle. See the System Environment section.

INITIALIZATION

A memory system operating with ECC requires some form of initialization at system power-up in order to set valid data and check bit information in memory. The 8206-2 supports memory initialization by the Write Zero function. By activating the W$\overline{Z}$ pin, the 8206-2 will write a data pattern of zeros and the associated check bits in the current write cycle. By thus writing to all memory at power-up, a controller can set memory to valid data and check bits. Massive memory failure, as signified by both data and check bits all ones or zeros, will be detected as an uncorrectable error.

HAMMING CODE

The 8206-2 uses a modified Hamming code. The code appears in Table 2. The check bits are derived from the table by XORing or XNORing together the bits indicated by 'X's in each row corresponding to a check bit. For example, check bit 0 for data word 1000110101101011 will be "0." It should be noted that the 8206-2 will detect the gross-error condition of all lows or all highs.

Error correction is accomplished by identifying the bad bit and inverting it. Table 2 can also be used as an error syndrome table by replacing the 'X's with '1's. Each column then represents a different syndrome word, and by locating the column corresponding to a particular syndrome the bit to be corrected may be identified. If the syndrome cannot be located then the error cannot be corrected. For example, if the syndrome word is 101100 the bit to be corrected is bit 10 in the slave one data word.

The syndrome decoding is also summarized in Table 3 which can be used for error logging. By finding the appropriate syndrome word (starting with bit zero, the least significant bit), the result is either; 1) no error; 2) an identified (correctable) single bit error; or 3) a double bit error.

Table 2. Modified Hamming Code Check Bit Generation

Check bits are generated by XOR'ing (except for the CB0 and CB1 data bits, which are XNOR'ed) the data bits in the rows corresponding to the check bits. Note there are 5 check bits in an 8-bit system and 6 check bits in a 16-bit system.

BYTE NUMBER		0								1								OPERATION
BIT NUMBER		0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	
CHECK BITS	CB0 =	x	x	-	x	-	x	x	-	x	-	-	x	-	x	-	-	XNOR
	CB1 =	x	-	x	-	-	x	-	x	-	x	-	x	x	-	x	-	XNOR
	CB2 =	-	x	x	-	x	-	x	x	-	-	x	-	x	-	-	x	XOR
	CB3 =	x	x	x	x	x	-	-	-	x	x	x	-	-	-	-	-	XOR
	CB4 =	-	-	-	x	x	x	x	x	-	-	-	-	-	x	x	x	XOR
	CB5 =	-	-	-	-	-	-	-	-	x	x	x	x	x	x	x	x	XOR
DATA BITS		0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	1 0	1 1	1 2	1 3	1 4	1 5	

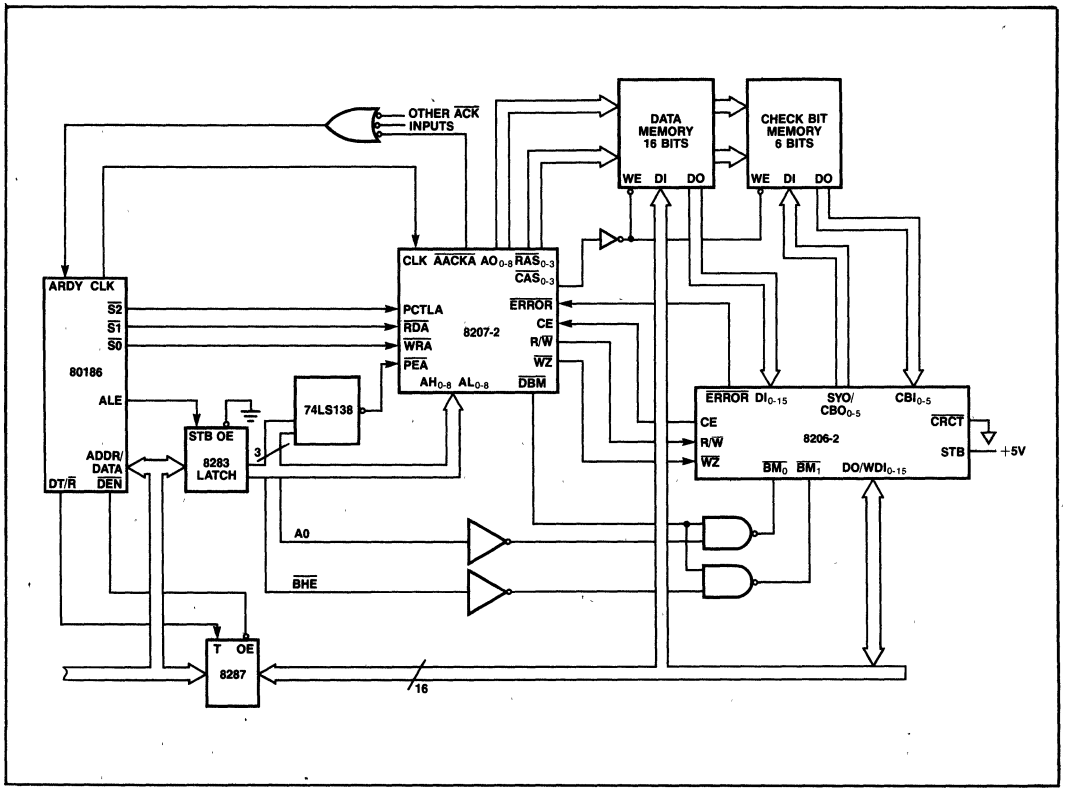
Table 7. 8206-2 Syndrome Decoding

Syndrome Bits 5	4	3	0 1 2	0 0 0	1 0 0	0 1 0	1 1 0	0 0 1	1 0 1	0 1 1	1 1 1
0	0	0		N	CB0	CB1	D	CB2	D	D	D
0	0	1		CB3	D	D	0	D	1	2	D
0	1	0		CB4	D	D	5	D	6	7	D
0	1	1		D	3	D	D	4	D	D	D
1	0	0		CB5	D	D	11	D	D	12	D
1	0	1		D	8	9	D	10	D	D	D
1	1	0		D	13	14	D	15	D	D	D
1	1	1		D	D	D	D	D	D	D	D

N = No Error
CBX = Error in Check Bit X
X = Error in Data Bit X
D = Double Bit Error

The 8206-2 handles 8 or 16 bits of data. For 8 bit 8206-2 systems, the DI_{8-15}, DO/WDI_{8-15} and $\overline{BM}_1$ inputs are grounded.

The 8206-2 is designed for direct connection to the Intel 8207-2 Advanced Dynamic RAM Controller. The 8207-2 has the ability to perform dual port memory control, and Figure 7 illustrates a highly integrated iAPX 186 RAM implementation using the 8206-2 and 8207-2. The 8206-2/8207-2 combination permits such features as automatic scrubbing (correcting errors in memory during refresh), extending RAS and CAS timings for Read-Modify-Writes in single memory cycles, and automatic memory initialization upon reset. Together these two chips provide a complete dual-port, error-corrected dynamic RAM subsystems.



MEMORY BOARD TESTING

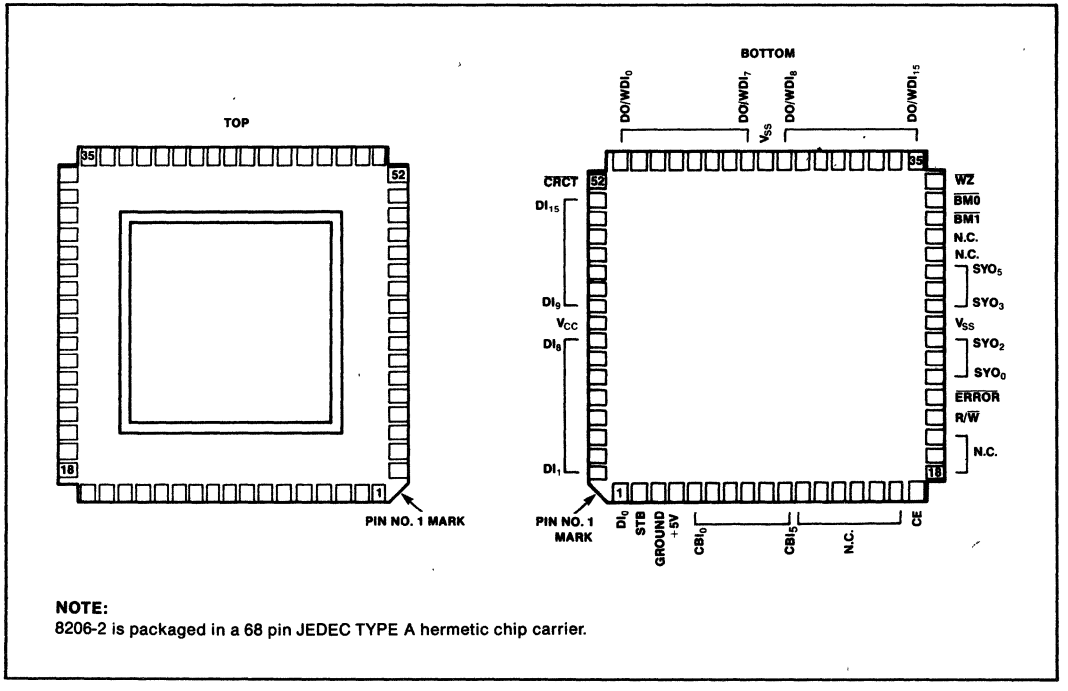
The 8206-2 lends itself to straightforward memory board testing with a minimum of hardware overhead. The following is a description of four common test modes and their implementation.

Mode 0—Read and write with error correction.
Implementation: This mode is the normal 8206-2 operating mode.

Mode 1—Read and write data with error correction disabled to allow test of data memory.
Implementation: This mode is performed with $\overline{CRCT}$ deactivated.

Mode 2—Read and write check bits with error correction disabled to allow test of check bits memory.
Implementation: Any pattern may be written into the check bits memory by judiciously choosing the proper data word to generate the desired check bits, through the use of the 8206-2 Hamming code. To read out the check bits it is first necessary to fill the data memory with all zeros, which may be done by activating $\overline{WZ}$ and incrementing memory addresses with $\overline{WE}$ to the check bits memory held inactive, and then performing ordinary reads. The check bits will then appear directly at the SYO outputs, with bits CB0 and CB1 inverted.

Mode 3—Write data, without altering or writing check bits, to allow the storage of bit combinations to cause error correction and detection.
Implementation: This mode is implemented by writing the desired word to memory with $\overline{WE}$ to the check bits array held inactive.



NOTE:
8206-2 is packaged in a 68 pin JEDEC TYPE A hermetic chip carrier.

Figure 3. 8206 Pinout Diagram

ABSOLUTE MAXIMUM RATINGS*

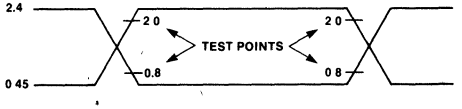
Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin
With Respect to Ground ..... −0.5V to +7V
Power Dissipation ..... 2.5 Watts

**NOTE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5.0V ± 10%, V_{SS} = GND)

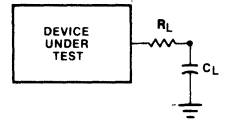
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
I_{CC}	Power Supply Current		270	mA	
V_{IL}	Input Low Voltage	−0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}+ 0.5V	V	
V_{OL}	Output Low Voltage —DO —All Other Outputs		 0.45 0.45	 V V	 I_{OL} = 8mA I_{OL} = 2.0mA
V_{OH}	Output High Voltage —DO, CBO —All Other Outputs	 2.6 2.4		 V V	 I_{OH} = −2mA I_{OH} = −0.4mA
I_{LO}	I/O Leakage Current —CE —DO/WDI_{0-15}		 ±20 ±10	 μA μA	$0.45V \leq V_{I/O} \leq V_{CC}$
I_{LI}	Input Leakage Current		±10	μA	$0V \leq V_{IN} \leq V_{CC}$

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC "1" AND 0 8V FOR A LOGIC "0"

A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

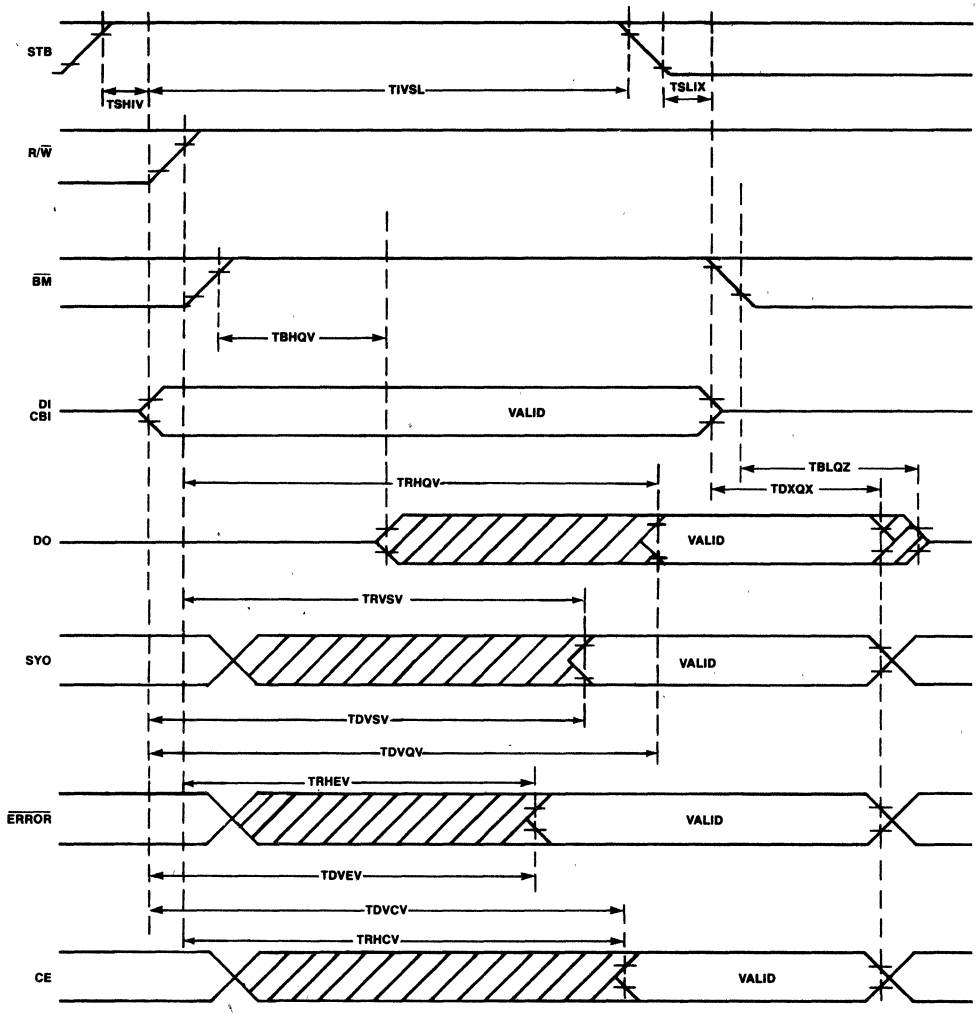
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, V_{SS} = 0V, R_L = 22Ω C_L = 50pF; all times are in nsec.)

Symbol	Parameter	8206-2 Min.	8206-2 Max.	Notes
TRHEV	$\overline{ERROR}$ Valid from R/$\overline{W}$↑		40	
TRHCV	CE Valid from R/$\overline{W}$↑		49	
TRHQV	Corrected Data Valid from R/$\overline{W}$↑		66	1
TRVSV	SYO/CBO Valid from R/$\overline{W}$		46	1
TDVEV	$\overline{ERROR}$ Valid from Data/Check Bits In		57	
TDVCV	CE Valid from Data/Check Bits In		76	
TDVQV	Corrected Data Valid from Data/Check Bits In		74	
TDVSV	SYO Valid from Data/Check Bits In		65	
TBHQV	Corrected Data Access Time		37	
TDXQX	Hold Time from Data/Check Bits In	0		1
TBLQZ	Corrected Data Float Delay	0	28	1
TSHIV	STB High to Data/Check Bits In Valid	30		2
TIVSL	Data/Check Bits In to STB↓ Set-up	5		
TSLIX	Data/Check Bits In from STB↓ Hold	25		
TQVQV	Check Bits Out from Write Data In		69	1
TRHSX	Check Bits Out from R/$\overline{W}$, $\overline{WZ}$ Hold	0		1
TRLSX	Syndrome Out from R/$\overline{W}$ Hold	0		
TQXQX	Hold Time from Write Data In	0		1
TDVRL	Data/Check Bits In to R/$\overline{W}$ Set-up	41		1
TDVQU	Uncorrected Data Out from Data In		38	
TTVQV	Corrected Data Out from $\overline{CRCT}$↓		33	
TWLQL	$\overline{WZ}$↓ to Zero Out		34	
TWHQX	Zero Out from $\overline{WZ}$↑ Hold	0		

NOTES:
1. A.C. Test Levels for CBO and DO are 2.4V and 0.8V.
2. T_{SHIV} is required to guarantee output delay timings: T_{DVEV}, T_{DVCV}, T_{DVQV}, T_{DVSV}. T_{SHIV} + T_{IVSL} guarantees a min STB pulse width of 35 ns.

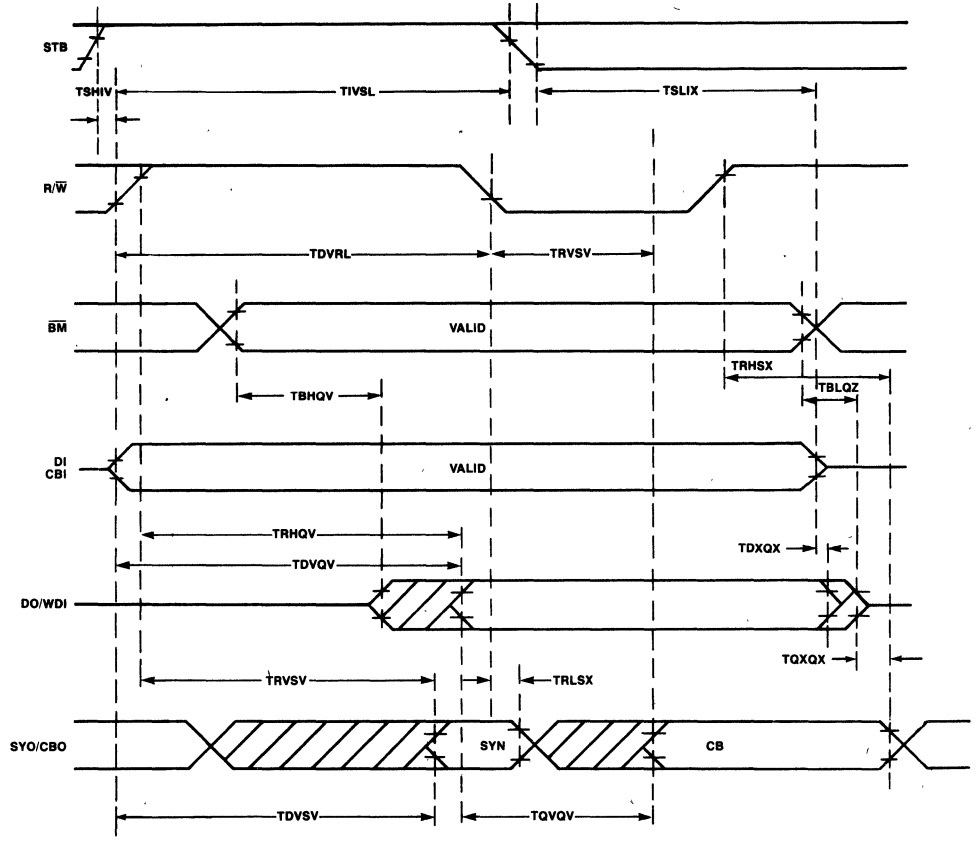
WAVEFORMS

READ



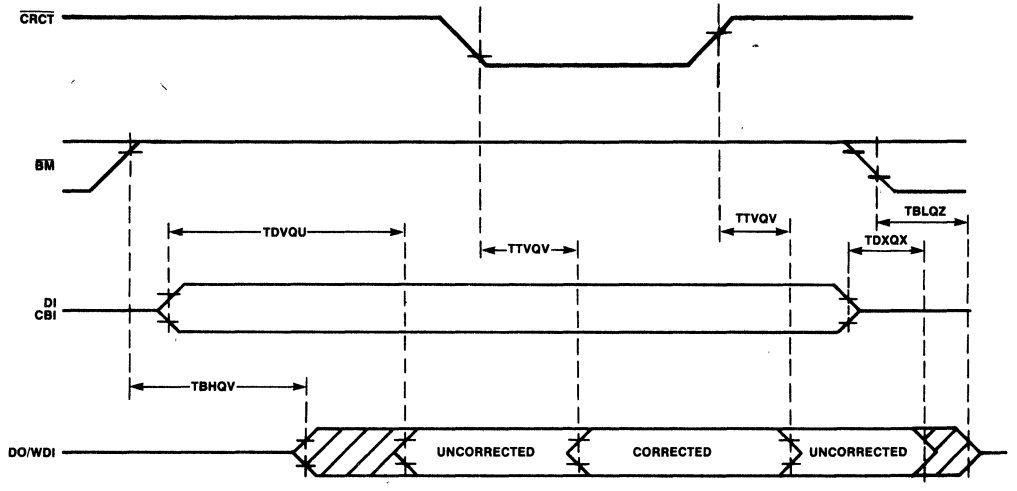
WAVEFORMS (Continued)

READ MODIFY WRITE

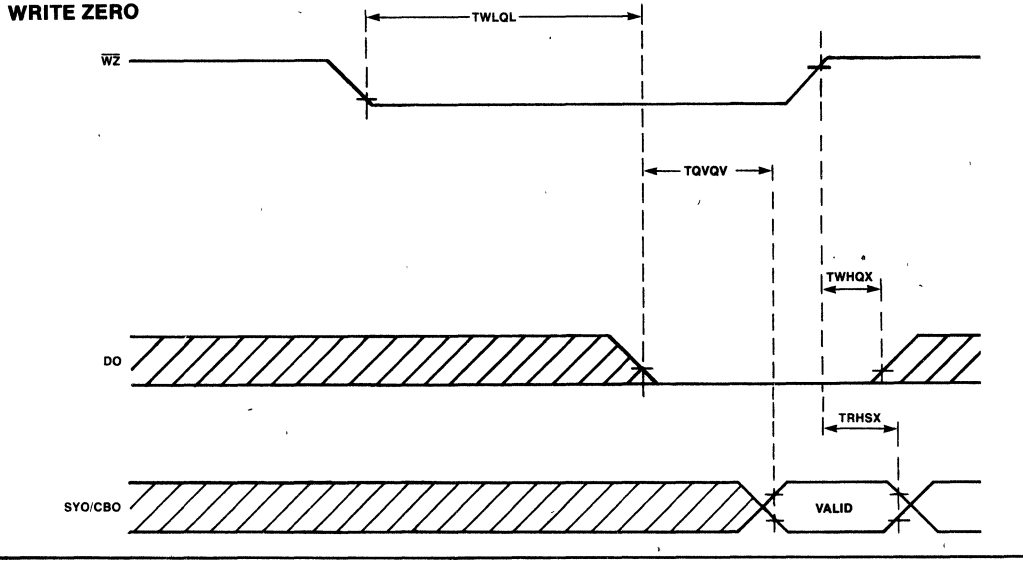


WAVEFORMS (Continued)

NON-CORRECTING READ

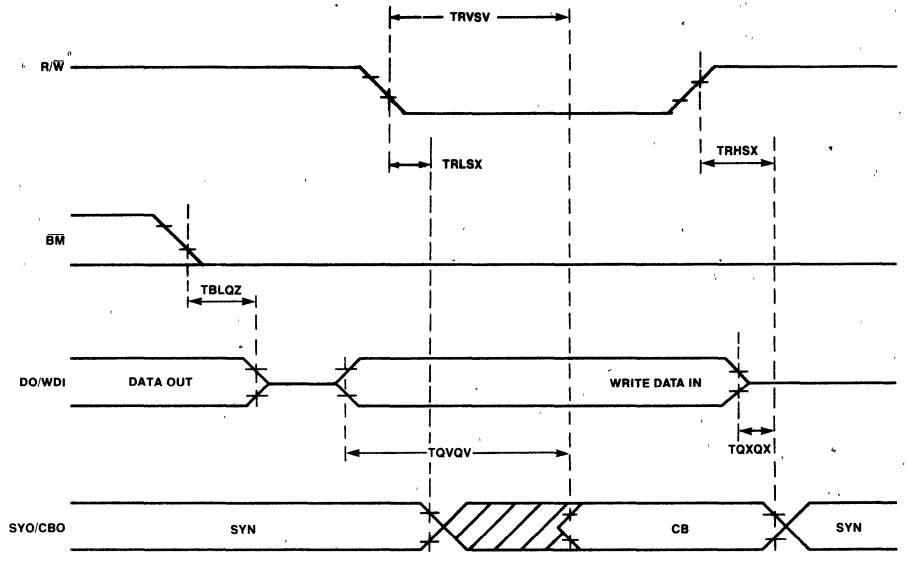


WRITE ZERO



WAVEFORMS (Continued)

FULL WRITE



PRELIMINARY

8207
ADVANCED DYNAMIC RAM CONTROLLER

- **Provides All Signals Necessary to Control 16K (2118), 64K (2164A) and 256K Dynamic RAMs**
- **Directly Addresses and Drives up to 2 Megabytes without External Drivers**
- **Supports Single and Dual-Port Configurations**
- **Automatic RAM Initialization in All Modes**
- **Four Programmable Refresh Modes**
- **Transparent Memory Scrubbing in ECC Mode**
- **Supports Intel iAPX 86, 88, 186, 188, and 286 Microprocessors**
- **Data Transfer and Advance Acknowledge Signals for Each Port**
- **Provides Signals to Directly Control the 8206 Error Detection and Correction Unit**
- **Supports Synchronous or Asynchronous Operation on Either Port**
- **+5 Volt Only HMOSII Technology for High Performance and Low Power**

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a high-performance, systems-oriented, Dynamic RAM controller that is designed to easily interface 16K, 64K and 256K Dynamic RAMs to Intel and other microprocessor systems. A dual-port interface allows two different busses to independently access memory. When configured with an 8206 Error Detection and Correction Unit the 8207 supplies the necessary logic for designing large error-corrected memory arrays. This combination provides automatic memory initialization and transparent memory error scrubbing.

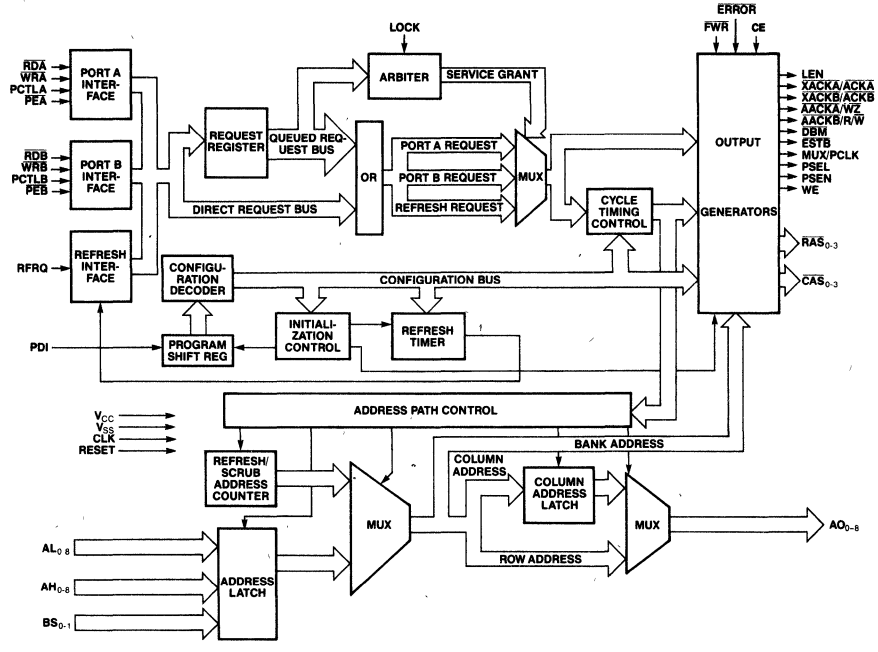


Figure 1. 8207 Block Diagram

JULY 1983
ORDER NUMBER: 210463-003

Table 1. Pin description

Symbol	Pin	Type	Name and Function
LEN	1	O	**ADDRESS LATCH ENABLE:** In two-port configurations, when Port A is running with iAPX 286 Status interface mode, this output replaces the ALE signal from the system bus controller of port A and generates an address latch enable signal which provides optimum setup and hold timing for the 8207. This signal is used in Fast Cycle operation only.
$\overline{\text{XACKA}}$/ $\overline{\text{ACKA}}$	2	O	**TRANSFER ACKNOWLEDGE PORTA/ACKNOWLEDGE PORTA:** In non-ECC mode, this pin is $\overline{\text{XACKA}}$ and inidcates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port A. $\overline{\text{XACKA}}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{\text{ACKA}}$ which can be configured, depending on the programming of the X program bit, as an $\overline{\text{XACK}}$ or $\overline{\text{AACK}}$ strobe. The SA programming bit determines whether the $\overline{\text{AACK}}$ will be an early $\overline{\text{EAACKA}}$ or a late $\overline{\text{LAACKA}}$ interface signal.
$\overline{\text{XACKB}}$/ $\overline{\text{ACKB}}$	3	O	**TRANSFER ACKNOWLEDGE PORT B/ACKNOWLEDGE PORT B:** In non-ECC mode, this pin is XACKB and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle for Port B. $\overline{\text{XACKB}}$ is a Multibus-compatible signal. In ECC mode, this pin is $\overline{\text{ACKB}}$ which can be configured, depending on the programming of the X program bit, as an $\overline{\text{XACK}}$ or $\overline{\text{AACK}}$ strobe. The SB programming bit determines whether the $\overline{\text{AACK}}$ will be an early $\overline{\text{EAACKB}}$ or a late $\overline{\text{LAACKB}}$ interface signal.
$\overline{\text{AACKA}}$/ $\overline{\text{WZ}}$	4	O	**ADVANCED ACKNOWLEDGE PORT A/WRITE ZERO:** In non-ECC mode, this pin is $\overline{\text{AACKA}}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SA program bit for synchronous or asynchronous operation. In ECC mode, after a RESET, this signal will cause the 8206 to force the data to all zeros and generate the appropriate check bits.
$\overline{\text{AACKB}}$/ R/$\overline{\text{W}}$	5	O	**ADVANCED ACKNOWLEDGE PORT B/READ/WRITE:** In non-ECC mode, this pin is $\overline{\text{AACKB}}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the SB program bit for synchronous or asynchronous operation. In ECC mode, this signal causes the 8206 EDCU to latch the syndrome and error flags and generate check bits.
$\overline{\text{DBM}}$	6	O	**DISABLE BYTE MARKS:** This is an ECC control output signal indicating that a read or refresh cycle is occurring. This output forces the byte address decoding logic to enable all 8206 data output buffers. In ECC mode, this output is also asserted during memory initialization and the 8-cycle dynamic RAM wake-up exercise. In non-ECC systems this signal indicates that either a read, refresh or 8-cycle warm-up is in progress.
$\overline{\text{ESTB}}$	7	O	**ERROR STROBE:** In ECC mode, this strobe is activated when an error is detected and allows a negative-edge triggered flip-flop to latch the status of the 8206 EDCU CE for systems with error logging capabilities. $\overline{\text{ESTB}}$ will not be issued during refresh cycles.
LOCK	8	I	**LOCK:** This input instructs the 8207 to lock out the port not being serviced at the time LOCK was issued.
V_{CC}	9 43	I	**DRIVER POWER:** +5 Volts. Supplies V_{CC} for the output drivers. **LOGIC POWER:** +5 Volts. Supplies V_{CC} for the internal logic circuits.
CE	10	I	**CORRECTABLE ERROR:** This is an ECC input from the 8206 EDCU which instructs the 8207 whether a detected error is correctable or not. A high input indicates a correctable error. A low input inhibits the 8207 from activating WE to write the data back into RAM. This should be connected to the CE output of the 8206.
$\overline{\text{ERROR}}$	11	I	**ERROR:** This is an ECC input from the 8206 EDCU and instructs the 8207 that an error was detected. This pin should be connected to the ERROR output of the 8206.
MUX/ PCLK	12	O	**MULTIPLEXER CONTROL/PROGRAMMING CLOCK:** Immediately after a RESET this pin is used to clock serial programming data into the PDI pin. In normal two-port operation, this pin is used to select memory addresses from the appropriate port. When this signal is high, port A is selected and when it is low, port B is selected. This signal may change state before the completion of a RAM cycle, but the RAM address hold time is satisfied.
PSEL	13	O	**PORT SELECT:** This signal is used to select the appropriate port for data transfer. When this signal is high port A is selected and when it is low port B is selected.
PSEN	14	O	**PORT SELECT ENABLE:** This signal used in conjunction with PSEL provides contention-free port exchange on the data bus. When PSEN is low, port selection is allowed to change state.
WE	15	O	**WRITE ENABLE:** This signal provides the dynamic RAM array the write enable input for a write operation.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
$\overline{FWR}$	16	I	**FULL WRITE:** This is an ECC input signal that instructs the 8207, in an ECC configuration, whether the present write cycle is normal RAM write (full write) or a RAM partial write (read-modify-write) cycle.
RESET	17	I	**RESET:** This signal causes all internal counters and state flip-flops to be reset and upon release of RESET, data appearing at the PDI pin is clocked in by the PCLK output. The states of the PDI, PCTLA, PCTLB and RFRQ pins are sampled by RESET going inactive and are used to program the 8207. An 8-cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8207.
$\overline{CAS0}$ $\overline{CAS1}$ $\overline{CAS2}$ $\overline{CAS3}$	18 19 20 21	O O O O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{RAS0}$ $\overline{RAS1}$ $\overline{RAS2}$ $\overline{RAS3}$	22 23 24 25	O O O O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS0 and BS1 as programmed by program bits RB0 and RB1. These outputs drive the dynamic RAM array directly and need no external drivers.
V_{SS}	26 60	I I	**DRIVER GROUND:** Provides a ground for the output drivers. **LOGIC GROUND:** Provides a ground for the remainder of the device.
AO0 AO1 AO2 AO3 AO4 AO5 AO6 AO7 AO8	35 34 33 32 31 30 29 28 27	O O O O O O O O O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses of the selected port to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.
BS0 BS1	36 37	I I	**BANK SELECT:** These inputs are used to select one of four banks of the dynamic RAM array as defined by the program bits RB0 and RB1.
AL0 AL1 AL2 AL3 AL4 AL5 AL6 AL7 AL8	38 39 40 41 42 44 45 46 47	I I I I I I I I I	**ADDRESS LOW:** These lower-order address inputs are used to generate the row address for the internal address multiplexer.
AH0 AH1 AH2 AH3 AH4 AH5 AH6 AH7 AH8	48 49 50 51 52 53 54 55 56	I I I I I I I I I	**ADDRESS HIGH:** These higher-order address inputs are used to generate the column address for the internal address multiplexer.
PDI	57	I	**PROGRAM DATA INPUT:** This input programs the various user-selectable options in the 8207. The PCLK pin shifts programming data into the PDI input from optional external shift registers. This pin may be strapped high or low to a default ECC (PDI =Logic "I") or non-ECC (PDI = Logic "O") mode configuration.
RFRQ	58	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If it is high at RESET, then the 8207 is programmed for internal refresh request or external refresh request with failsafe protection. If it is low at RESET, then the 8207 is programmed for external refresh without failsafe protection or burst refresh. Once programmed the RFRQ pin accepts signals to start an external refresh with failsafe protection or external refresh without failsafe protection or a burst refresh.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
CLK	59	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic.
$\overline{RDB}$	61	I	**READ FOR PORT B:** This pin is the read memory request command input for port B. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRB}$	62	I	**WRITE FOR PORT B:** This pin is the write memory request command input for port B. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEB}$	63	I	**PORT ENABLE FOR PORT B:** This pin serves to enable a RAM cycle request for port B. It is generally decoded from the port address.
PCTLB	64	I	**PORT CONTROL FOR PORT B:** This pin is sampled on the falling edge of RESET. It configures port B to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be used as a Multibus-compatible inhibit signal.
$\overline{RDA}$	65	I	**READ FOR PORT A:** This pin is the read memory request command input for port A. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
$\overline{WRA}$	66	I	**WRITE FOR PORT A:** This pin is the write memory request command input for port A. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
$\overline{PEA}$	67	I	**PORT ENABLE FOR PORT A:** This pin serves to enable a RAM cycle request for port A. It is generally decoded from the port address.
PCTLA	68	I	**PORT CONTROL FOR PORT A:** This pin is sampled on the falling edge of RESET. It configures port A to accept command inputs or processor status inputs. If low after RESET, the 8207 is programmed to accept command or iAPX 286 status inputs or Multibus commands. If high after RESET, the 8207 is programmed to accept status inputs from iAPX 86 or iAPX 186 processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept commands or iAPX 286 status, it should be tied low or it may be connected to INHIBIT when operating with Multibus.

GENERAL DESCRIPTION

The Intel 8207 Advanced Dynamic RAM Controller (ADRC) is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 16K, 64K and 256K dynamic RAMs. This controller also provides the necessary arbitration circuitry to support dual-port access of the dynamic RAM array.

The ADRC supports several microprocessor interface options including synchronous and asynchronous connection to iAPX 86, iAPX 88, iAPX 186, iAPX 188, iAPX 286 and Multibus.

This device may be used with the 8206 Error Detection and Correction Unit (EDCU). When used with the 8206, the 8207 is programmed in the Error Checking and Correction (ECC) mode. In this mode, the 8207 provides all the necessary control signals for the 8206 to perform memory initialization and transparent error scrubbing during refresh.

FUNCTIONAL DESCRIPTION

Processor Interface

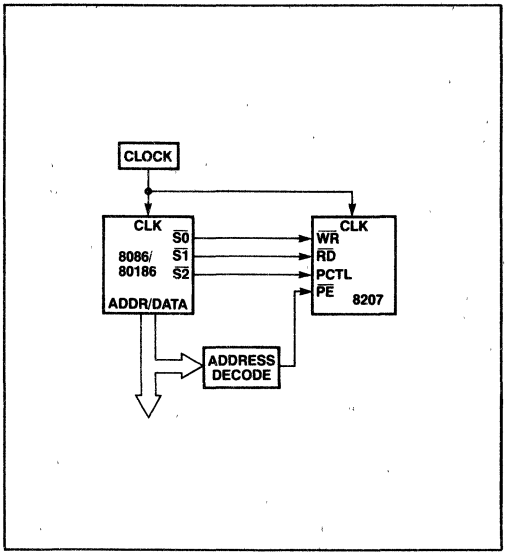
The 8207 has control circuitry for two ports each capable of supporting one of several possible bus structures. The ports are independently configurable allowing the dynamic RAM to serve as an interface between two different bus structures.

Each port of the 8207 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8207 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286. When the 8207 is programmed to run in asynchronous mode, the 8207 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

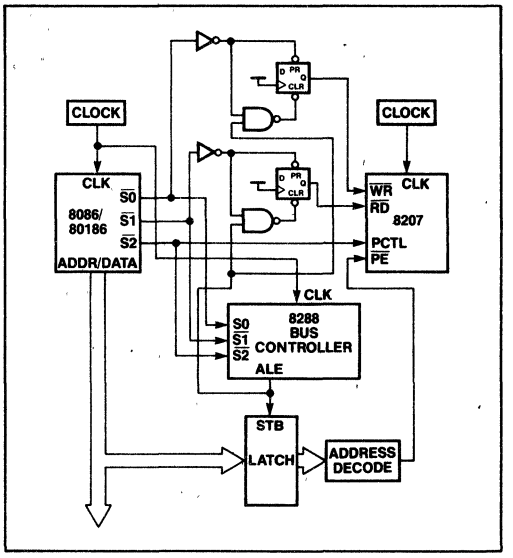
The 8207 achieves high performance (i.e. no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186, iAPX 188 and iAPX 286 processors. The 8207 can also be programmed to receive read or write Multibus commands or commands from a bus controller. (See Status/Command Mode)

The 8207 may be programmed to accept the clock of the iAPX 86, 88, 186, 188, or 286. The 8207 adjusts its internal timing to allow for the different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

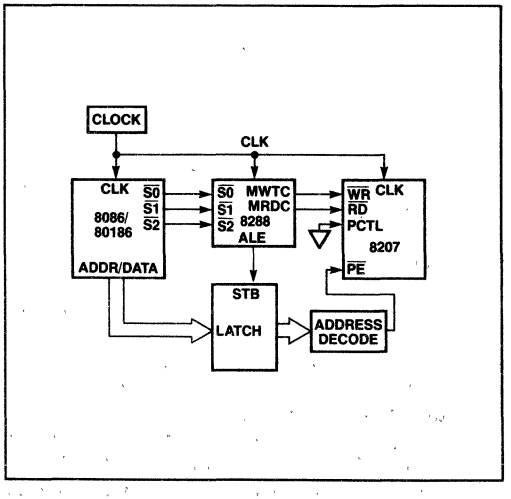
Figure 2 shows the different processor interfaces to the 8207 using the synchronous or asynchronous mode and status or command interface.



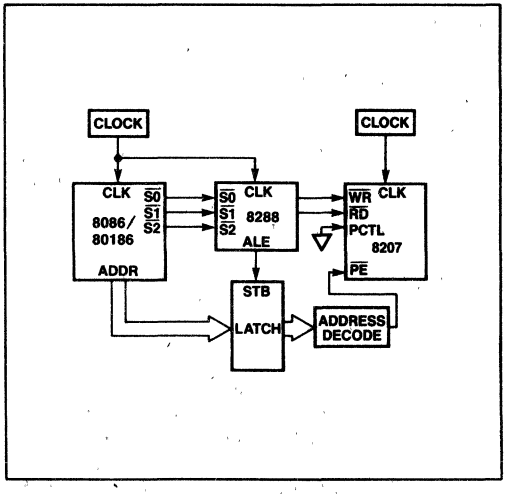
Slow-Cycle Synchronous-Status Interface



Slow-Cycle Asynchronous-Status Interface

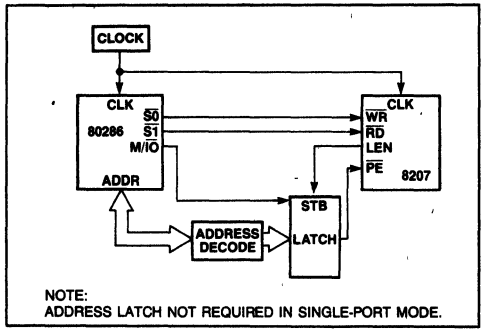


Slow-Cycle Synchronous-Command Interface

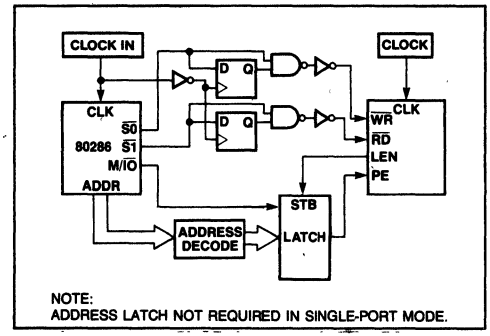


Slow-Cycle Asynchronous-Command Interface

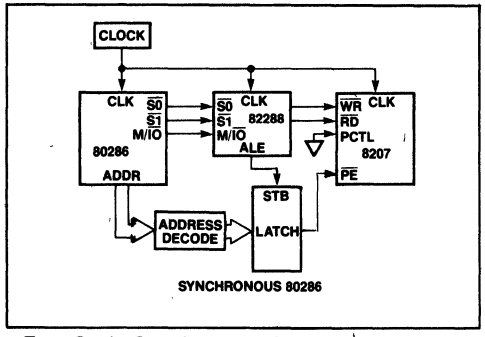
Figure 2A. Slow-cycle Port Interfaces Supported by the 8207



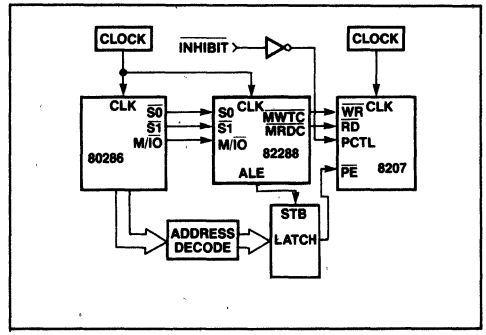
Fast-Cycle Synchronous-Status Interface



Fast-Cycle Asynchronous-Status Interface



Fast-Cycle Synchronous-Command Interface



Fast-Cycle Asynchronous-Command Interface

Figure 2B. Fast-cycle Port Interfaces Supported by the 8207

Single-Port Operation

The use of an address latch with the iAPX 286 status interface is not needed since the 8207 can internally latch the addresses with an internal signal similar in behavior to the LEN output. This operation is active only in single-port applications when the processor is interfaced to port A.

Dual-Port Operation

The 8207 provides for two-port operation. Two independent processors may access memory controlled by the 8207. The 8207 arbitrates between each of the processor requests and directs data to or from the appropriate port. Selection is done on a priority concept that reassigns priorities based upon past history. Processor requests are internally queued.

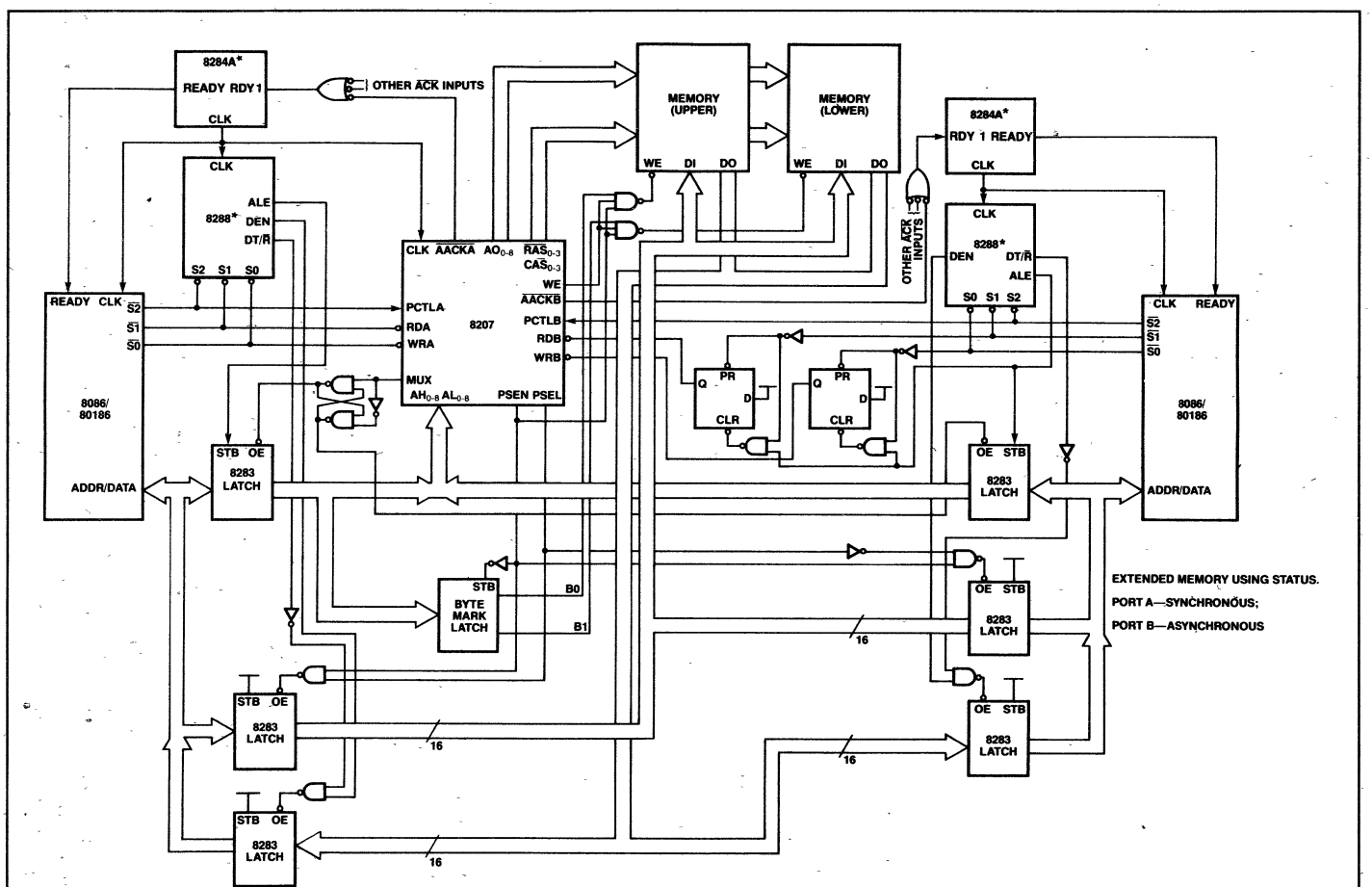
Figure 3 shows a dual-port configuration with two iAPX 86 systems interfacing to dynamic RAM. One of the processor systems is interfaced synchronously using the status interface and the other is interfaced asynchronously also using the status interface.

Dynamic RAM Interface

The 8207 is capable of addressing 16K, 64K and 256K dynamic RAMs. Figure 4 shows the connection of the processor address bus to the 8207 using the different RAMs. The 8207 directly supports the 2118 RAM family or any RAM with similar timing requirements and responses including the Intel 2164A RAM.

The 8207 divides memory into as many as four banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving and permits error scrubbing during ECC refresh cycles. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM Precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in alternate banks.

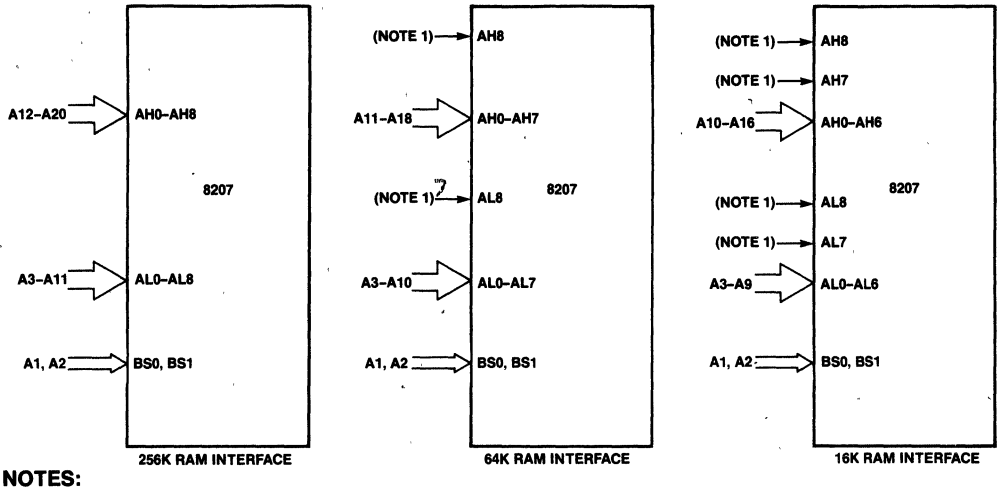
Successive data access to the same bank will cause the 8207 to wait for the precharge time of the previous RAM cycle.



NOTE:
*These components are not necessary when using the 80186 components. These functions are provided directly by the 80186.

Figure 3. 8086/80186 Dual Port System

210463-003



NOTES:
(1) Unassigned address input pins should be strapped high or low.
(2) A0 along with BHE are used to select a byte within a processor word.
(3) Low order address bits are used as bank select inputs so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 4. Processor Address Interface to the 8207 Using 16K, 64K, and 256K RAMS

If not all RAM banks are occupied, the 8207 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only two RAM banks are occupied, then two $\overline{RAS}$ and two $\overline{CAS}$ strobes are activated per bank. Program bits RB1 and RB0 are not used to check the bank select inputs BS1 and BS0. The system design must protect from accesses to "illegal", non-existent banks of memory, by deactivating the PEA, PEB inputs when addressing an illegal bank.

The 8207 can interface to fast (e.g., 2118-10) or slow (e.g., 2118-15) RAMs. The 8207 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option).

Memory Initialization

After programming, the 8207 performs eight RAM "warm-up" cycles to prepare the dynamic RAM for proper device operation. During "warm-up" some RAM parameters, such as tRAH, tASC, may not be met. This causes no harm to the dynamic RAM array. If configured for operation with error correction, the 8207 and 8206 EDCU will proceed to initialize all of memory (memory is written with zeros with corresponding check bits).

Table 2.
Bank Selection Decoding and Word Expansion

Program Bits		Bank Input		
RB1	RB0	BS1	BSØ	RAS/CAS Pair Allocation
0	0	0	0	$RAS_{0\text{-}3}$, $CAS_{0\text{-}3}$ to Bank 0
0	0	0	1	Illegal
0	0	1	0	Illegal
0	0	1	1	Illegal
0	1	0	0	$RAS_{0,1}$, $CAS_{0,1}$ to Bank 0
0	1	0	1	$RAS_{2,3}$, $CAS_{2,3}$ to Bank 1
0	1	1	0	Illegal
0	1	1	1	Illegal
1	0	0	0	RAS_0, CAS_0 to Bank 0
1	0	0	1	RAS_1, CAS_1 to Bank 1
1	0	1	0	RAS_2, CAS_2 to Bank 2
1	0	1	1	Illegal
1	1	0	0	RAS_0, CAS_0 to Bank 0
1	1	0	1	RAS_1, CAS_1 to Bank 1
1	1	1	0	RAS_2, CAS_2 to Bank 2
1	1	1	1	RAS_3, CAS_3 to Bank 3

Because the time to initialize memory is fairly long, the 8207 may be programmed to skip initialization in ECC mode. The time required to initialize all of memory is dependent on the clock cycle time to the 8207 and can be calculated by the following equation:

eq.1 $T_{INIT} = (2^{23})\ T_{CLCL}$

if T_{CLCL} = 125 ns then $T_{INIT} \approx 1$ sec.

8206 ECC Interface

For operation with Error Checking and Correction (ECC), the 8207 adjusts its internal timing and changes some pin functions to optimize performance and provide a clean dual-port memory interface between the 8206 EDCU and memory. The 8207 directly supports a master-only (16-bit word plus 6 check bits) system. Under extended operation and reduced clock frequency, the 8207 will support any ECC master-slave configuration up to 80 data bits, which is the maximum set by the 8206 EDCU. (See Extend Option)

Correctable errors detected during memory read cycles are corrected immediately and then written back into memory.

In a synchronous bus environment, ECC system performance has been optimized to enhance processor throughput, while in an asynchronous bus environment (the Multibus), ECC performance has been optimized to get valid data onto the bus as quickly as possible. Performance optimization, processor throughput or quick data access may be selected via the Transfer Acknowledge Option.

The main difference between the two ECC implementations is that, when optimized for processor throughput, RAM data is always corrected and an advanced transfer acknowledge is issued at a point when, by knowing the processor characteristics, data is guaranteed to be valid by the time the processor needs it.

When optimized for quick data access, (valid for Multibus) the 8206 is configured in the uncorrecting mode where the delay associated with error correction circuitry is transparent, and a transfer acknowledge is issued as soon as valid data is known to exist. If the $\overline{ERROR}$ flag is activated, then the transfer acknowledge is delayed until after the 8207 has instructed the 8206 to correct the data and the corrected data becomes available on the bus. Figure 5 illustrates a dual-port ECC system.

Figure 6 illustrates the interface required to drive the $\overline{CRCT}$ pin of the 8206, in the case that one port (PORT A) receives an advanced acknowledge (not Multibus-compatible), while the other port (PORT B) receives $\overline{XACK}$ (which is Multibus-compatible).

Error Scrubbing

The 8207/8206 performs error correction during refresh cycles (error scrubbing). Since the 8207 must refresh RAM, performing error scrubbing during refresh allows it to be accomplished without additional performance penalties.

Upon detection of a correctable error during refresh, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored.

Refresh

The 8207 provides an internal refresh interval counter and a refresh address counter to allow the 8207 to refresh memory. The 8207 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8207 may be programmed for any of four different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh mode, or no refresh. (See Refresh Options)

It is possible to decrease the refresh time interval by 10%, 20% or 30%. This option allows the 8207 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8207 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper

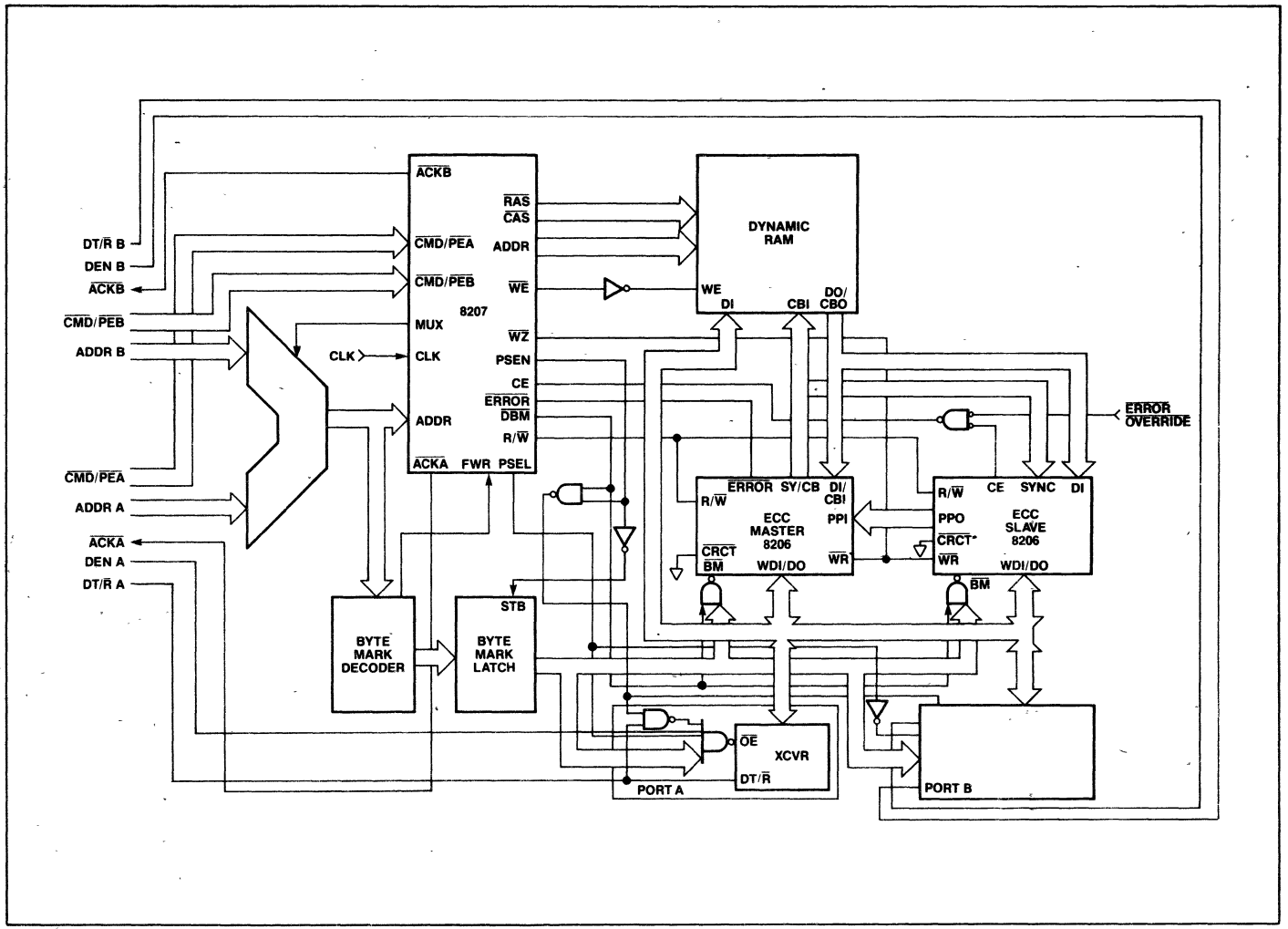


Figure 5. Two-Port ECC Implementation Using the 8207 and the 8206

210463-003

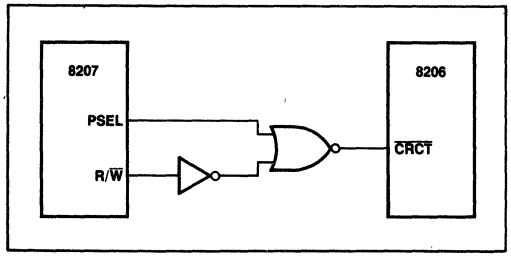


Figure 6. Interface to 8206 $\overline{\text{CRCT}}$ Input When Port A Receives $\overline{\text{AACK}}$ and Port B Receives $\overline{\text{XACK}}$

dynamic RAM operation, and memory initialization if error correction is used. Many dynamic RAMs require this warm-up period for proper operation. The time it takes for the 8207 to recognize a request is shown below.

eq. 2 Non-ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP}$

eq. 3 where: T_{PROG} = (66) (T_{CLCL}) which is programming time

eq. 4 T_{PREP} = (8) (32) (T_{CLCL}) which is the RAM warm-up time

if T_{CLCL} = 125 ns then $T_{RESP} \approx$ 41 us

eq. 5 ECC Systems: $T_{RESP} = T_{PROG} + T_{PREP} + T_{INIT}$

if T_{CLCL} = 125 ns then $T_{RESP} \approx$ 1 sec

RESET

RESET is an asynchronous input, the falling edge of which is used by the 8207 to directly sample to logic levels of the PCTLA, PCTLB, RFRQ, and PDI inputs. The internally synchronized falling edge of RESET is used to begin programming operations (shifting in the contents of the external shift register into the PDI input).

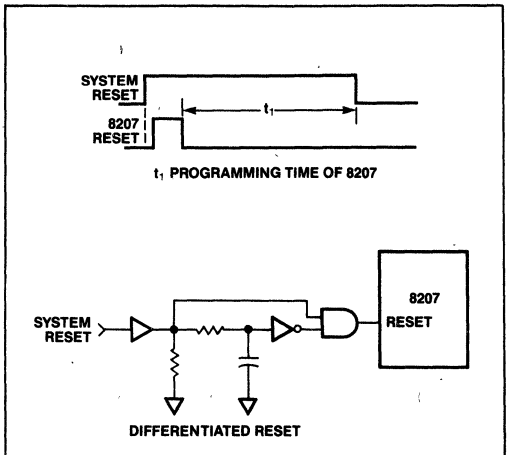
Until programming is complete the 8207 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8207. The total time of the reset pulse and the 8207 programming time must be less than the time before the first command in systems that alter the default port synchronization programming bits (default is Port A synchronous, Port B asynchronous). Differentiated reset is unnecessary when the default port synchronization programming is used. The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8207. The differentiated reset pulse first resets the 8207, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8207 completes its programming. Figure 7 illustrates a circuit to accomplish this task.

Within four clocks after RESET goes active, all the 8207 outputs will go high, except for PSEN, WE, and AO0-2, which will go low.

OPERATIONAL DESCRIPTION

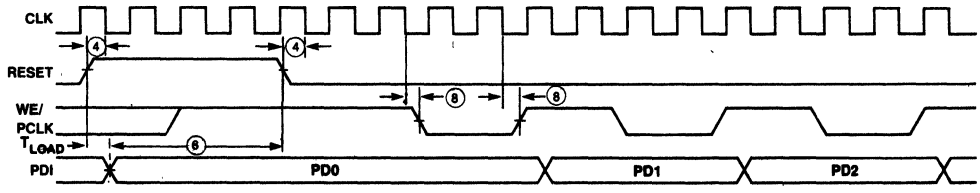
Programming the 8207

The 8207 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTLA, PCTLB, REFRQ, and PDI pins. Figure 8 shows the necessary timing for programming the 8207.



NOTES:
(1) Required only when the port synchronization options (SA & SB) are altered from their initial default values.
(2) V_{CC} must be stable before system reset is activated when using this circuit.

Figure 7. 8207 Differentiated Reset Circuit



NOTES:
TRTVCL — Reset is an asynchronous input, if reset occurs before T_1, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asynchronous load data propagation delay.

Figure 8. Timing Illustrating External Shift Register Requirements for Programming the 8207

Status/Command Mode

The two processor ports of the 8207 are configured by the states of the PCTLA and PCTLB pins. Which interface is selected depends on the state of the individual port's PCTL pin at the end of reset. If PCTL is high at the end of the reset, the 8086 Status interface is selected; if it is low, then the Command interface is selected.

The status lines of the 80286 are similar in code and timing to the Multibus command lines, while the status code and timing of the 8076 and 8088 are identical to those of the 80186 and 80188 (ignoring the differences in clock duty cycle). Thus there exists two interface configurations, one for the 80286 status or Multibus memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 8086 Status interface. The Command interface can also directly interface to the command lines of the bus controllers for the 8086, 8088, 80186 and the 80286.

The 8086 Status interface allows direct decoding of the status of the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. While in the Command mode the iAPX 286 status can be directly decoded. Microprocessor bus controller read or write commands or Multibus commands can also be directed to the 8207 when in Command mode.

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8207 provides the user with the choice between self-refresh or user-generated refresh with failsafe protection. Failsafe protection guarantees that if the

Table 3A. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	INTERRUPT	INTERRUPT
0	0	1	I/O READ	I/O READ
0	1	0	I/O WRITE	I/O WRITE
0	1	1	HALT	IDLE
1	0	0	INSTRUCTION FETCH	HALT
1	0	1	MEMORY READ	MEMORY READ
1	1	0	MEMORY WRITE	MEMORY WRITE
1	1	1	IDLE	IDLE

Table 3B. 8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086/80186 Status Interface	80286/Status or Command Interface
0	0	0	IGNORE	IGNORE
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8207 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8207. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period causes a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods causes a burst of up to 128 row address locations to be refreshed.

In ECC-configured systems, 128 locations are scrubbed. Any refresh request is not recognized until a previous request has been serviced (i.e., burst completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 16 program data bits, PD0—PD15. If the first program data bit PD0 is set to logic 1, the 8207 is configured to support ECC. If it is logic 0, the 8207 is configured to support a non-ECC system. The remaining bits, PD1—PD15, may then be programmed to optimize a selected configuration. Figures 9 and 10 show the Program words for non-ECC and ECC operation.

Using an External Shift Register

The 8207 may be configured to use an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8207 supplies the clocking signal to shift the data in. Figure 11 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8207 via the PDI pin (57), and clock is provided by the MUX/PCLK pin (12), which generates a total of 16 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. MUX/PCLK is a dual-function pin. During programming, it serves to clock the external shift register, and after programming is completed, it reverts to a MUX control pin. As the pin changes state to select different port addresses, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 8 illustrates the timing requirements of the shift register circuitry.

ECC Mode (ECC Program Bit)

The state of PDI (Program Data In) pin at reset determines whether the system is an ECC or non-ECC configuration. It is used internally by the 8207 to begin configuring timing circuits, even before programming is completely finished. The 8207 then begins programming the rest of the options.

Default Programming Options

After reset, the 8207 serially shifts in a program data word via the PDI pin. This pin may be strapped either high or low, or connected to an external shift register. Strapping PDI high causes the 8207 to default to a particular system configuration with error correction, and strapping it low causes the 8207 to default to a particular system configuration without error correction. Table 4 shows the default configurations.

PD15								PD8 PD7							PD0
0	0	TM 1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=0 FOR NON-ECC MODE
PD1	$\overline{SA}$	$\overline{SA}$=0 PORT A IS SYNCHRONOUS $\overline{SA}$=1 PORT A IS ASYNCHRONOUS
PD2	SB	SB=0 PORT B IS ASYNCHRONOUS SB=1 PORT B IS SYNCHRONOUS
PD3	$\overline{CFS}$	$\overline{CFS}$=0 FAST-CYCLE iAPX 286 MODE $\overline{CFS}$=1 SLOW-CYCLE iAPX 86 MODE
PD4	$\overline{RFS}$	$\overline{RFS}$=0 FAST RAM $\overline{RFS}$=1 SLOW RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM BANK OCCUPANCY SEE TABLE 2
PD7 PD8	CI1 CI0	COUNT INTERVAL BIT 1; SEE TABLE 6 COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	$\overline{PLS}$	$\overline{PLS}$=0 LONG REFRESH PERIOD $\overline{PLS}$=1 SHORT REFRESH PERIOD
PD10	EXT	EXT=0 NOT EXTENDED EXT=1 EXTENDED
PD11	$\overline{FFS}$	$\overline{FFS}$=0 FAST CPU FREQUENCY $\overline{FFS}$=1 SLOW CPU FREQUENCY
PD12	$\overline{PPR}$	$\overline{PPR}$=0 MOST RECENTLY USED PORT PRIORITY $\overline{PPR}$=1 PORT A PREFERRED PRIORITY
PD13	TM1	TM1=0 TEST MODE 1 OFF TM1=1 TEST MODE 1 ENABLED
PD14	0	RESERVED MUST BE ZERO
PD15	0	RESERVED MUST BE ZERO

Figure 9. Non-ECC Mode Program Data Word

PD15								PD8 PD7							PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	ECC	ECC=1 ECC MODE
PD1	SA	SA=0 PORT A ASYNCHRONOUS SA=1 PORT A SYNCHRONOUS
PD2	$\overline{SB}$	$\overline{SB}$=0 PORT B SYNCHRONOUS $\overline{SB}$=1 PORT B ASYNCHRONOUS
PD3	CFS	CFS=0 SLOW-CYCLE iAPX 86 MODE CFS=1 FAST-CYCLE iAPX 286 MODE
PD4	RFS	RFS=0 SLOW RAM RFS=1 FAST RAM
PD5	$\overline{XA}$	$\overline{XA}$=0 MULTIBUS-COMPATIBLE ACKA $\overline{XA}$=1 ADVANCED ACKA NOT MULTIBUS-COMPATIBLE
PD6	XB	XB=0 ADVANCED ACKB NOT MULTIBUS COMPATIBLE XB=1 MULTIBUS-COMPATIBLE ACKB
PD7	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD8	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD9	PLS	PLS=0 SHORT REFRESH PERIOD PLS=1 LONG REFRESH PERIOD
PD10	$\overline{EXT}$	$\overline{EXT}$=0 MASTER AND SLAVE EDCU $\overline{EXT}$=1 MASTER EDCU ONLY
PD11	FFS	FFS=0 SLOW CPU FREQUENCY FFS=1 FAST CPU FREQUENCY
PD12	PPR	PPR=0 PORT A PREFERRED PRIORITY PPR=1 MOST RECENTLY USED PORT PRIORITY
PD13 PD14	RB0 RB1	RAM BANK OCCUPANCY SEE TABLE 2
PD15	$\overline{TM2}$	$\overline{TM2}$=0 TEST MODE 2 ENABLED $\overline{TM2}$=1 TEST MODE 2 OFF

Figure 10. ECC Mode Program Data Word

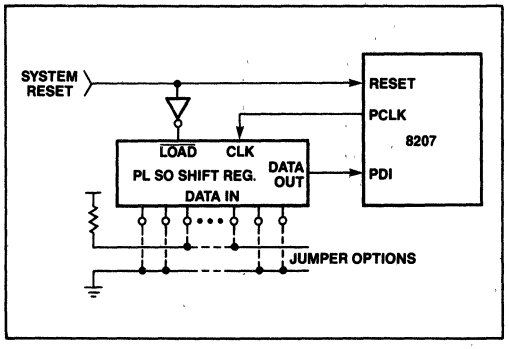


Figure 11. External Shift Register Interface

Table 4A.
Default Non-ECC Programming, PDI Pin (57) Tied to Ground.

Port A is Synchronous ($\overline{EAACKA}$ and $\overline{XACKA}$)
Port B is Asynchronous ($\overline{LAACKB}$ and $\overline{XACKB}$)
Fast-cycle Processor Interface (10 or 16 MHz)
Fast RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Table 4B.
Default ECC Programming, PDI Pin (57) Tied to V_{CC}.

Port A is Synchronous
Port B is Asynchronous
Fast-cycle Processor Interface (10 or 16 MHz)
Fast RAM
Port A has $\overline{EAACKA}$ strobe (non-multibus)
Port B has $\overline{XACKB}$ strobe (multibus)
Refresh interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Master EDCU only (16-bit system)
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

If further system flexibility is needed, one or two external shift registers can be used to tailor the 8207 to its operating environment.

Synchronous/Asynchronous Mode (SA and SB Program Bits)

Each port of the 8207 may be independently configured to accept synchronous or asynchronous port commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the program bits SA and SB. The state of the SA and SB programming bits determine whether their associated ports are synchronous or asynchronous.

While a port may be configured with either the Status or Command interface in the synchronous mode, certain restrictions exist in the asynchronous mode. An asynchronous Command interface using the control lines of the Multibus is supported, and an asynchronous 8086 interface using the control lines of the 8086 is supported, with the use of TTL gates as illustrated in Figure 2. In the 8086 case, the TTL gates are needed to guarantee that status does not appear at the 8207's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Frequency Option (CFS and FFS Program Bits)

The 8207 can be programmed to interface with slow-cycle microprocessors like the 8086, 8088, 80188 and 80186 or fast-cycle microprocessors like the 80286. The CFS bit configures the microprocessor interface to accept slow or fast cycle signals from either microprocessor group.

This option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed.

Table 5.
Microprocessor Clock Frequency Options

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86, 88, 186, 188	5 MHz
0	1	iAPX 86, 88, 186, 188	8 MHz
1	0	iAPX 286	10 MHz
1	1	iAPX 286	16 MHz

The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

RAM Speed Option (RFS Program Bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to the 2118-10 (Fast) or the 2118-15 (Slow) RAM specifications.

Refresh Period Options (CI0, CI1, and PLS Program Bits)

The 8207 refreshes with either 128 rows every 2 milliseconds or 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option. The 7.8 microsecond refresh request rate is intended for those RAMs, 64K and above, which may require a faster refresh rate.

In addition to PLS program option, two other programming bits for refresh exist: Count Interval 0 (CI0) and Count Interval 1 (CI1). These two programming bits allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the same 15.6 or 7.8 microsecond period when the 8207 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Extend Option (EXT Program Bit)

The Extend option lengthens the memory cycle to allow longer access time which may be required by the system. Extend alters the RAM timing to compensate for increased loading on the Row and Column Address Strobes, and in the multiplexed Address Out lines.

Port Priority Option and Arbitration (PPR Program Bit)

The 8207 has to internally arbitrate among three ports: Port A, Port B and Port C—the refresh port. Port C is an internal port dedicated to servicing refresh requests, whether they are generated internally by the refresh inverval counter, or externally by the user. Two arbitration approaches are available via

Table 6. Refresh Count Interval Table

Freq. (MHz)	Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (8207 Clock Periods)			
					00 (0%)	01 (10%)	10 (20%)	11 (30%)
16	15.6	1	1	1	236	212	188	164
	7.8	1	0	1	118	106	94	82
10	15.6	1	1	0	148	132	116	100
	7.8	1	0	0	74	66	58	50
8	15.6	0	1	1	118	106	94	82
	7.8	0	0	1	59	53	47	41
5	15.6	0	1	0	74	66	58	50
	7.8	0	0	0	37	33	29	25

the Port Priority programming option, program bit PPR. PPR determines whether the most recently used port will remain selected (PPR = 1) or whether Port A will be favored or preferred over Port B (PPR = 0).

A port is selected if the arbiter has given the selected port direct access to the timing generators. The front-end logic, which includes the arbiter, is designed to operate in parallel with the selected port. Thus a request on the selected port is serviced immediately. In contrast, an unselected port only has access to the timing generators through the front-end logic. Before a RAM cycle can start for an unselected port, that port must first become selected (i.e., the MUX output now gates that port's address into the 8207 in the case of Port A or B). Also, in order to allow its address to stabilize, a newly selected port's first RAM cycle is started by the front-end logic. Therefore, the selected port has direct access to the timing generators. What all this means is that a request on a selected port is started immediately, while a request on an unselected port is started two to three clock periods after the request, assuming that the other two ports are idle. Under normal operating conditions, this arbitration time is hidden behind the RAM cycle of the selected port so that as soon as the present cycle is over a new cycle is started. Table 7 lists the arbitration rules for both options.

Port LOCK Function

The LOCK function provides each port with the ability to obtain uninterrupted access to a critical region of memory and, thereby, to guarantee that the opposite port cannot "sneak in" and read from or write to the critical region prematurely.

Only one LOCK pin is present and is multiplexed between the two ports as follows: when MUX is high, the 8207 treats the LOCK input as originating at PORT A, while when MUX is low, the 8207 treats LOCK as originating at PORT B. When the 8207 recognizes a LOCK, the MUX output will remain pointed to the locking port until LOCK is deactivated. Refresh is not affected by LOCK and can occur during a locked memory cycle.

Table 7. The Arbitration Rules for the Most Recently Used Port Priority and for Port A Priority Options Are As follows:

1.	If only one port requests service, then that port—if not already selected—becomes selected.
2a.	When no service requests are pending, the last selected processor port (Port A or B) will remain selected. (Most Recently Used Port Priority Option)
2b.	When no service requests are pending, Port A is selected whether it requests service or not. (Port A Priority Option)
3.	During reset initialization only Port C, the refresh port, is selected.
4.	If no processor requests are pending after reset initialization, Port A will be selected.
5a.	If Ports A and B simultaneously(*) request service while Port C is being serviced, then the next port to be selected is the one which was not selected prior to servicing Port C. (Most Recently Used Port Priority Option)
5b.	If Ports A and B simultaneously(*) request service while Port C is selected, then the next port to be selected is Port A. (Port A Priority Option)
6.	If a port simultaneously requests service with the currently selected port, service is granted to the selected port.
7.	The MUX output remains in its last state whenever Port C is selected.
8.	If Port C and either Port A or Port B (or both) simultaneously request service, then service is granted to the requester whose port is already selected. If the selected port is not requesting service, then service is granted to Port C.
9.	If during the servicing of one port, the other port requests service before or simultaneously with the refresh port, the refresh port is selected. A new port is not selected before the presently selected port is deactivated.
10.	Activating LOCK will mask off service requests from Port B if the MUX output is high, or from Port A if the MUX output is low.
* By "simultaneous" it is meant that two or more requests are valid at the clock edge at which the internal arbiter samples them.	

Dual-Port Considerations

For both ports to be operated synchronously, several conditions must be met. The processors must be the same type (Fast or Slow Cycle) as defined by Table 8 and they must have synchronized clocks. Also when processor types are mixed, even though the clocks may be in phase, one frequency may be twice that of the other. So to run both ports synchronous using the status interface, the processors must have related timings (both phase and frequency). If these conditions cannot be met, then one port must run synchronous and the other asynchronous.

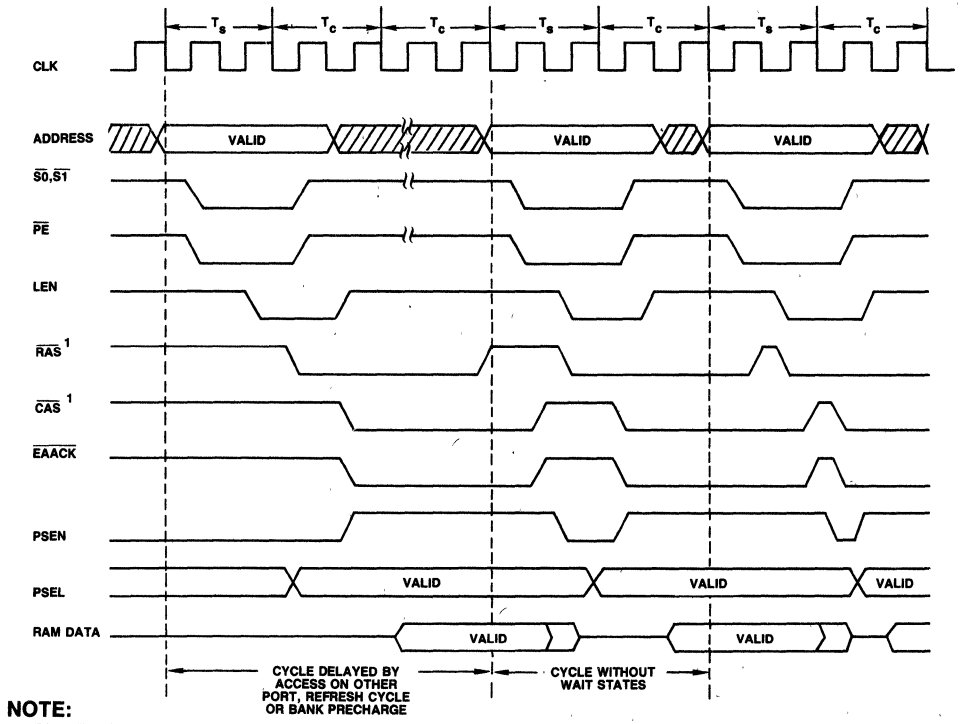
Figure 3 illustrates an example of dual-port operation using the processors in the slow cycle group. Note the use of cross-coupled NAND gates at the MUX output for minimizing contention between the two latches, and the use of flip flops on the status lines of the asynchronous processor for delaying the status and thereby guaranteeing RAS will not be issued, even in the worst case, until address is valid.

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processor's request The timing is such, however, that the processor will run without wait states, barring refresh cycles, bank precharge, and RAM accesses from the other port. In non-ECC fast cycle, fast RAM, non-extended configurations (80286), $\overline{AACK}$ is issued on the next falling edge of the clock after the edge that issues RAS. In non-ECC, slow cycle, non-extended, or extended with fast RAM cycle configurations (8086, 80188, 80186), $\overline{AACK}$ is issued on the same clock cycle that issues RAS. Figure 14 illustrates the timing relationship between $\overline{AACK}$, the RAM cycle, and the processor cycle for several different situations.

Port Enable ($\overline{PE}$ setup time requirements depend on whether the associated port is configured for synchronous or asynchronous fast or slow cycle operation. In a synchronous fast cycle configuration, PE is required to be setup to the same clock edge as the status or commands. If PE is true (low), a RAM cycle is started; if not, the cycle is aborted. The memory cycle will only begin when both valid signals ($\overline{PE}$ and $\overline{RD}$ or WR) are recognized at a particular clock edge. In aynchronous operation, PE is required to be setup to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command)-to-$\overline{PE}$ delay time, thus allowing for more external decode time that is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 8207 aborts the cycle. If a memory cycle intended for the 8207 is aborted, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states. Figure 15 illustrates the status (command) timing requirements for synchronous and asynchronous systems. Figures 16 and 17 show a more detailed hook-up of the 8207 to the 8086 and the 80286, respectively.



NOTE:
1. The RAS and CAS shown in figure are different banks being accessed.

Figure 14. iAPX 286/8207 Synchronous-Status Timing Programmed in non-ECC Mode, C0 Configuration (Read Cycle)

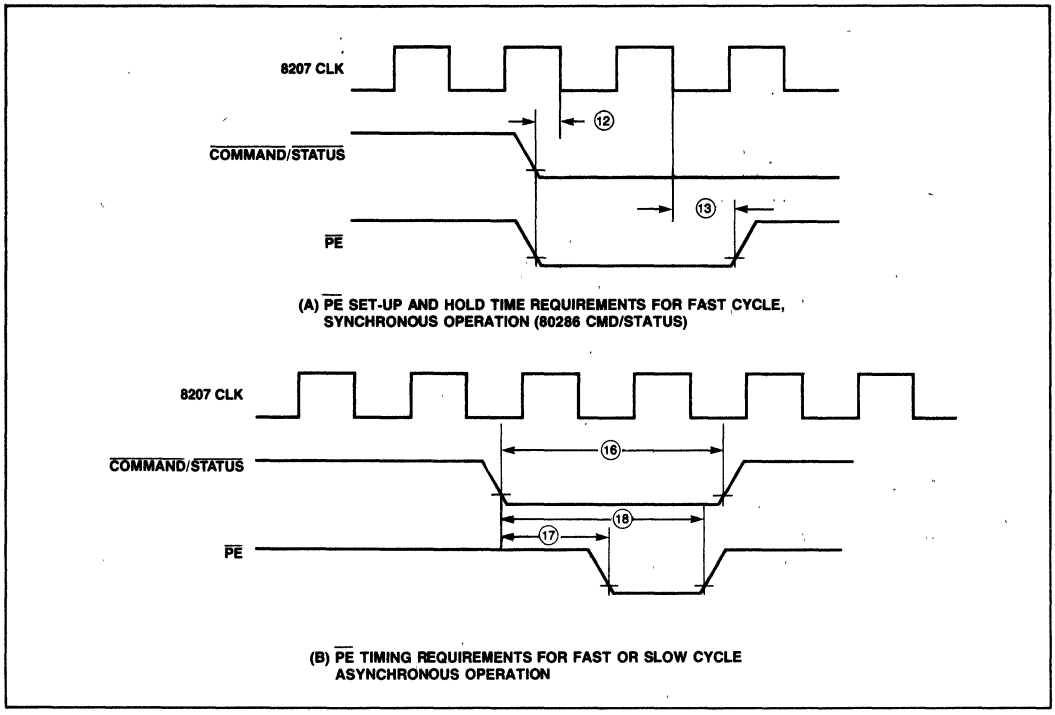


Figure 15.

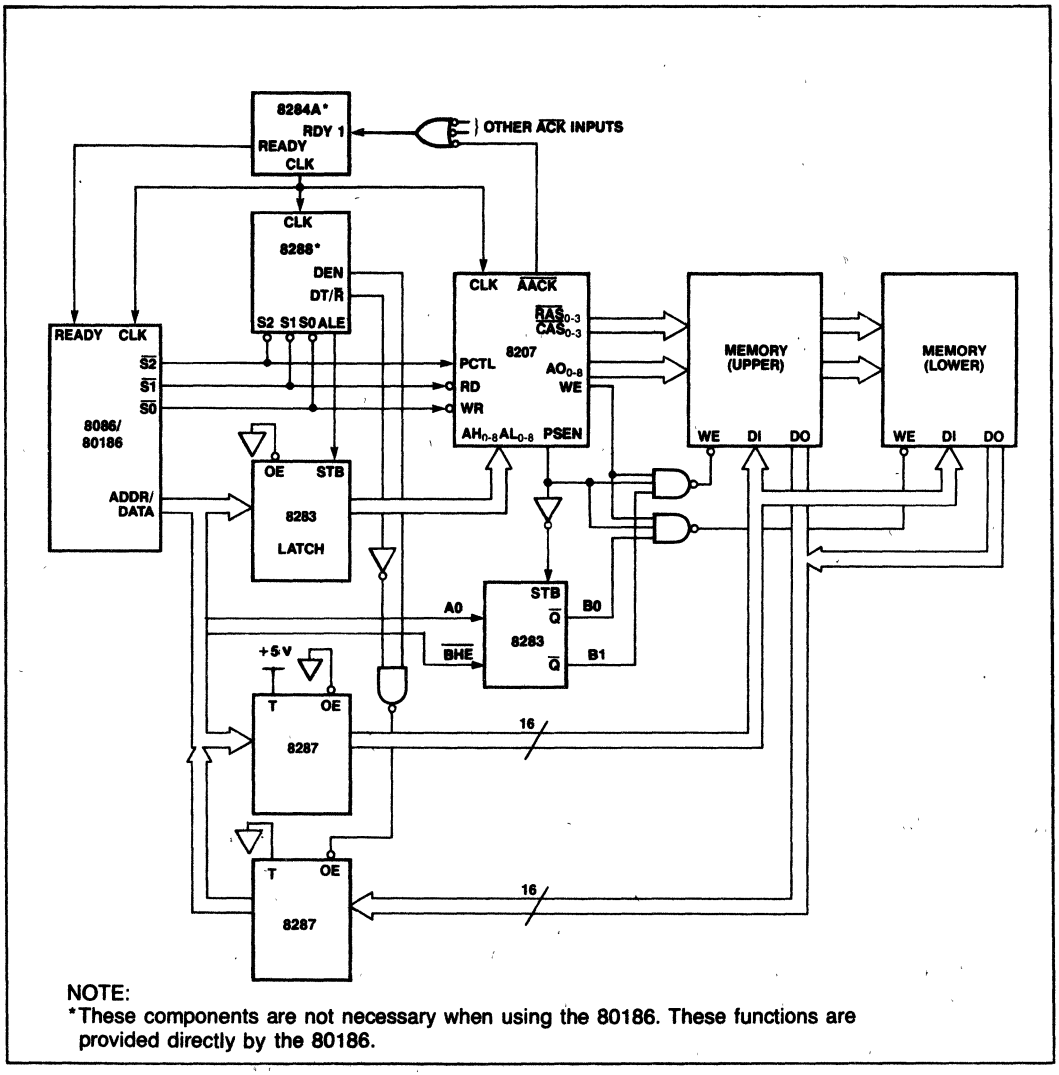
Memory Acknowledge (AACK, XACK)

In system configurations without error correction, two memory acknowledge signals per port are supplied by the 8207. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The CFS programming bit determines for which processor $\overline{AACKA}$ and $\overline{AACKB}$ are optimized, either 80286 (CFS = 1) or 8086/186 (CFS = 0), while the SA and SB programming bits optimize $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$).

Both the early and late $\overline{AACK}$ strobes are three clocks long for CFS = 1 and two clocks long for CFS = 0. The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the Multibus requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive. Since in asynchronous operation the 8207 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation, data latching is unnecessary since the 8207 will not remove data until the CPU has read it.

In ECC-based systems there is one memory acknowledge ($\overline{XACK}$ or $\overline{AACK}$) per port and a programming bit associated with each acknowledge. If the X programming bit is high, the strobe is configured as $\overline{XACK}$, while if the bit is low, the strobe is configured as $\overline{AACK}$. As in non-ECC, the SA and SB programming bits determine whether the $\overline{AACK}$ strobe is early or late ($\overline{EAACK}$ or $\overline{LAACK}$).

Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Table 9 summarizes the various transfer acknowledge options.



NOTE:
*These components are not necessary when using the 80186. These functions are provided directly by the 80186.

Figure 16. 8086/80186, 8207 Single Port Non-ECC Synchronous Systems

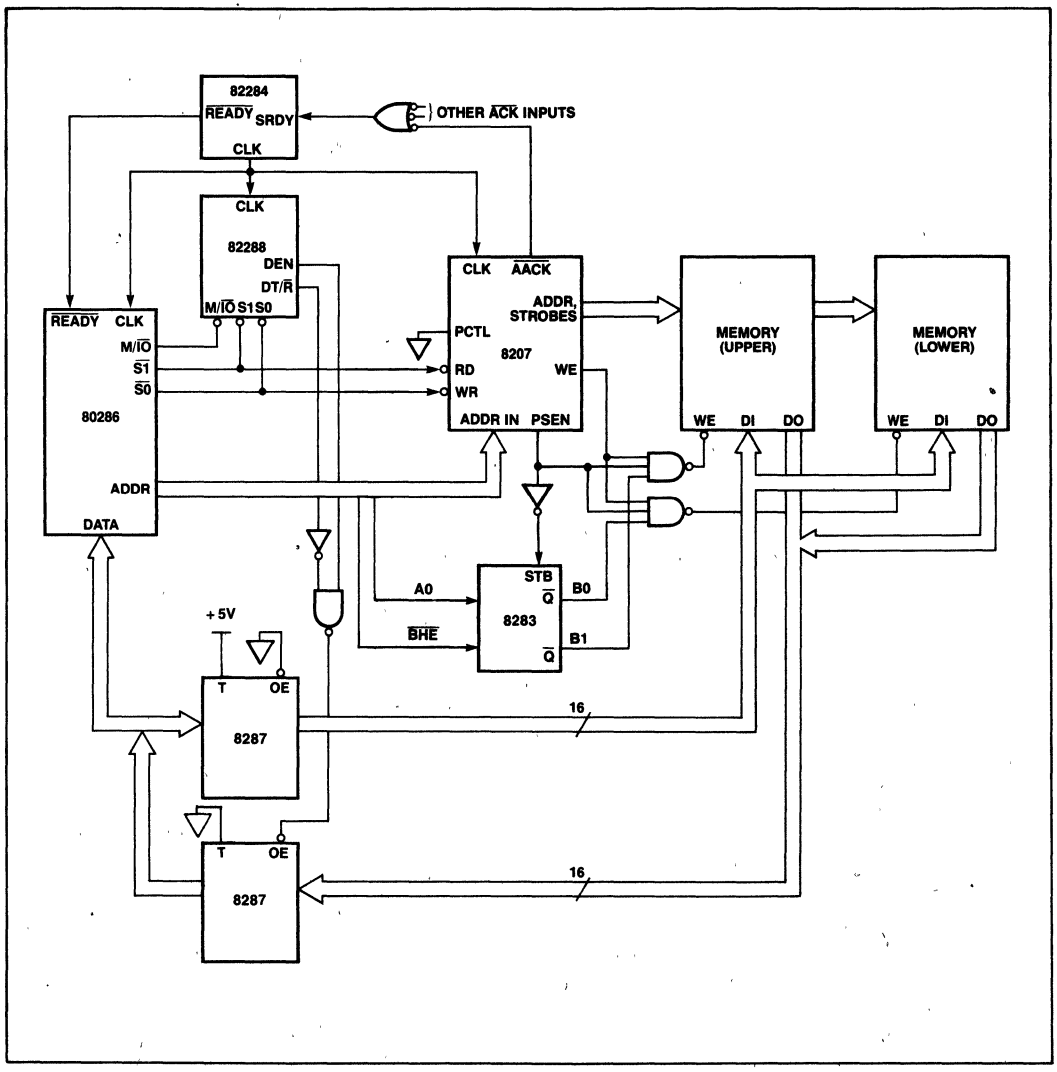


Figure 17. 80286 Hook-up to 8207 Non-ECC Synchronous System–Single Port.

Table 8. Processor Interface/Acknowledge Summary

CYCLE	PROCESSOR	REQUEST TYPE	SYNC/ASYNC INTERFACE	ACKNOWLEDGE TYPE
FAST CYCLE CFS=1	80286	STATUS	SYNC	EAACK
	80286	STATUS	ASYNC	LAACK
	80286	COMMAND	SYNC	EAACK
	80286	COMMAND	ASYNC	LAACK
	8086/80186	STATUS	ASYNC	LAACK
	8086/80186	COMMAND	ASYNC	LAACK
	MULTIBUS	COMMAND	ASYNC	XACK
SLOW CYCLE CFS=0	8086/80186	STATUS	SYNC	EAACK
	8086/80186	STATUS	ASYNC	LAACK
	8086/80186	COMMAND	SYNC	EAACK
	8086/80186	COMMAND	ASYNC	LAACK
	MULTIBUS	COMMAND	ASYNC	XACK

Table 9. Memory Acknowledge Option Summary

	Synchronous	Asynchronous	XACK
Fast Cycle	AACK Optimized for Local 80286	AACK Optimized for Remote 80286	Multibus Compatible
Slow Cycle	AACK Optimized for Local 8086/186	AACK Optimized for Remote 8086/186	Multibus Compatible

Test Modes

Two special test modes exist in the 8207 to facilitate testing. Test Mode 1 (non-ECC mode) splits the refresh address counter into two separate counters and Test Mode 2 (ECC mode) presets the refresh address counter to a value slightly less than rollover.

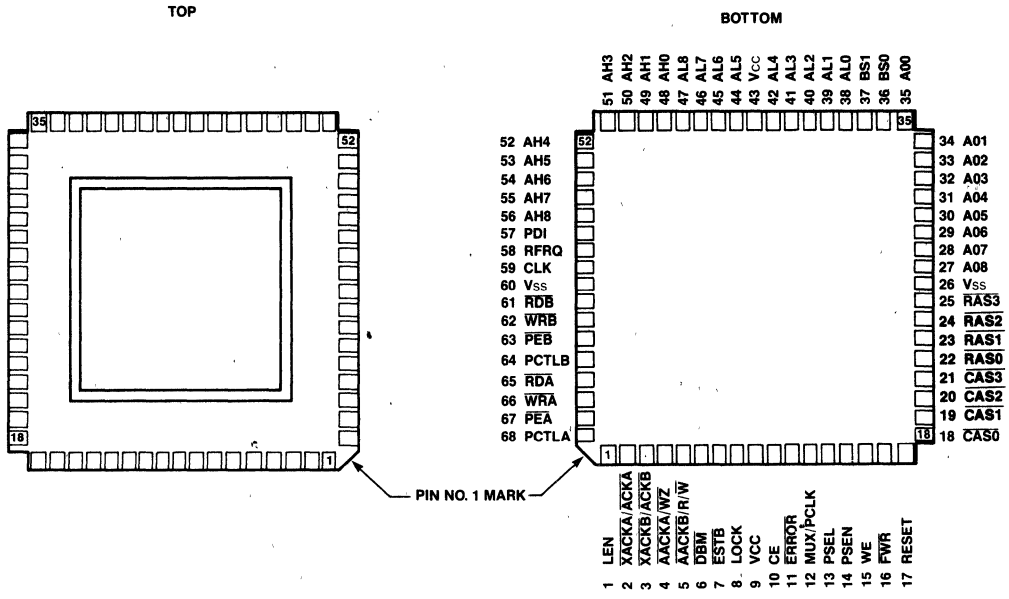
Test Mode 1 splits the address counter into two, and increments both counters simultaneously with each refresh address update. By generating external refresh requests, the tester is able to check for proper operation of both counters. Once proper individual counter operation has been established, the 8207 must be returned to normal mode and a second test performed to check that the carry from the first counter increments the second counter. The outputs of the counters are presented on the address out bus with the same timing as the row and column addresses of a normal scrubbing operation. During Test Mode 1, memory initialization is inhibited, since the 8207, by definition, is in non-ECC mode.

Test Mode 2 sets the internal refresh counter to a value slightly less than rollover. During functional testing other than that covered in Test Mode 1, the 8207 will normally be set in Test Mode 2. Test Mode 2 eliminates memory initialization in ECC mode. This allows quick examination of the circuitry which brings the 8207 out of memory initialization and into normal operation.

General System Considerations

The RAS_{0-3}, CAS_{0-3}, AO_{0-8}, output buffers were designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment and causing noise in other output pins it is necessary to match the output impedance of the RAM output buffers with the RAM array by using series resistors and to add series resistors to other control outputs for noise reduction if necessary. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application. In non-ECC systems unused ECC input pins should be tied high or low to improve noise immunity.

The 8207 is packaged in a 68-pin, leadless JEDEC type A hermetic chip carrier.



NOTE:
8207 is packaged in a 68 pin JEDEC Type A hermetic leadless chip carrier.

Figure 19. 8207 Pinout Diagram

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature
Under Bias .....−0° C to +70° C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin With
Respect to Ground ..... −.5V to +7V
Power Dissipation ..... 2.5 Watts

NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS

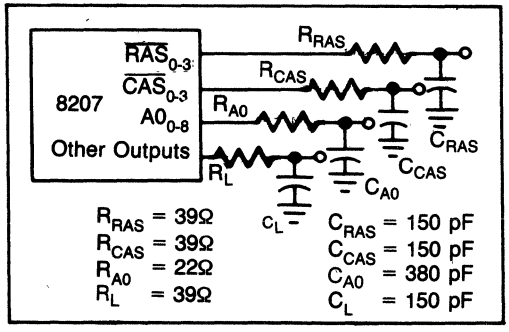
(T_A = 0°C to +70°C, V_{CC} = 5.0V ± 10% for 8207; ± 5% for 8207-2 and 8207-5, V_{SS} = GND)

Symbol	Parameter	Min.	Max.	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 1
V_{ROL}	RAM Output Low Voltage		0.45	V	Note 1
V_{ROH}	RAM Output High Voltage	2.6		V	Note 1
I_{CC}	Supply Current		400	mA	T_A = 25°C
I_{LI}	Input Leakage Current		+10	μA	0V ≤ V_{IN} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz

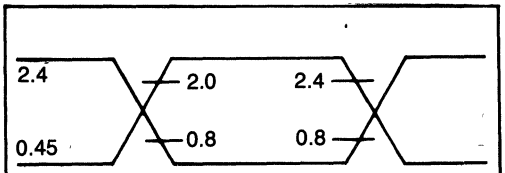
NOTES:

I_{OL} = 8 mA and I_{OH} = -0.2 mA (Typically I_{OL} = 10 mA and I_{OH} = -0.88 mA)

A.C. Testing Load Circuit



A.C. Testing Input, Output Waveform



A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8 V for logic "0".

210463-003

A.C. CHARACTERISTICS

(T_A = 0°C to 70°C; V_{CC} = +5V ± 10% for 8207; ± 5% for 8207-2, 8207-5; V_{SS} = 0V)
Measurements made with respect to RAS_{0-3}, CAS_{0-3}, AO_{0-8}, are at 2.4V and 0.8V. All other pins are measured at 2.0V and 0.8V. All times in nsec unless otherwise indicated. Testing done with specified test load.

CLOCK AND PROGRAMMING

Ref.	Symbol	Parameter		8207 & 8207-2 Min.	8207 & 8207-2 Max.	8207-5 Min.	8207-5 Max.	Units	Notes
—	tF	Clock Fall Time			10		10	ns	35
—	tR	Clock Rise Time			10		10	ns	35
1	TCLCL	Clock Period	8207	62.5	250			ns	1
			8207	125	500			ns	2
			8207-2	125	500	200	500	ns	3
2	TCL	Clock Low Time	8207	15	230			ns	1
			8207	TCLCL/2-12				ns	2
			8207-2	TCLCL/2-12		TCLCL/2-12		ns	3
3	TCH	Clock High Time	8207	20	235			ns	1
			8207	TCLCL/3-3				ns	2
			8207-2	TCLCL/3-3		TCLCL/3-3		ns	3
4	TRTVCL	Reset to CLK↓ Setup		40		65		ns	4
5	TRTH	Reset Pulse Width		4 TCLCL		4 TCLCL		ns	
6	TPGVRTL	PCTL, PDI, RFRQ to RESET↓ Setup		125		200		ns	5
7	TRTLPGX	PCTL, RFRQ to RESET↓ Hold		10		10		ns	
8	TCLPC	PCLK from CLK↓ Delay			45		65	ns	
9	TPDVCL	PDIn to CLK↓ Setup		60		100		ns	
10	TCLPDX	PDIn to CLK↓ Hold		40		65		ns	6

A.C. CHARACTERISTICS (Continued)

RAM WARM-UP AND INITIALIZATION

64	TCLWZL	$\overline{WZ}$ from CLK↓ Delay		40		65	ns	7

SYNCHRONOUS μP PORT INTERFACE

11	TPEVCL	$\overline{PE}$ to CLK↓ Setup	30		50			2
12	TKVCL	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK↓ Setup	20		30		ns	1
13	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK↓ Hold	0		0		ns	
14	TKVCH	$\overline{RD}$, $\overline{WR}$, PCTL to CLK↑ Setup	20		30		ns	2

ASYNCHRONOUS μP PORT INTERFACE

15	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK↓ Setup	30		30		ns	8,9
16	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		2TCLCL+50		ns	
17	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, $\overline{WR}$↓ Delay 8207 8207 8207-2		TCLCL-20 TCLCL-30 TCLCL-30		 TCLCL-50	ns ns ns	1 2 3
18	TRWLPEX	$\overline{PE}$ to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+50		ns	
19	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$↓ Delay		TCLCL-30		TCLCL-50	ns	2
20	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+50		ns	2
21	TRWLPTV	PCTL from $\overline{RD}$, $\overline{WR}$↓ Delay		2TCLCL-20			ns	1
22	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$↓ Hold	3TCLCL+30				ns	1

A.C. CHARACTERISTICS (Continued)

RAM INTERFACE

Ref.	Symbol	Parameter	8207 & 8207-2		8207-5		Units	Notes
			Min.	Max.	Min.	Max.		
23	TAVCL	AL, AH, BS to CLK↓ Setup	35 + tASR		55 + tASR		ns	10
24	TCLAX	AL, AH, BS TO CLK↓ Hold	0		0		ns	
25	TCLLN	LEN from CLK↓ Delay		35		55	ns	
26	TCLRSL	$\overline{RAS}$↓ from CLK↓ Delay		35		55	ns	
28	TCLRSH	$\overline{RAS}$↑ from CLK↓ Delay		50		70	ns	
27	tRCD	$\overline{RAS}$ to $\overline{CAS}$ Delay	TCLCL/2-25 75 TCLCL-25		60		ns ns ns	11,13,14 12,13,14 1,13,14
29	tRAH	Row AO to $\overline{RAS}$↓ Hold	TCLCL/4-10 40 TCLCL/2-10 90		30		ns ns ns ns	11,13,15 12,13,15 1,13,15,16 13,15,17
30	tASR	Row AO to $\overline{RAS}$↓ Setup						10,18
31	tASC	Column AO to $\overline{CAS}$↓ Setup	5 5 TCLCL/2-26		5		ns ns ns	11,13,19,20 12,13,19 1,13,19
32	tCAH	Column AO to $\overline{CAS}$ Hold	(See DRAM Interface Tables)					21
33	TCLCSL	$\overline{CAS}$↓ from CLK↓ Delay	TCLCL/4+30	TCLCL/1.8+53	TCLCL/4+30	TCLCL/1.8+78	ns ns	11 12
34	TCLCSL	$\overline{CAS}$↓ from CLK↓ Delay		35			ns	1
35	TCLCSH	$\overline{CAS}$↑ from CLK↓ Delay		50		70	ns	
36	TCLW	WE from CLK↓ Delay		35		55	ns	
37	TCLTKL	$\overline{XACK}$↓ from CLK↓ Delay		35		55	ns	
38	TRWLTKH	$\overline{XACK}$↑ from $\overline{RD}$↑, $\overline{WR}$↑ Delay		50		60	ns	
39	TCLAKL	$\overline{AACK}$↓ from CLK↓ Delay		35		55	ns	
40	TCLAKH	$\overline{AACK}$↑ from CLK↓ Delay		50		70	ns	
41	TCLDL	$\overline{DBM}$ from CLK↓ Delay		35		55	ns	

ECC INTERFACE

42	TWRLFV	$\overline{FWR}$ from $\overline{WR}$↓ Delay 8207 8207-2		2TCLCL-40 TCLCL+ TCL-40		TCLCL+ TCL-65	ns ns	1,22 2,22
43	TFVCL	$\overline{FWR}$ to CLK↓ Setup	40		65		ns	23
44	TCLFX	$\overline{FWR}$ to CLK↓ Hold	0		0		ns	24
45	TEVCL	$\overline{ERROR}$ to CLK↓ Setup	20		30		ns	25,26
46	TCLEX	$\overline{ERROR}$ to CLK↓ Hold	0		0		ns	
47	TCLRL	R/$\overline{W}$ from CLK↓ Delay		40		55	ns	
48	TCLRH	R/$\overline{W}$↑ from CLK↓ Delay		50		70	ns	
49	TCEVCL	CE to CLK↓ Setup	20		30		ns	25,27
50	TCLCEX	CE to CLK↓ Hold	0		0		ns	
51	TCLES	$\overline{ESTB}$ from CLK↓ Delay		35		55	ns	

A.C. CHARACTERISTICS (Continued)

PORT SWITCHING AND LOCK

Ref.	Symbol	Parameter	8207 & 8207-2 Min.	8207 & 8207-2 Max.	8207-5 Min.	8207-5 Max.	Units	Notes
52	TCLMV	MUX from CLK↓ Delay		45		65	ns	
53	TCLPNV	PSEN from CLK↓ Delay	TCL TCL	60 TCL+35	TCL TCL	60 TCL+35	ns ns	28 29
54	TCLPSV	PSEL from CLK↓		35		55	ns	
55	TLKVCL	LOCK to CLK↓ Setup	30		50		ns	30,31
56	TCLLKX	LOCK to CLK↓ Hold	10		10		ns	30,31
57	TRWLLKV	LOCK from RD↓, WR↓ Delay		2TCLCL-30		2TCLCL-50	ns	31,32
58	TRWHLKX	LOCK to RD↑, WR↑ Hold	3TCLCL+30		3TCLCL+50		ns	31,32

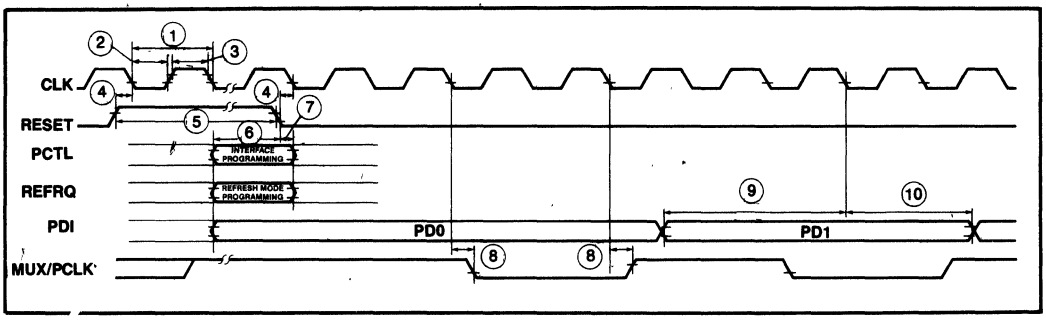
REFRESH REQUEST

59	TRFVCL	RFRQ to CLK↓ Setup	20		30		ns	
60	TCLRFX	RFRQ to CLK↓ Hold	10		10		ns	
61	TFRFH	Failsafe RFRQ Pulse Width	TCLCL+30		TCLCL+50		ns	33
62	TRFXCL	Single RFRQ Inactive to CLK↓ Setup	20		30		ns	34
63	TBRFH	Burst RFRQ Pulse Width	2TCLCL+30		2TCLCL+50		ns	33

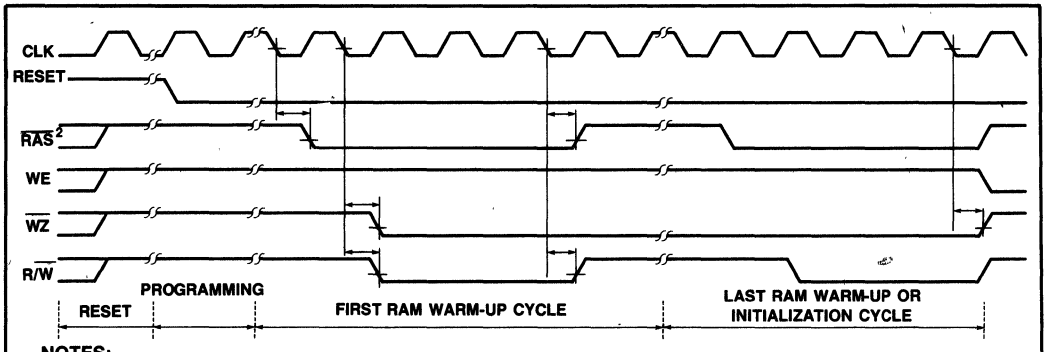
NOTES:
1. Specification when programmed in the Fast Cycle processor mode (iAPX 286 mode).
2. Specification when programmed in the Slow Cycle processor mode (iAPX 186 mode).
3. Must be programmed in Slow Cycle processor mode.
4. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
5. The first programming bit (PD0) is also sampled by RESET going low.
6. TCLPDX is guaranteed if programming data is shifted using PCLK.
7. WZ is issued only in ECC mode.
8. TRWVCL is not required for an asynchronous command except to guarantee its recognition at a particular clock edge.
9. Valid when programmed in either Fast or Slow Cycle mode.
10. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
11. When programmed in Slow Cycle mode and 125 ns ⩽ TCLCL < 200 ns.
12. When programmed in Slow Cycle mode and 200 ns ⩽ TCLCL.
13. Specification for Test Load conditions.
14. tRCD (actual) = tRCD (specification) +0.06 (ΔC_{RAS}) - 0.06 (ΔC_{CAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
15. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) - 0.022 (ΔC_{A0}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
16. When programmed in Fast Cycle mode (8207 only) and 62.5 ns ⩽ TCLCL < 200 ns.
17. When programmed in Fast Cycle mode (8207 only) and 200 ns ⩽ TCLCL.
18. tASR (actual) = tASR (specification) +0 06 (ΔC_{A0}) - 0.025 (ΔC_{RAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
19. tASC (actual) = tASC (specification) +0.06 (ΔC_{A0}) - 0.025 (ΔC_{CAS}) where ΔC = C (test load) - C (actual) in pF. (These are first order approximations.)
20. tASC is a function of clock frequency and thus varies with changes in frequency A mimimum value is specified.
21. See 8207 DRAM Interface Tables 14 - 18.
22. TWRLFV is defined for both synchronous and asynchronous $\overline{FWR}$. In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207. TCLFV is automatically guaranteed by TCLAV.
23. TFVCL is defined for synchronous $\overline{FWR}$
24. TCLFV is defined for both synchronous and asynchronous $\overline{FWR}$ In systems in which $\overline{FWR}$ is decoded directly from the address inputs to the 8207. TCLFV is automatically guaranteed by TCLAV.
25. $\overline{ERROR}$ and CE are set-up to CLK↓ in fast cycle mode and CLK↑ in slow cycle mode.
26. $\overline{ERROR}$ is set-up to the same edge as R/$\overline{W}$ is referenced to, in RMW cycles
27. CE is set-up to the same edge as WE is referenced to in RMW cycles.
28. Specification when TCL < 25 ns.
29. Specification when TCL ⩾ 25 ns.
30. Synchronous operation only. Must arrive by the second clock falling edge after the clock edge which recognizes the command in order to be effective.
31. LOCK must be held active for the entire period the opposite port must be locked out. One clock after the release of LOCK the opposite port will be able to obtain access to memory.
32. Asynchronous mode only. In this mode a synchronizer stage is used internally in the 8207 to synchronize up LOCK. TRWLLKV and TRWHLKX are only required for guaranteeing that LOCK will be recognized for the requesting port, but these parameters are not required for correct 8207 operation
33. TFRFH and TBRFH pertain to asynchronous operation only.
34. Single RFRQ cannot by supplied asynchronously
35. tR and tF are referenced from the 3.5V and 1.0V levels.

WAVEFORMS

Clock and Programming Timings



RAM Warm-up and Memory Initialization Cycles

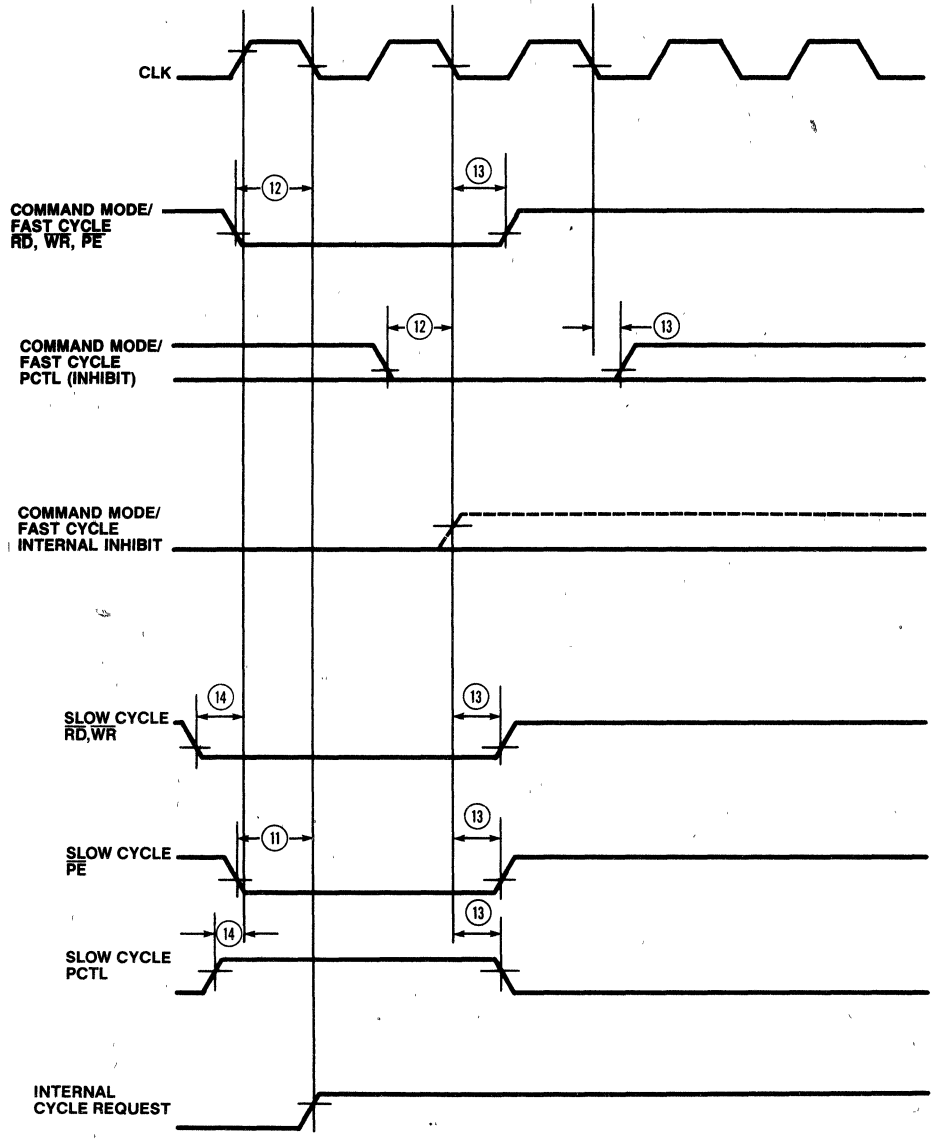


NOTES:

1. When in non-ECC mode or in ECC mode with the TM2 programming bit on, there are no initialization cycles, when in ECC mode with TM2 off, the dummy cycles are followed by initialization cycles.
2. The present example assumes a RAS four clocks long.

WAVEFORMS (Continued)

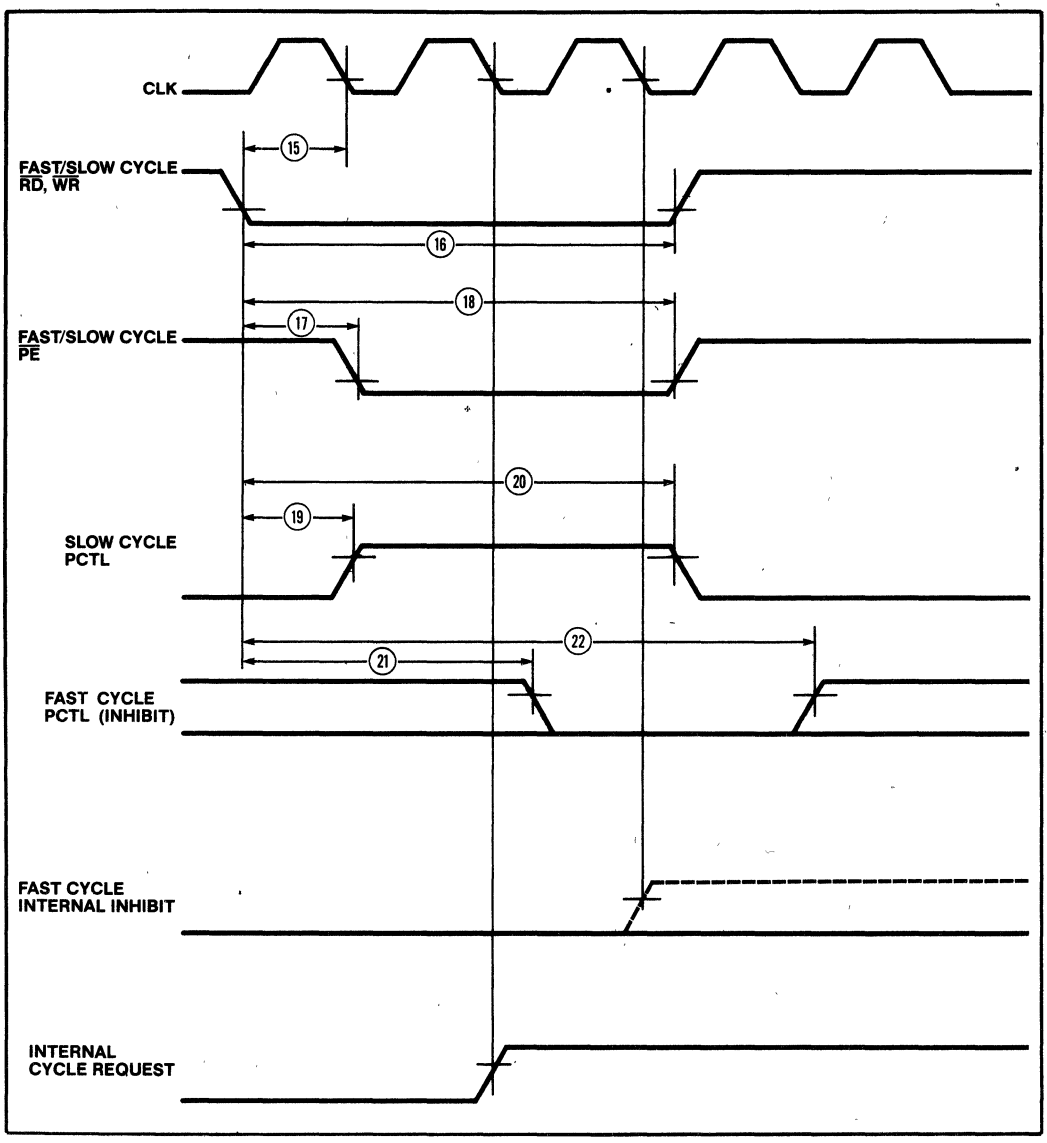
Synchronous Port Interface



NOTE:
Refer to Tables 12, 13 when using A.C. timing waveforms

WAVEFORMS (Continued)

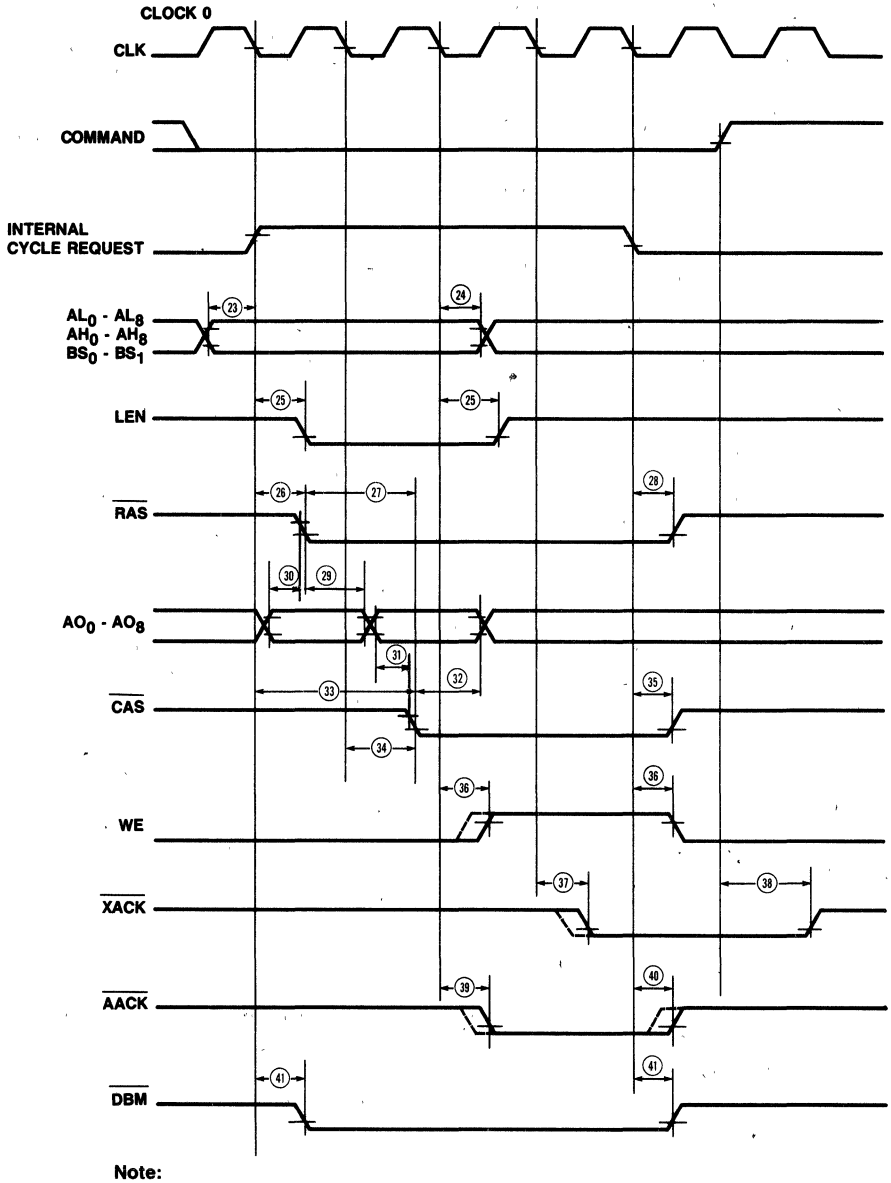
Asynchronous Port Interface



210463-003

WAVEFORMS (Continued)

RAM Interface Timing
ECC and Non-ECC Mode

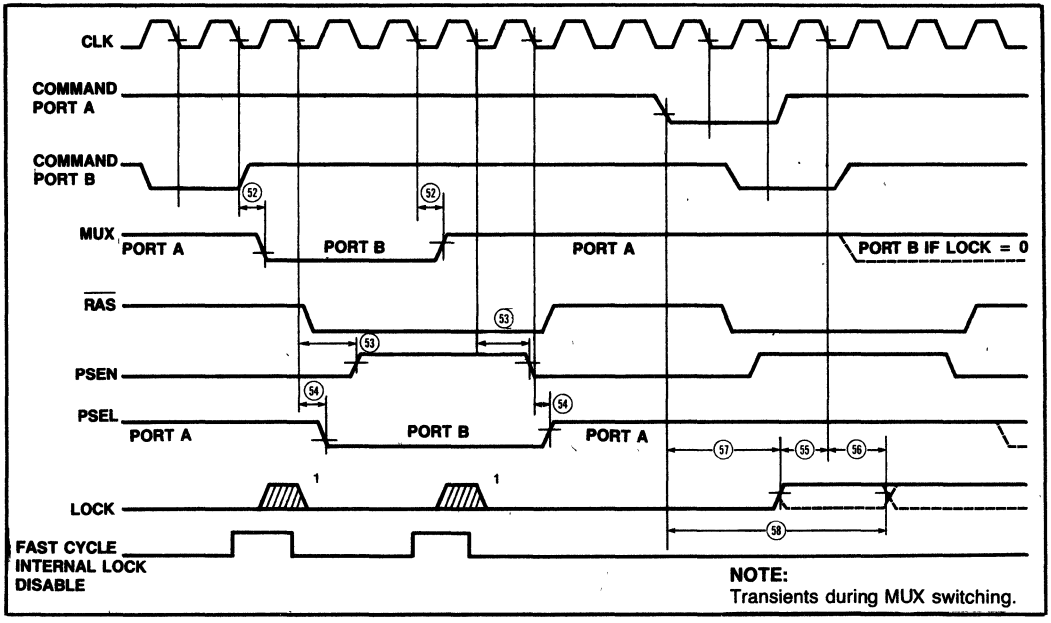


Note:
Dashed waveform indicates that either clock edge may cause the signal transition

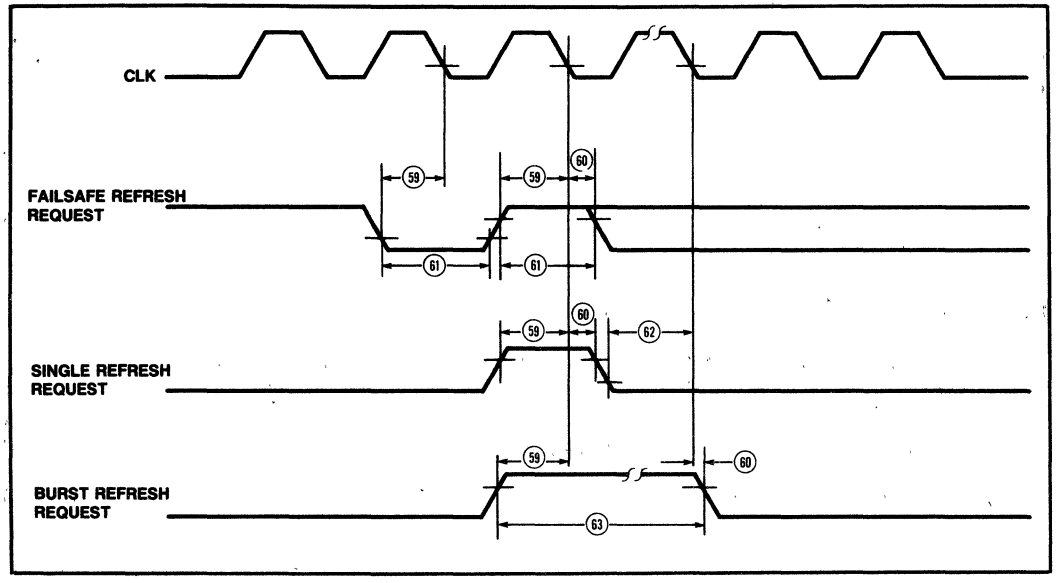
210463-003

WAVEFORMS (Continued)

Port Switching and Lock Timing



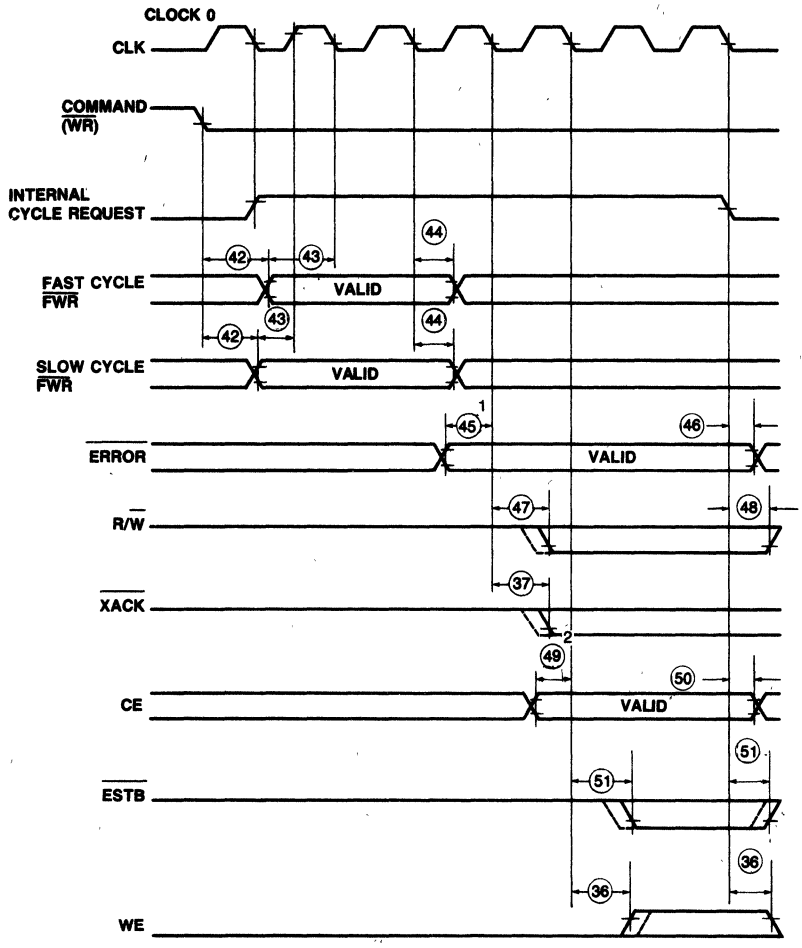
Refresh Request Timing



210463-003

WAVEFORMS (Continued)

ECC Interface Timing



NOTE:

1. This parameter is set-up to the falling edge of clock, as shown, for fast cycle configurations. It is set-up to the rising edge of clock if in slow cycle configurations. Table 13A shows which clock and clock edge these signals are set-up in the R/$\overline{W}$ L column.

2. CE is set-up to the same edge as WE is referenced to in RMW cycles.

CONFIGURATION TIMING CHARTS

The timing charts that follow are based on 8 basic system configurations where the 8207 operates.

Tables 10 and 11 give a description of non-ECC and ECC system configurations based on the 8207's PD0, PD3, PD4, PD10 and PD11 programming bits.

Table 10. Non-ECC System Configurations

Non-ECC Mode: PD0=0

Timing Conf.	$\overline{CFS}$(PD3)	$\overline{RFS}$(PD4)	EXT(PD10)	$\overline{FFS}$(PD11)
C_0	iAPX286(0)	FAST RAM(0)	NOT EXT(0)	10 MHZ(1)
C_0	iAPX286(0)	FAST RAM(0)	EXT(1)	10 MHZ(1)
C_0	iAPX286(0)	SLOW RAM(1)	NOT EXT(0)	10 MHZ(1)
C_0	iAPX286(0)	SLOW RAM(1)	EXT(1)	10 MHZ(1)
C_0	iAPX286(0)	FAST RAM(0)	NOT EXT(0)	16 MHZ(0)
C_1	iAPX286(0)	SLOW RAM(1)	NOT EXT(0)	16 MHZ(0)
C_1	iAPX286(0)	FAST RAM(0)	EXT(1)	16 MHZ(0)
C_2	iAPX286(0)	SLOW RAM(1)	EXT(1)	16 MHZ(0)
C_3	iAPX186(1)	FAST RAM(0)	NOT EXT(0)	8 MHZ(0)
C_3	iAPX186(1)	SLOW RAM(1)	NOT EXT(0)	8 MHZ(0)
C_3	iAPX186(1)	FAST RAM(0)	EXT(1)	8 MHZ(0)
C_3	iAPX186(1)	FAST RAM(0)	NOT EXT(0)	5 MHZ(1)
C_3	iAPX186(1)	FAST RAM(0)	EXT(1)	5 MHZ(1)
C_3	iAPX186(1)	SLOW RAM(1)	NOT EXT(0)	5 MHZ(1)
C_3	iAPX186(1)	SLOW RAM(1)	EXT(1)	5 MHZ(1)
C_4	iAPX186(1)	SLOW RAM(1)	EXT(1)	8 MHZ(0)

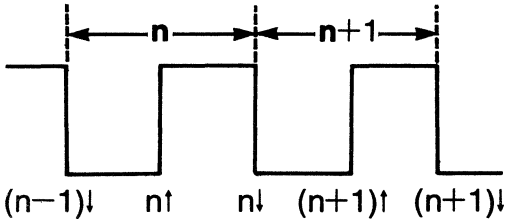
Table 11. ECC System Configurations

ECC Mode: PD0=1

Timing Conf.	CFS(PD3)	RFS(PD4)	$\overline{EXT}$(PD10)	FFS(PD11)
C_0	iAPX286(1)	SLOW RAM(0)	M/S EDCU(0)	10 MHZ(0)
C_0	iAPX286(1)	SLOW RAM(0)	M EDCU(1)	10 MHZ(0)
C_0	iAPX286(1)	FAST RAM(1)	M/S EDCU(0)	10 MHZ(0)
C_0	iAPX286(1)	FAST RAM(1)	M EDCU(1)	10 MHZ(0)
C_0	iAPX286(1)	FAST RAM(1)	M EDCU(1)	16 MHZ(1)
C_1	iAPX286(1)	SLOW RAM(0)	M EDCU(1)	16 MHZ(1)
C_2	iAPX286(1)	FAST RAM(1)	M/S EDCU(0)	16 MHZ(1)
C_3	iAPX286(1)	SLOW RAM(0)	M/S EDCU(0)	16 MHZ(1)
C_4	iAPX186(0)	SLOW RAM(0)	M/S EDCU(0)	5 MHZ(0)
C_4	iAPX186(0)	FAST RAM(1)	M/S EDCU(0)	5 MHZ(0)
C_4	iAPX186(0)	SLOW RAM(0)	M EDCU(1)	8 MHZ(1)
C_4	iAPX186(0)	FAST RAM(1)	M EDCU(1)	8 MHZ(1)
C_5	iAPX186(0)	SLOW RAM(0)	M/S EDCU(0)	8 MHZ(1)
C_5	iAPX186(0)	FAST RAM(1)	M/S EDCU(0)	8 MHZ(1)
C_6	iAPX186(0)	SLOW RAM(0)	M EDCU(1)	5 MHZ(0)
C_6	iAPX186(0)	FAST RAM(1)	M EDCU(1)	5 MHZ(0)

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓ to falling edge. A clock period is defined as the interval from a clock falling edge to the following falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 8207 output are tabulated in Table 12 for non-ECC mode, and Table 13 for ECC mode. "H" refers to the high-going transition, and "L" to low-going transition; "V" refers to valid, and "$\overline{V}$" to non-valid.

Clock 0 is defined as the clock in which the 8207 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8207 was performing another memory cycle. Clock 0 may be identified externally by the leading edge of RAS. which is always triggered on 0↓.

Notes for interpreting the timing charts.

1. **PSEL - valid** is given as the latest time it can occur. It is entirely possible for PSEL to become valid before the time given. In a refresh cycle, PSEL can switch as defined in the chart, but it has no bearing on the refresh cycle itself, but only on a subsequent cycle for one of the external ports.

2. **LEN - low** is given as the latest time it can occur. LEN is only activated by port A configured in Fast Cycle iAPX286 mode, and thus it is not activated by a refresh cycle, although it may be activated by port A during a refresh cycle.

3. **ADDRESS - col** is the time column address becomes valid.

4. In non-ECC mode the $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.

5. In ECC mode there are really seven types of cycles: Read without error, read with error, full write, partial write without error, partial write with error, refresh without error, and refresh with error. These cycles may be derived from the timing chart as follows:

   A. Read without error: Use row marked 'RD, RF'.

   B. Read with error: Use row marked 'RMW', except for $\overline{EAACK}$ and $\overline{LAACK}$, which should be taken from 'RD, RF'. If the error is uncorrectable, WE will not be issued.

   C. Full write: Use row marked 'WR'.

   D. Partial write without error: Use row marked 'RMW', except that $\overline{DBM}$ and $\overline{ESTB}$ will not be issued.

   E. Partial write with error: Use row marked 'RMW', except that $\overline{DBM}$ will not be issued. If the error is uncorrectable, WE will not be issued.

   F. Refresh without error: Use row marked 'RD, RF', except that $\overline{ESTB}$, $\overline{EAACK}$, $\overline{LAACK}$, and $\overline{XACK}$ will not be issued.

   G. Refresh with error: Use row marked 'RMW' except that $\overline{EAACK}$, $\overline{LAACK}$, $\overline{ESTB}$, and $\overline{XACK}$ will not be issued. If the error is uncorrectable WE will not be issued.

6. **XACK - high** is reset asynchronously by command going inactive and not by a clock edge.

7. **MUX - valid** is given as the latest time it can occur.

Table 13 A. Timing Chart — ECC Mode

		PSEN		PSEL		$\overline{DBM}$		$\overline{LEN}$		$\overline{RAS}$		$\overline{CAS}$		R/$\overline{W}$		WE	
C_n	CYCLE	H	L	V	$\overline{V}$	L	H	L	H	L	H	L	H	L	H	H	L
C_0	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_1	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓				
	WR	0↓	5↓	0↓	6↓			0↓	2↓	0↓	6↓	1↓	6↓	1↓	6↓	3↓	6↓
	RMW	0↓	8↓	0↓	9↓	0↓	9↓	0↓	2↓	0↓	9↓	1↓	9↓	4↓	9↓	6↓	9↓
C_2	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_3	RD, RF	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	5↓	1↓	7↓				
	WR	0↓	6↓	0↓	7↓			0↓	2↓	0↓	7↓	1↓	7↓	1↓	7↓	4↓	7↓
	RMW	0↓	10↓	0↓	11↓	0↓	11↓	0↓	2↓	0↓	11↓	1↓	11↓	5↓	11↓	8↓	11↓
C_4	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_5	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	0↓	5↓	1↑	5↓	3↑	5↓
	RMW	0↓	6↓	0↓	7↓	0↓	7↓	0↓	2↓	0↓	7↓	0↓	7↓	3↑	7↓	5↑	7↓
C_6	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	0↓	4↓				
	WR	0↓	3↓	0↓	4↓			0↓	2↓	0↓	4↓	0↓	4↓	1↑	4↓	2↑	4↓
	RMW	0↓	4↓	0↓	5↓	0↓	5↓	0↓	2↓	0↓	5↓	0↓	5↓	2↑	5↓	3↑	5↓

Table 13 B. Timing Chart — ECC Mode

		COL ADDR		$\overline{ESTB}$		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$		MUX	
C_n	CYCLE	V	V	L	H	L	H	L	H	L	H	V	V
C_0	RD, RF	0↓	2↓			2↓	5↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	2↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	2↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_1	RD, RF	0↓	3↓			3↓	6↓	3↓	6↓	4↓	RD	−2↓	2↓
	WR	0↓	3↓			2↓	5↓	2↓	5↓	4↓	WR	−2↓	2↓
	RMW	0↓	3↓	6↓	8↓	5↓	8↓	5↓	8↓	7↓	WR	−2↓	2↓
C_2	RD, RF	0↓	3↓			4↓	7↓	4↓	7↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_3	RD, RF	0↓	3↓			4↓	7↓	5↓	8↓	5↓	RD	−2↓	2↓
	WR	0↓	3↓			3↓	6↓	3↓	6↓	5↓	WR	−2↓	2↓
	RMW	0↓	3↓	8↓	10↓	7↓	10↓	7↓	10↓	9↓	WR	−2↓	2↓
C_4	RD, RF	0↓	2↓			1↓	3↓	2↑	4↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_5	RD, RF	0↓	2↓			2↓	4↓	3↑	5↑	3↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓
	RMW	0↓	2↓	5↑	6↑	3↓	5↓	4↑	6↑	5↓	WR	−1↓	2↓
C_6	RD, RF	0↓	2↓			1↓	3↓	1↑	3↑	2↑	RD	−1↓	2↓
	WR	0↓	2↓			1↓	3↓	1↑	3↑	2↓	WR	−1↓	2↓
	RMW	0↓	2↓	3↑	4↑	1↓	3↓	2↑	4↑	3↓	WR	−1↓	2↓

8207 — DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 8207 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

READ, WRITE, READ-MODIFY-WRITE & REFRESH CYCLES

tRAC: response parameter.
tCAC: response parameter.
tREF: See "Refresh Period Options"
tCRP: must be met only if CAS-only cycles, which do not occur with 8207, exist.
tRAH: See "A.C. Characteristics"
tRCD: See "A.C. Characteristics"
tASC: See "A.C. Characteristics"
tASR: See "A.C. Characteristics"
tOFF: response parameter.

READ & REFRESH CYCLES

tRCH: WE always goes active after $\overline{CAS}$ goes active, hence tRCH is guaranteed by tCPN.

WRITE CYCLE

tRC: guaranteed by tRWC.
tRAS: guaranteed by tRRW.
tCAS: guaranteed by tCRW.
tWCS: WE always activated after $\overline{CAS}$ is activated, except in memory initialization, hence tWCS is always negative (this is important for RMW only) except in memory initialization; in memory initialization tWCS is positive and has several clocks of margin.
tDS: system-dependent parameter.
tDH: system-dependent parameter.
tDHR: system-dependent parameter.

READ-MODIFY-WRITE CYCLE

tRWD: don't care in 8207 write cycles, but tabulated for 8207 RMW cycles.
tCWD: don't care in 8207 write cycles, but tabulated for 8207 RMW cycles.

Table 14. Non-ECC Mode - RD, RF Cycles

Parameter	Fast Cycle Configurations			Slow Cycle Configurations		Notes
	C_0	C_1	C_2	C_3	C_4	
tRP	3TCLCL−T26	4TCLCL−T26	4TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	2TCLCL−T34	3TCLCL−T34	3TCLCL−T34	3TCLCL−T34	4TCLCL−T34	1
tCSH	4TCLCL−T26	6TCLCL−T26	6TCLCL−T26	3TCLCL−T26	4TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	3/30	2
tRC	6TCLCL	8TCLCL	8TCLCL	5TCLCL	6TCLCL	1
tRAS	3TCLCL−T26	4TCLCL−T26	4TCLCL−T26	3TCLCL−T26	4TCLCL−T26	1
tCAS	3TCLCL−T34	5TCLCL−T34	5TCLCL−T34	3TCLCL−T34	4TCLCL−T34	1
tRCS	2TCLCL−TCL −T36−TBUF	2TCLCL−TCL −T36−TBUF	2TCLCL−TCL −T36−TBUF	1.5TCLCL−TCL −T36−TBUF	1.5TCLCL−TCL −T36−TBUF	1

Table 12 A. Timing Chart — Non-ECC Mode

		PSEN		PSEL		$\overline{DBM}$		LEN		$\overline{RAS}$		$\overline{CAS}$		WE	
C_n	CYCLE	H	L	V	$\overline{V}$	L	H	L	H	L	H	L	H	H	L
C_0	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	3↓	1↓	4↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_1	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_2	RD, RF	0↓	5↓	0↓	6↓	0↓	6↓	0↓	2↓	0↓	4↓	1↓	6↓		
	WR	0↓	4↓	0↓	5↓			0↓	2↓	0↓	5↓	1↓	5↓	2↓	5↓
C_3	RD, RF	0↓	2↓	0↓	3↓	0↓	3↓	0↓	2↓	0↓	3↓	0↓	3↓		
	WR	0↓	3↓	0↓	4↓			0↓	2↓	0↓	4↓	0↓	4↓	2↑	4↓
C_4	RD, RF	0↓	3↓	0↓	4↓	0↓	4↓	0↓	2↓	0↓	4↓	0↓	4↓		
	WR	0↓	3↓	0↓	4↓			0↓	2↓	0↓	4↓	0↓	4↓	2↑	4↓

Table 12 B. Timing Chart — Non-ECC Mode

		COL ADDR		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$		MUX	
C_n	CYCLE	V	$\overline{V}$	L	H	L	H	L	H	V	$\overline{V}$
C_0	RD, RF	0↓	2↓	1↓	4↓	2↓	5↓	3↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	2↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_1	RD, RF	0↓	3↓	2↓	5↓	2↓	5↓	4↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	3↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_2	RD, RF	0↓	3↓	2↓	5↓	3↓	6↓	4↓	$\overline{RD}$	−2↓	2↓
	WR	0↓	3↓	1↓	4↓	1↓	4↓	3↓	$\overline{WR}$	−2↓	2↓
C_3	RD, RF	0↓	2↓	0↓	2↓	1↓	3↓	2↓	$\overline{RD}$	−1↓	2↓
	WR	0↓	2↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$	−1↓	2↓
C_4	RD, RF	0↓	2↓	1↓	3↓	1↓	3↓	3↑	$\overline{RD}$	−1↓	2↓
	WR	0↓	2↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$	−1↓	2↓

Table 15. Non-ECC Mode - WR Cycle

Parameter	Fast Cycle Configurations			Slow Cycle Configurations		Notes
	C_0	C_1	C_2	C_3	C_4	
tRP	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	4TCLCL−T35	4TCLCL−T35	4TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	3/30	2
tRWC	8TCLCL	8TCLCL	8TCLCL	6TCLCL	6TCLCL	1
tRRW	5TCLCL−T26	5TCLCL−T26	5TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCRW	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tWCH	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	3TCLCL+TCL −T34	1, 3
tWCR	4TCLCL+TCL −T26	4TCLCL+TCL −T26	4TCLCL+TCL −T26	3TCLCL+TCL −T26	3TCLCL+TCL −T26	1, 3
tWP	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL+TCL −T36−TBUF	2TCLCL−T36 −TBUF	2TCLCL−T36 −TBUF	1
tRWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−T36 −TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

Table 16 A. ECC Mode — RD, RF Cycles

Parameter	Fast Cycle Mode				
	C_0	C_1	C_2	C_3	Notes
tRP	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCPN	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	3TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tCSH	6TCLCL−T26	6TCLCL−T26	7TCLCL−T26	7TCLCL−T26	1
tCAH	TCLCL−T34	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tT	3/30	3/30	3/30	3/30	2
tRC	8TCLCL	8TCLCL	9TCLCL	9TCLCL	1
tRAS	4TCLCL−T26	4TCLCL−T26	5TCLCL−T26	5TCLCL−T26	1
tCAS	5TCLCL−T34	5TCLCL−T34	6TCLCL−T34	6TCLCL−T34	1
tRCS	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	TCLCL−T36 −TBUF	1

Table 16 B. ECC Mode — RD, RF Cycles

Parameter	Slow Cycle Mode			
	C_4	C_5	C_6	Notes
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	1.5TCLCL−T35	1.5TCLCL−T35	1.5TCLCL−T35	1
tRSH	3TCLCL−T34	3TCLCL−T34	3TCLCL−T34	1
tCSH	4TCLCL−T26	4TCLCL−T26	4TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRC	5TCLCL	5TCLCL	5TCLCL	1
tRAS	3TCLCL−T26	3TCLCL−T26	3TCLCL−T26	1
tCAS	4TCLCL−T34	4TCLCL−T34	4TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1

Table 17 A. ECC Mode — WR Cycle

Parameters	Fast Cycle Mode				
	C_0	C_1	C_2	C_3	Notes
tRP	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	1
tCPN	4TCLCL–T35	4TCLCL–T35	4TCLCL–T35	4TCLCL–T35	1
tRSH	5TCLCL–T34	5TCLCL–T34	6TCLCL–T34	6TCLCL–T34	1
tCSH	6TCLCL–T26	6TCLCL–T26	7TCLCL–T26	7TCLCL–T26	1
tCAH	TCLCL–T34	2TCLCL–T34	2TCLCL–T34	2TCLCL–T34	1
tAR	2TCLCL–T26	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	1
tT	3/30	3/30	3/30	3/30	2
tRWC	9TCLCL	9TCLCL	10TCLCL	10TCLCL	1
tRRW	6TCLCL–T26	6TCLCL–T26	7TCLCL–T26	7TCLCL–T26	1
tCRW	5TCLCL–T34	5TCLCL–T34	6TCLCL–T34	6TCLCL–T34	1
tWCH	5TCLCL–T34	5TCLCL–T34	6TCLCL–T34	6TCLCL–T34	1, 4
tWCR	6TCLCL–T26	6TCLCL–T26	7TCLCL–T26	7TCLCL–T26	1, 4
tWP	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1
tRWL	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1
tCWL	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1

Table 17 B. ECC Mode — WR Cycle

Parameters	Slow Cycle Mode			
	C_4	C_5	C_6	Notes
tRP	2TCLCL–T26	2TCLCL–T26	2TCLCL–T26	1
tCPN	2.5TCLCL–T35	2.5TCLCL–T35	2.5TCLCL–T35	1
tRSH	5TCLCL–T34	5TCLCL–T34	4TCLCL–T34	1
tCSH	5TCLCL–T26	5TCLCL–T26	4TCLCL–T26	1
tCAH	2TCLCL–T34	2TCLCL–T34	2TCLCL–T34	1
tAR	2TCLCL–T26	2TCLCL–T26	2TCLCL–T26	1
tT	3/30	3/30	3/30	2
tRWC	7TCLCL	7TCLCL	6TCLCL	1
tRRW	5TCLCL–T26	5TCLCL–T26	4TCLCL–T26	1
tCRW	5TCLCL–T34	5TCLCL–T34	4TCLCL–T34	1
tWCH	5TCLCL–T34	5TCLCL–T34	4TCLCL–T34	1, 4
tWCR	5TCLCL–T26	5TCLCL–T26	4TCLCL–T26	1, 4
tWP	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	1
tRWL	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	1
tCWL	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	3TCLCL–TCL –T36–TBUF	1

Table 18 A. ECC Mode — RMW

Parameters	Fast Cycle Mode				Notes
	C_0	C_1	C_2	C_3	
tRP	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	1
tCPN	4TCLCL–T35	4TCLCL–T35	4TCLCL–T35	4TCLCL–T35	1
tRSH	8TCLCL–T34	8TCLCL–T34	10TCLCL–T34	10TCLCL–T34	1
tCSH	9TCLCL–T26	9TCLCL–T26	11TCLCL–T26	11TCLCL–T26	1
tCAH	TCLCL–T34	2TCLCL–T34	2TCLCL–T34	2TCLCL–T34	1
tAR	2TCLCL–T26	3TCLCL–T26	3TCLCL–T26	3TCLCL–T26	1
tT	3/30	3/30	3/30	3/30	2
tRWC	12TCLCL	12TCLCL	14TCLCL	14TCLCL	1
tRRW	9TCLCL–T26	9TCLCL–T26	11TCLCL–T26	11TCLCL–T26	1
tCRW	8TCLCL–T34	8TCLCL–T34	10TCLCL–T34	10TCLCL–T34	1
tRCS	TCLCL–T36 –TBUF	TCLCL–T36 –TBUF	TCLCL–T36 –TBUF	TCLCL–T36 –TBUF	1
tRWD	6TCLCL–T26	6TCLCL–T26	8TCLCL–T26	8TCLCL–T26	1
tCWD	5TCLCL–T34	5TCLCL–T34	7TCLCL–T34	7TCLCL–T34	1
tWP	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1
tRWL	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1
tCWL	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	3TCLCL–T36 –TBUF	1

Table 18 B. ECC Mode — RMW

Parameters	Slow Cycle Mode			Notes
	C_4	C_5	C_6	
tRP	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tCPN	2.5TCLCL−T35	2.5TCLCL−T35	2.5TCLCL−T35	1
tRSH	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tCSH	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCAH	2TCLCL−T34	2TCLCL−T34	2TCLCL−T34	1
tAR	2TCLCL−T26	2TCLCL−T26	2TCLCL−T26	1
tT	3/30	3/30	3/30	2
tRWC	9TCLCL	9TCLCL	7TCLCL	1
tRRW	7TCLCL−T26	7TCLCL−T26	5TCLCL−T26	1
tCRW	7TCLCL−T34	7TCLCL−T34	5TCLCL−T34	1
tRCS	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	0.5TCLCL−T36 −TBUF	1
tRWD	4TCLCL+TCL −T26	4TCLCL+TCL −T26	2TCLCL+TCL −T26	1
tCWD	4TCLCL+TCL −T34	4TCLCL+TCL −T34	2TCLCL+TCL −T34	1
tWP	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tRWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1
tCWL	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	3TCLCL−TCL −T36−TBUF	1

NOTES:
1. Minimum
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. Applies to the eight warm-up cycles and to the memory initilization cycles during initialization only.
5. TP = TCLCL
   T26 = TCLRSL
   T34 = TCLCSL
   T35 = TCLCSH
   T36 = TCLW
   TBUF = TTL Buffer delay

8208
DYNAMIC RAM CONTROLLER

- **0 Wait State, 8 MHz iAPX 186, iAPX 188, IAPX 86 and iAPX 88 Interface**
- **Provides all Signals necessary to Control 64K (2164A) and 256K Dynamic RAMs**
- **Supports Synchronous or Asynchronous Microprocessor Interfaces**
- **Automatic RAM Initialiation**
- **Directly Addresses and Drives up to 1 Megabyte without External Drivers**
- **Microprocessor Data Transfer and Advance Acknowledge Signals**
- **Four Programmable Refresh Modes**
- **+5 Volt Only HMOSII Technology for High Performance and Low Power**

The Intel 8208 Dynamic RAM Controller is a high performance, systems oriented, Dynamic RAM controller that is designed to easily interface 64K and 256K Dynamic RAMs to Intel and other microcomputer systems. The 8208 is designed to easily interface to the iAPX 186, iAPX 188, iAPX 86, and the iAPX 88 by strapping the programming pin to logic 0.

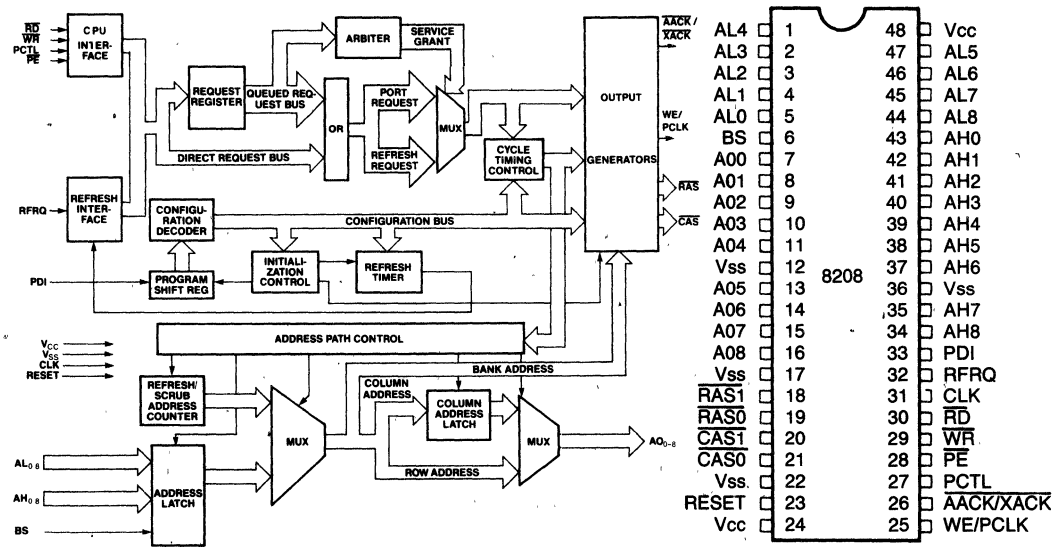


Figure 1. Block Diagram and Pinout Diagram

June, 1983
ORDER NUMBER 230734-001

Table 1. Pin Description

Symbol	Pin	Type	Name and Function
AL0 AL1 AL2 AL3 AL4 AL5 AL6 AL7 AL8	5 4 3 2 1 47 46 45 44	I I I I I I I I I	**ADDRESS LOW:** These lower order address inputs are used to generate the row address for the internal address multiplexer.
BS	6	I	**BANK SELECT:** This input is used to select one of the two banks of the dynamic RAM array as defined by the program-bit RB.
AO0 AO1 AO2 AO3 AO4 AO5 AO6 AO7 AO8	7 8 9 10 11 13 14 15 16	O O O O O O O O O	**ADDRESS OUTPUTS:** These outputs are designed to provide the row and column addresses, of either the CPU or the refresh counter, to the dynamic RAM array. These outputs drive the dynamic RAM array directly and need no external drivers.
VSS	12 17 22 36	I I I I	**GROUND** **GROUND** **GROUND** **GROUND**
$\overline{RAS0}$ $\overline{RAS1}$	19 18	O O	**ROW ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the row address, present on the AO0-8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.
$\overline{CAS0}$ $\overline{CAS1}$	21 20	O O	**COLUMN ADDRESS STROBE:** These outputs are used by the dynamic RAM array to latch the column address, present on the AO0-8 pins. These outputs are selected by the BS pin as programmed by program-bit RB. These outputs drive the dynamic RAM array directly and need no external drivers.
RESET	23	I	**RESET:** This active high signal causes all internal counters to be reset and upon release of RESET, data appearing at the PDI pin is clocked-in by the PCLK output. The states of the PDI, PCTL and RFRQ pins are sampled by RESET going inactive and are used to program the 8208. An 8 cycle dynamic RAM warm-up is performed after clocking PDI bits into the 8208.
WE/ PCLK	25	O	**WRITE ENABLE/PROGRAMMING CLOCK:** Immediately after a RESET this pin becomes PCLK and is used to clock serial programming data into the PDI pin. After the 8208 is programmed this active high signal provides the dynamic RAM array the write enable input for a write operation.
VCC	24 48	I I	**POWER:** +5 Volts. **POWER:** +5 Volts.
$\overline{AACK}$/ XACK	26	O	**ADVANCE ACKNOWLEDGE/TRANSFER ACKNOWLEDGE:** When the X programming bit is set to logic 0 this pin is $\overline{AACK}$ and indicates that the processor may continue processing and that data will be available when required. This signal is optimized for the system by programming the S program-bit for synchronous or asynchronous operation. The S programming bit determines whether this strobe will be early or late. If another dynamic RAM cycle is in progress at the time of the new request, the $\overline{AACK}$ is delayed. When the X programming bit is set to logic 1 this pin is $\overline{XACK}$ and indicates that data on the bus is valid during a read cycle or that data may be removed from the bus during a write cycle. $\overline{XACK}$ is a MULTIBUS compatible signal.
PCTL	27	I	**PORT CONTROL:** This pin is sampled on the falling edge of RESET. It configures the 8208 to accept command inputs or processor status inputs. If PCTL is low after RESET the 8208 is programmed to accept bus command inputs. If PCTL is high after RESET the 8208 is programmed to accept status inputs from iAPX 86 or iAPX 186 type processors. The $\overline{S2}$ status line should be connected to this input if programmed to accept iAPX 86 or iAPX 186 status inputs. When programmed to accept bus commands it should be tied low or it may be connected to INHIBIT when operating with MULTIBUS.
PE	28	I	**PORT ENABLE:** This pin serves to enable a RAM cycle request. It is generally decoded from the address bus.

Table 1. Pin Description (Continued)

Symbol	Pin	Type	Name and Function
WR	29	I	**WRITE:** This pin is the write memory request command input. This input also directly accepts the $\overline{S0}$ status line from Intel processors.
RD	30	I	**READ:** This pin is the read memory request command input. This input also directly accepts the $\overline{S1}$ status line from Intel processors.
CLK	31	I	**CLOCK:** This input provides the basic timing for sequencing the internal logic
RFRQ	32	I	**REFRESH REQUEST:** This input is sampled on the falling edge of RESET. If RFRQ is high at RESET then the 8208 is programmed for internal-refresh request or external-refresh request with failsafe protection. If RFRQ is low at RESET then the 8208 is programmed for external-refresh without failsafe protection or burst-refresh. Once programmed the RFRQ pin accepts signals to start an external-refresh with failsafe protection or external-refresh without failsafe protection or a burst-refresh.
PDI	33	I	**PROGRAM DATA INPUT:** This input is sampled by RESET going low. It programs the various user selectable options in the 8208. The PCLK pin shifts programming data into the PDI input from an external shift register. This pin may be strapped low to a default iAPX 186 (PDI=Low) mode configuration.
AH0 AH1 AH2 AH3 AH4 AH5 AH6 AH7 AH8	43 42 41 40 39 38 37 35 34	I I I I I I I I I	**ADDRESS HIGH:** These higher order address inputs are used to generate the column address for the internal address multiplexer.

GENERAL DESCRIPTION

The Intel 8208 Dynamic RAM Controller is a microcomputer peripheral device which provides the necessary signals to address, refresh and directly drive 64K and 256K dynamic RAMs.

The 8208 supports several microprocessor interface options including synchronous and asynchronous operations for iAPX 86, iAPX 186, iAPX 188 and MULTIBUS.

FUNCTIONAL DESCRIPTION

Processor Interface

The 8208 has control circuitry capable of supporting one of several possible bus structures. The 8208 may be programmed to run synchronous or asynchronous to the processor clock. (See Synchronous/Asynchronous Mode) The 8208 has been optimized to run synchronously with Intel's iAPX 86, iAPX 88, iAPX 186 and iAPX 188. When the 8208 is programmed to run in asynchronous mode, the 8208 inserts the necessary synchronization circuitry for the $\overline{RD}$, $\overline{WR}$, $\overline{PE}$, and PCTL inputs.

The 8208 achieves high performance (i.e. no wait states) by decoding the status lines directly from the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. The 8208 can also be programmed to receive read or write MULTIBUS commands or commands from a bus controller. (See Status/Command Mode)

The 8208 may be programmed to accept the clock of the iAPX 86, iAPX 88, iAPX 186 or 188. The 8208 adjusts its internal timing to allow for different clock frequencies of these microprocessors. (See Microprocessor Clock Frequency Option)

Figure 2 shows the different processor interfaces to the 8208 using the synchronous or asynchronous mode and status or command interface.

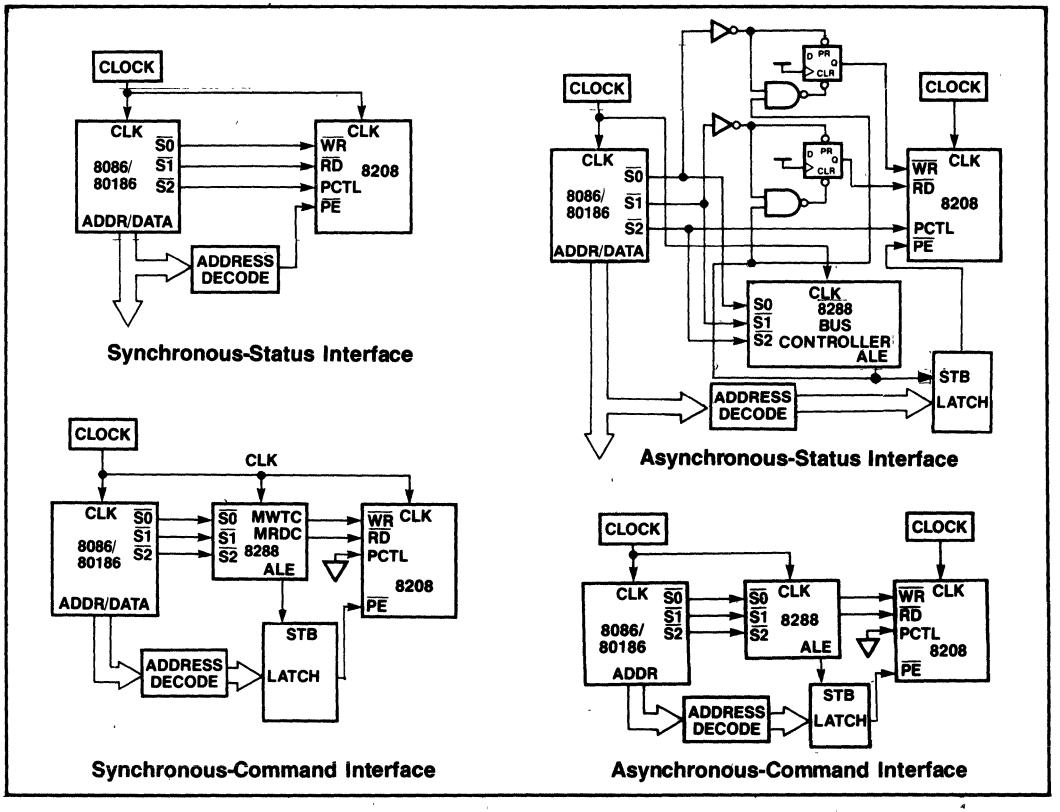
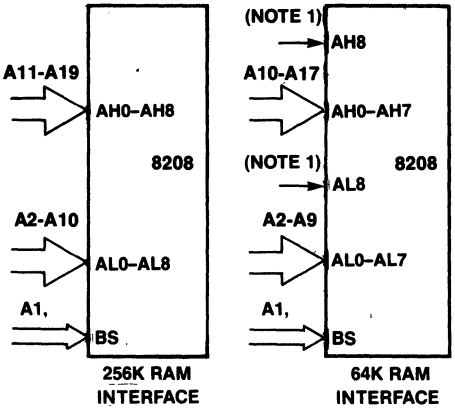


Figure 2. Interfaces Supported by the 8208.

Dynamic RAM Interface

The 8208 is capable of addressing 64K and 256K dynamic RAMs. Figure 3 shows the connection of the processor address bus to the 8208 using the different RAMs. The 8208 directly supports the 2164A RAM family or any RAM with similar timing requirements and responses.



NOTES:

1. Unassigned address input pins should be strapped high or low.
2. A0 along with BHE are used to select a byte within a processor word.
3. Low order address bit is used as a bank select input so that consecutive memory access requests are to alternate banks allowing bank interleaving of memory cycles.

Figure 3. Processor Address Interface to the 8208 Using 64K, and 256K, RAMS

The 8208 divides memory into two banks, each bank having its own Row ($\overline{RAS}$) and Column ($\overline{CAS}$) Address Strobe pair. This organization permits RAM cycle interleaving. RAM cycle interleaving overlaps the start of the next RAM cycle with the RAM precharge period of the previous cycle. Hiding the precharge period of one RAM cycle behind the data access period of the next RAM cycle optimizes memory bandwidth and is effective as long as successive RAM cycles occur in the alternate banks.

Successive data access to the same bank cause the 8208 to wait for the precharge time of the previous RAM cycle. But when the 8208 is programmed in an iAPX 186 synchronous configuration consecutive read cycles to the same bank does not result in additional wait states (i.e. 0 wait state reads result).

If not all RAM banks are occupied, the 8208 reassigns the $\overline{RAS}$ and $\overline{CAS}$ strobes to allow using wider data words without increasing the loading on the $\overline{RAS}$ and $\overline{CAS}$ drivers. Table 2 shows the bank selection decoding and the horizontal word expansion, including $\overline{RAS}$ and $\overline{CAS}$ assignments. For example, if only one RAM bank is occupied, then the two $\overline{RAS}$ and $\overline{CAS}$ strobes are activated with the same timing.

Table 2. Bank Selection Decoding and Word Expansion

Program Bit RB	Bank Input BS	8208 $\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	Illegal
1	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1

Program bit RB is not used to check the bank select input BS. The system design must protect from accesses to "illegal", non-existent banks of memory by deactivating the PE input when addressing an "illegal", non-existent bank of memory.

The 8208 adjusts and optimizes internal timings for either the fast or slow RAMs as programmed. (See RAM Speed Option)

Memory Initialization

After programming, the 8208 performs eight RAM "wake-up" cycles to prepare the dynamic RAM for proper device operation (during "warm-up" some RAM interface parameters may not be met, this should cause no harm to the dynamic RAM array).

Refresh

The 8208 provides an internal refresh interval counter and a refresh address counter to allow the 8208 to refresh memory. The 8208 will refresh 128 rows every 2 milliseconds or 256 rows every 4 milliseconds, which allows all RAM refresh options to be supported. In addition, there exists the ability to refresh 256 row address locations every 2 milliseconds via the Refresh Period programming option.

The 8208 may be programmed for any of five different refresh options: Internal refresh only, External refresh with failsafe protection, External refresh without failsafe protection, Burst Refresh modes, or no refresh. (See Refresh Options)

It is possible to decrease the refresh time interval by 10% 20% or 30%. This option allows the 8208 to compensate for reduced clock frequencies. Note that an additional 5% interval shortening is built-in in all refresh interval options to compensate for clock variations and non-immediate response to the internally generated refresh request. (See Refresh Period Options)

External Refresh Requests after RESET

External refresh requests are not recognized by the 8208 until after it is finished programming and preparing memory for access. Memory preparation includes 8 RAM cycles to prepare and ensure proper dynamic RAM operation. The time it takes for the 8208 to recognize a request is shown below.

eq. 8208 System Response: TRESP = PROG + TPREP
where: TPROG = (40) (TCLCL) which is programming time
TPREP = (8) (32) (TCLCL) which is the RAM warm-up time
if TCLCL = 125 ns then TRESP = 37 us

Reset

RESET is an asynchronous input, the falling edge of which is used by the 8208 to directly sample the logic levels of the PCTL, RFRQ, and PDI inputs. The internally synchronized falling edge of reset is used to begin programming operations (shifting in the contents of the external shift register, if needed, into the PDI input).

Differentiated reset is unnecessary when the default synchronization programming is used. ($\overline{S}$=0)

Until programming is complete the 8208 registers but does not respond to command or status inputs. A simple means of preventing commands or status from occurring during this period is to differentiate the system reset pulse to obtain a smaller reset pulse for the 8208. The total time of the 8208 reset pulse and the 8208 programming time must be less than the time before the first command the CPU issues in systems that alter the default port synchronization programming bit (default is synchronous interface).

The differentiated reset pulse would be shorter than the system reset pulse by at least the programming period required by the 8208. The differentiated reset pulse first resets the 8208, and system reset would reset the rest of the system. While the rest of the system is still in reset, the 8208 completes its programming. Figure 4 illustrates a circuit to accomplish this task.

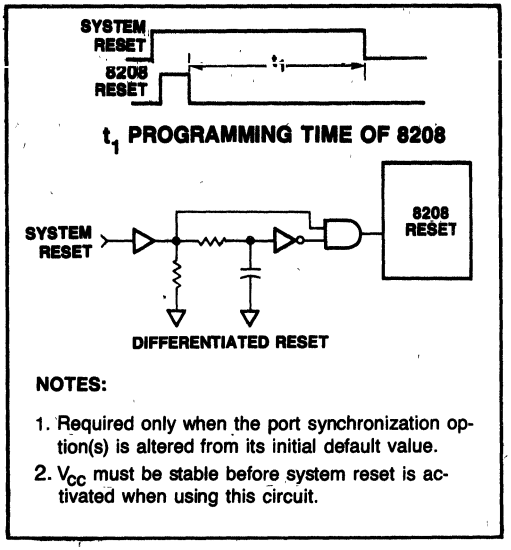


Figure 4. 8208 Differentiated Reset Circuit

Within four clocks after RESET goes active, all the 8208 outputs will go high, except for AO0-2, which will go low.

OPERATIONAL DESCRIPTION

Programming the 8208

The 8208 is programmed after reset. On the falling edge of RESET, the logic states of several input pins are latched internally. The falling edge of RESET actually performs the latching, which means that the logic levels on these inputs must be stable prior to that time. The inputs whose logic levels are latched at the end of reset are the PCTL, REFRQ, and PDI pins.

Status/Command Mode

The processor port of the 8208 is configured by the states of the PCTL pin. Which interface is selected depends on the state of the PCTL pin at the end of reset. If PCTL is high at the end of reset, the 80186 Status interface is selected; if it is low, then the MULTIBUS or Command interface is selected.

There exist two interface configurations, one for MULTIBUS memory commands, which is called the Command interface, and one for 8086, 8088, 80186 or 80188 status, called the 80186 Status interface. The Command interface also directly interfaces to the command lines of the bus controllers for the 8086, 8088.

The 80186 Status interface allows direct decoding of the status lines for the iAPX 86, iAPX 88, iAPX 186 and the iAPX 188. Table 3 shows how the status lines are decoded. Microprocessor bus controller read or write commands or MULTIBUS commands can also be directed to the 8208 when in Command mode.

Table 3. 8208 Response

8208 Command			Function	
PCTL	RD	WR	8086/80186 Status Interface	Multibus or Command Interface
0	0	0	IGNORE	IGNORE
0	0	1	IGNORE	READ
0	1	0	IGNORE	WRITE
0	1	1	IGNORE	IGNORE
1	0	0	READ	IGNORE
1	0	1	READ	INHIBIT
1	1	0	WRITE	INHIBIT
1	1	1	IGNORE	IGNORE

Refresh Options

Immediately after system reset, the state of the REFRQ input pin is examined. If REFRQ is high, the 8208 provides the user with the choice between self-refresh and user-generated refresh with failsafe protection. Failsafe protection guarantees that if the user does not come back with another refresh request before the internal refresh interval counter times out, a refresh request will be automatically generated. If the REFRQ pin is low immediately after a reset, then the user has the choice of a single external refresh cycle without failsafe, burst refresh or no refresh.

Internal Refresh Only

For the 8208 to generate internal refresh requests, it is necessary only to strap the REFRQ input pin high.

External Refresh with Failsafe

To allow user-generated refresh requests with failsafe protection, it is necessary to hold the REFRQ input high until after reset. Thereafter, a low-to-high transition on this input causes a refresh request to be generated and the internal refresh interval counter to be reset. A high-to-low transition has no effect on the 8208. A refresh request is not recognized until a previous request has been serviced.

External Refresh without Failsafe

To generate single external refresh requests without failsafe protection, it is necessary to hold REFRQ low until after reset. Thereafter, bringing REFRQ high for one clock period will cause a refresh request to be generated. A refresh request is not recognized until a previous request has been serviced.

Burst Refresh

Burst refresh is implemented through the same procedure as a single external refresh without failsafe (i.e., REFRQ is kept low until after reset). Thereafter, bringing REFRQ high for at least two clock periods will cause a burst of up to 128 row address locations to be refreshed. Any refresh request is not recognized until a previous request has been serviced (i.e. burst is completed).

No Refresh

It is necessary to hold REFRQ low until after reset. This is the same as programming External Refresh without Failsafe. No refresh is accomplished by keeping REFRQ low.

Option Program Data Word

The program data word consists of 9 program data bits, PD0-PD8. If the first program data bit, PD0 is set to logic 0, the 8208 is configured to support iAPX 186, 188, 86, or 88 systems. The remaining bits, PD1-PD8, may then be programmed to optimize a selected system configuration. A default of all zeros in the remaining program bits optimizes the 8208 timing for 8 MHz Intel CPUs using 150 nS (or faster) dynamic RAMs with no performance penalty. Figure 5 shows the various options that can be programmed into the 8208.

Using an External Shift Register

The 8208 may be programmed by using an external shift register with asynchronous load capability such as a 74LS165. The reset pulse serves to parallel load the shift register and the 8208 supplies the clocking signal (PCLK) to shift the data into the PDI programming pin. Figure 6 shows a sample circuit diagram of an external shift register circuit.

Serial data is shifted into the 8208 via the PDI pin (33), and clock is provided by the WE/PCLK pin (23), which generates a total of 9 clock pulses. After programming is complete, data appearing at the input of the PDI pin is ignored. WE/PCLK is a dual function pin.

During programming, it serves to clock the external shift register, and after programming is completed, it reverts to the write enable RAM control output pin. As the pin changes state to provide the write enable signal to the dynamic RAM array, it continues to clock the shift register. This does not present a problem because data at the PDI pin is ignored after programming. Figure 7 illustrates the timing requirements of the shift register.

PD8	PD7							PD0
$\overline{X}$	$\overline{FFS}$	$\overline{PLS}$	$\overline{CIO}$	$\overline{CI1}$	$\overline{RB}$	$\overline{RFS}$	$\overline{S}$	0

PROGRAM DATA BIT	NAME	POLARITY/FUNCTION
PD0	CFS	MUST BE ZERO
PD1	$\overline{S}$	$\overline{S}$ = 0 SYNCHRONOUS $\overline{S}$ = 1 ASYNCHRONOUS
PD2	$\overline{RFS}$	$\overline{RFS}$ = 0 FAST RAM $\overline{RFS}$ = 1 SLOW RAM
PD3	$\overline{RB}$	RAM BANK OCCUPANCY SEE TABLE 2
PD4	$\overline{CI1}$	COUNT INTERVAL BIT 1; SEE TABLE 6
PD5	$\overline{CI0}$	COUNT INTERVAL BIT 0; SEE TABLE 6
PD6	$\overline{PLS}$	$\overline{PLS}$ = 0 LONG REFRESH PERIOD $\overline{PLS}$ = 1 SHORT REFRESH PERIOD
PD7	$\overline{FFS}$	$\overline{FFS}$ = 0 FAST CPU FREQUENCY $\overline{FFS}$ = 1 SLOW CPU FREQUENCY
PD8	$\overline{X}$	X = 0 $\overline{AACK}$ X = 1 $\overline{XACK}$

Figure 5. Program Data Word

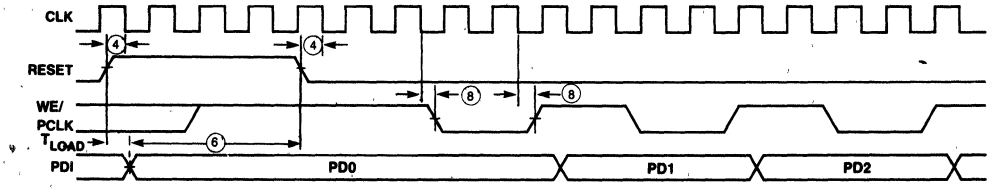
Default Programming Options

After reset, the 8208 serially shifts in a program data word via the PDI pin. This pin may be strapped low, or connected to an external shift register. Strapping PDI low causes the 8208 to default to the iAPX 186 system configuration. Table 4 shows the characteristics of the default configuration. If further system flexibility is needed, one external shift register, like a 74LS165, can be used to tailor the 8208 to its operating environment. Figure 8 illustrates an iAPX 186 and 8208 system.

Table 4. Programming, PDI Pin Tied to Ground.

Synchronous 80186 interface
2 RAM banks occupied
Fast processor clock frequency (8 MHz)
Fast RAM (Note 1)
Refresh interval uses 118 clocks
128 row refresh in 2 ms; 256 row refresh in 4 ms
Advanced ACK strobe

NOTE:
1. For iAPX 186 systems either slow or fast (150 or 100 ns) RAMS are ok to use.



NOTES:

TRTVCL — Reset is an asynchronous input, if reset occurs before TRTVCL, then it is guaranteed to be recognized.
TPGVCL — Minimum PDI valid time prior to reset going low.
TCLPC — MUX/PCLK delay.
TLOAD — Asynchronous load data propagation delay.

Figure 6. Timing Illustrating External Shift Register Requirements for Programming the 8208.

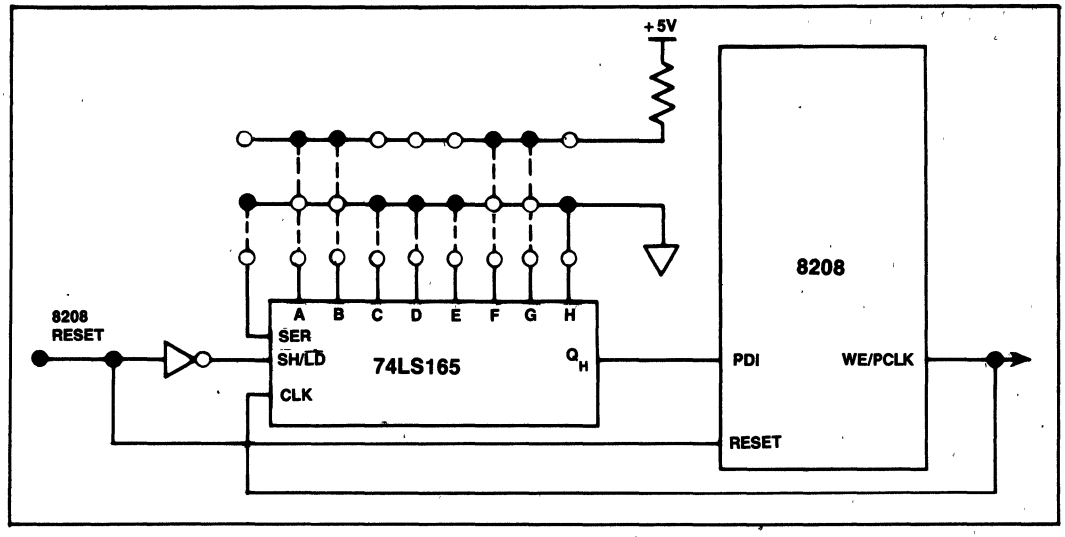


Figure 7. External Shift Register Interface

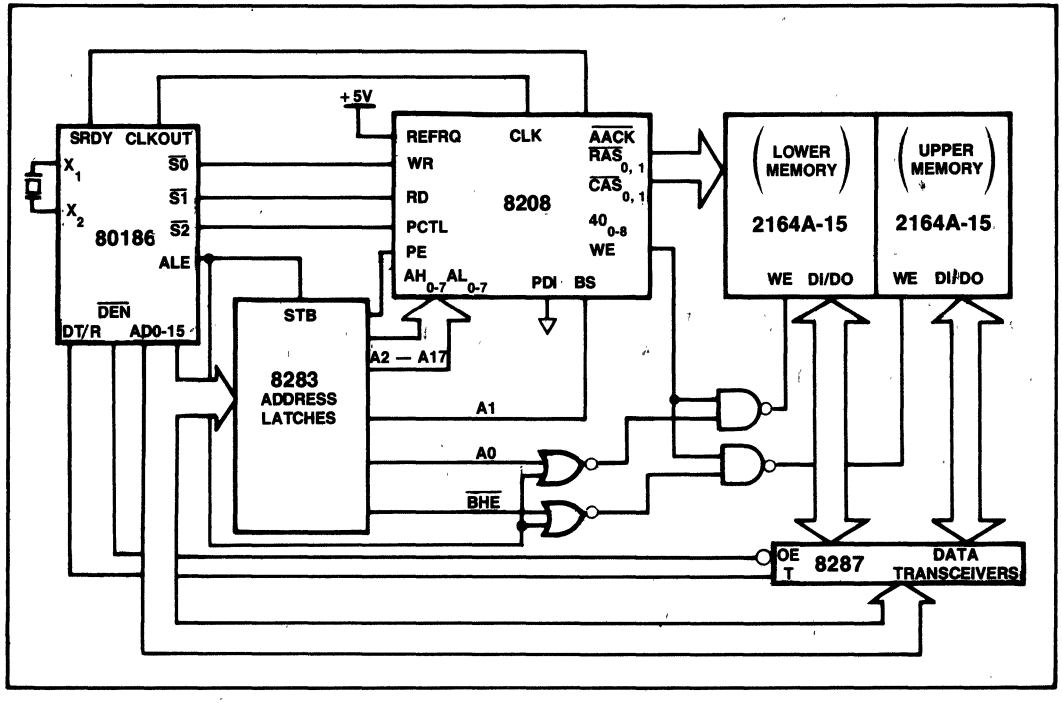


Figure 8. 8208 Interface to an 80186

Synchronous/Asynchronous Mode (S program bit)

The 8208 may be independently configured to accept synchronous or asynchronous commands ($\overline{RD}$, $\overline{WR}$, PCTL) and Port Enable ($\overline{PE}$) via the S program bit. The state of the S programming bit determines whether the interface is synchronous or asynchronous.

While the 8208 may be configured with either the 80186 Status or Command (MULTIBUS) interface in the Synchronous mode, certain restrictions exist in the Asynchronous mode. An Asynchronous-Command interface using the control lines of the MULTIBUS is supported, and an Asynchronous-80186 Status interface using the status lines of the 80186 is supported, with the use of TTL gates as illustrated in Figure 2. In the 80186 case, the TTL gates are needed to guarantee that status does not appear at the 8208's inputs too much before address, so that a cycle would start before address was valid.

Microprocessor Clock Cycle Option (CFS and FFS program bits

The 8208 can be programmed to interface with microprocessors with slow cycle microprocessors like the 8086, 8088, 80186, and 80188 cycle timing. The CFS bit configures the microprocessor interface to accept signals from either microprocessor group or commands from MULTIBUS. The CFS programming bit must be programmed to logic 0.

The FFS option is used to select the speed of the microprocessor clock. Table 5 shows the various microprocessor clock frequency options that can be programmed. The external clock frequency must be programmed so that the failsafe refresh repetition circuitry can adjust its internal timing accordingly to produce a refresh request as programmed.

Table 5. Microprocessor Clock Frequency Options.

Program Bits		Processor	Clock Frequency
CFS	FFS		
0	0	iAPX 86,88,186	5 MHz
0	1	iAPX 86, 88,186	8 MHz

RAM Speed Option (RFS program bit)

The RAM Speed programming option determines whether RAM timing will be optimized for a fast or slow RAM. Whether a RAM is fast or slow is measured relative to the 2118-10 (Fast) or the 2118-15 (Slow) RAM specifications.

Refresh Period Options (CI0 CI1 and PLS program bits)

The 8208 refreshes with either 128 rows every 2 milliseconds or the 256 rows every 4 milliseconds. This translates to one refresh cycle being executed approximately once every 15.6 microseconds. This rate can be changed to 256 rows every 2 milliseconds or a refresh approximately once every 7.8 microseconds via the Period Long/Short, program bit PLS, programming option.

The Count Interval 0 (CI0) and Count Interval 1 (CI1) programming options allow the rate at which refresh requests are generated to be increased in order to permit refresh requests to be generated close to the 15.6 or 7.8 microsecond period when the 8208 is operating at reduced frequencies. The interval between refreshes is decreased by 0%, 10%, 20%, or 30% as a function of how the count interval bits are programmed. A 5% guardband is built-in to allow for any clock frequency variations. Table 6 shows the refresh period options available.

Table 6. Refresh Count Interval Table

Freq. (MHz)	Ref. Period (μS)	CFS	PLS	FFS	Count Interval CI1, CI0 (8208 Clock Periods)			
					00 (0%)	01 (10%)	10 (20%)	11 (30%)
8	15.6	0	1	1	118	106	94	82
	7.8	0	0	1	59	53	47	41
5	15.6	0	1	0	74	66	58	50
	7.8	0	0	0	37	33	29	25

The numbers tabulated under Count Interval represent the number of clock periods between internal refresh requests. The percentages in parentheses represent the decrease in the interval between refresh requests. Note that all intervals have a built-in 5% (approximately) safety factor to compensate for minor clock frequency deviations and non-immediate response to internal refresh requests.

Processor Timing

In order to run without wait states, $\overline{AACK}$ must be used and connected to the $\overline{SRDY}$ input of the appropriate bus controller. $\overline{AACK}$ is issued relative to a point within the RAM cycle and has no fixed relationship to the processor's request. The timing is such, however, that the processor will run without wait states, barring refresh cycles, and bank precharge. In slow cycle, fast RAM configurations (8086, 80186), $\overline{AACK}$ is issued on the same same clock cycle that issues $\overline{RAS}$.

Port Enable ($\overline{PE}$) set-up time requirements depend on whether the 8208 is configured for synchronous or asynchronous, fast or slow cycle operation. In a synchronous fast cycle configuration, $\overline{PE}$ is required to be set-up to the same clock edge as the commands. If $\overline{PE}$ is true (low), a RAM cycle is started; if not, the cycle is aborted.

In asynchronous operation, $\overline{PE}$ is required to be set-up to the same clock edge as the internally synchronized status or commands. Externally, this allows the internal synchronization delay to be added to the status (or command) -to-$\overline{PE}$ delay time, thus allowing for more external decode time than is available in synchronous operation.

The minimum synchronization delay is the additional amount that $\overline{PE}$ must be held valid. If $\overline{PE}$ is not held valid for the maximum synchronization delay time, it is possible that $\overline{PE}$ will go invalid prior to the status or command being synchronized. In such a case the 8208 aborts the cycle. If a memory cycle intended for the 8208 is aborted, then no acknowledge ($\overline{AACK}$ or $\overline{XACK}$) is issued and the processor locks up in endless wait states.

Memory Acknowledge ($\overline{AACK}$, $\overline{XACK}$)

Two type of memory acknowledge signals are supplied by the 8208. They are the Advanced Acknowledge strobe ($\overline{AACK}$) and the Transfer Acknowledge strobe ($\overline{XACK}$). The S programming bit optimizes $\overline{AACK}$ for synchronous operation ("early" $\overline{AACK}$) or asynchronous operation ("late" $\overline{AACK}$). Both the early and late $\overline{AACK}$ strobes are two clocks long. The $\overline{XACK}$ strobe is asserted when data is valid (for reads) or when data may be removed (for writes) and meets the MULTIBUS requirements. $\overline{XACK}$ is removed asynchronously by the command going inactive.

Since in a asynchronous operation the 8208 removes read data before late $\overline{AACK}$ or $\overline{XACK}$ is recognized by the CPU, the user must provide for data latching in the system until the CPU reads the data. In synchronous operation data latching is unnecessary, since the 8208 will not remove data until the CPU has read it.

If the X programming bit is high, the strobe is configured as $\overline{XACK}$, while if the bit is low, the strobe is configured as $\overline{AACK}$.

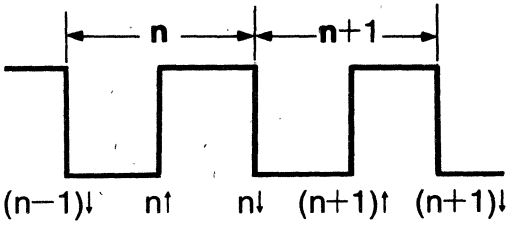
Data will always be valid a fixed time after the occurrence of the advanced acknowledge. Thus, the advanced acknowledge may also serve as a RAM cycle timing indicator.

General System Considerations

The $\overline{RAS}$0,1, $\overline{CAS}$0,1 and AO0-8 output buffers are designed to directly drive the heavy capacitive loads associated with dynamic RAM arrays. To keep the RAM driver outputs from ringing excessively in the system environment it is necessary to match the output impedance with the RAM array by using series resistors. Each application may have different impedance characteristics and may require different series resistance values. The series resistance values should be determined for each application.

Using the Timing Charts

The notation used to indicate which clock edge triggers an output transition is "n↑" or "n↓", where "n" is the number of clock periods that have passed since clock 0, the reference clock, and "↑" refers to rising edge and "↓" to falling edge. A clock period is defined as the interval from a clock falling edge to the follwoing falling edge. Clock edges are defined as shown below.



The clock edges which trigger transitions on each 8208 output are tabulated in Table 7. "H" refers to the high-going transition, and "L" to low-going transition; "V refers to valid, and "V" to non valid.

Clock 0 is defined as the clock in which the 8208 begins a memory cycle, either as a result of a port request which has just arrived, or of a port request which was stored previously but could not be serviced at the time of its arrival because the 8208 was performing another memory cycle, Clock 0 may be identified externally by the leading edge of $\overline{RAS}$, which is always triggered on 0↓.

NOTES FOR INTERPRETING THE TIMING CHARTS:

1. COLUMN ADDRESS is the time column address becomes valid.
2. The $\overline{CAS}$, $\overline{EAACK}$, $\overline{LAACK}$ and $\overline{XACK}$ outputs are not issued during refresh.
3. $\overline{XACK}$ - high is reset asynchronously by command going inactive and not be a clock edge.

Table 7. Timing Chart.

	$\overline{RAS}$		COLUMN ADDRESS		$\overline{CAS}$		WE		$\overline{EAACK}$		$\overline{LAACK}$		$\overline{XACK}$	
CYCLE	L	H	V	$\overline{V}$	L	H	H	L	L	H	L	H	L	H
RD, RF	0↓	2↓	0↓	2↓	0↓	3↓	—	—	0↓	2↓	1↓	3↓	2↓	$\overline{RD}$
WR	0↓	4↓	0↓	3↓	1↓	4↓	0↓	4↓	0↓	2↓	1↑	3↑	2↓	$\overline{WR}$

8208–DRAM Interface Parameter Equations

Several DRAM parameters, but not all, are a direct function of 8208 timings, and the equations for these parameters are given in the following tables. The following is a list of those DRAM parameters which have NOT been included in the following tables, with an explanation for their exclusion.

READ, WRITE REFRESH CYCLES

tRAC: response parameter.
tCAC: response parameter.
tREF: See "Refresh Period Options".
tCRP: must be met only if CAS-only cycles, which do not occur with 8208, exist.
tRAH: See "A.C. Characteristics"
tRCD: See "A.C. Characteristics"
tASC: See "A.C. Characteristics"
tASR: See "A.C. Characteristics"
tOFF: response parameter.

WRITE CYCLE

tDS: system-dependent parameter.
tDH: system-dependent parameter.
tDHR: system-dependent parameter.

NOTES:

1. Minimum.
2. Value on right is maximum; value on left is minimum.
3. Applies to the eight warm-up cycles during initialization only.
4. TP = TCLCL
   T26 = TCLRSL
   T34 = TCLCSL
   T35 = TCLCSH
   T36 = TCLW
   TBUF = TTL buffer delay

Table 8. RD, RF & WR Cycles

Parameter	Rd, RF Cycles	Notes
tRP	2TCLCL—T26	1
tCPN	2.5TCLCL—T35	1
tRSH	3TCLCL—T34	1
tCSH	3TCLCL—T26	1
tCAH	2TCLCL—T34	1
tAR	2TCLCL—T26	1
tT	3/30	2
tRC	4TCLCL	1
tRAS	2TCLCL—T26	1
tCAS	3TCLCL—T34	1
tRCS	1.5TCLCL—TCL—T36—TBUF	1
tRCH	0.5TCLCL—T34	1
Parameter	**WR Cycles**	**Notes**
tRP	2TCLCL—T26	1
tCPN	2.5TCLCL—T35	1
tRSH	3TCLCL—T34	1
tCSH	4TCLCL—T26	1
tCAH	2TCLCL—T34	1
tAR	3TCLCL—T26	1
tT	3/30	2
tRC	6TCLCL	1
tRAS	4TCLCL—T26	1
tCAS	TCLCL—T34	1
tWCH	3TCLCL—T34	1, 3
tWCR	4TCLCL—T26	1, 3
tWP	4TCLCL—T36—TBUF	1
tRWL	4TCLCL—T36—TBUF	1
tCWL	4TCLCL—T36—TBUF	1
TWCS	TCLCL—T36—TBUF	

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature
Under Bias ..... 0°C to +70°C
Storage Temperature ..... −65°C to +150°C
Voltage On Any Pin With
Respect to Ground ..... −.5V to +7V
Power Dissipation ..... 2 Watts

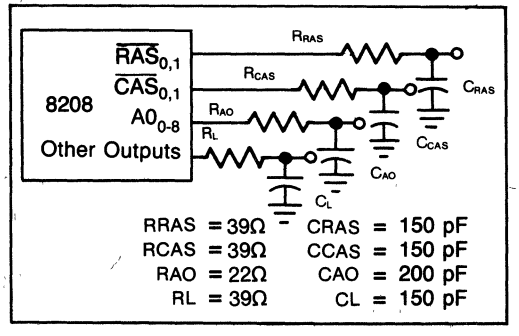
NOTICE: Stress above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to +70°C, V_{CC} = 5.0V ± 5% V_{SS} = GND)

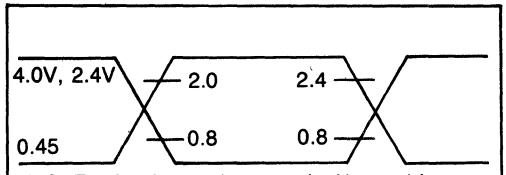
Symbol	Parameter	Min.	Max.	Units	Comments
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage		0.45	V	Note 1
V_{OH}	Output High Voltage	2.4		V	Note 1
V_{ROL}	RAM Output Low Voltage		0.45	V	Note 1
V_{ROH}	RAM Output High Voltage	2.6		V	Note 1
I_{CC}	Supply Current		280	mA	T_A = 25°C
I_{LI}	Input Leakage Current		+10	μA	0V ≤ V_{IN} ≤ V_{CC}
V_{CL}	Clock Input Low Voltage	−0.5	+0.6	V	
V_{CH}	Clock Input High Voltage	3.8	V_{CC} + 0.5	V	
C_{IN}	Input Capacitance		20	pF	fc = 1 MHz

NOTES:
1. I_{OL} = 8 mA and I_{OH} = -0.2 mA (typically I_{OL} = 10 mA and I_{OH} = −0.25mA)

A.C. Testing Load Circuit



A.C. Testing Input, Output Waveform



A.C. Testing inputs (except clock) are driven at 2.4V for a logic "1" and 0.45V for a logic "0" (clock is driven at 4.0V and 0.45V for logic "1" and "0" respectively). Timing measurements are made at 2.0V, 2.4V for logic "1" and 0.8 V for logic "0".

A.C. CHARACTISTICS (T_a = 0 to 70 C V_{CC} = +5V ± 5%)

Measurements made with respect to $\overline{RAS0}$, 1, $\overline{CAS0}$ 1, $\overline{AO0\text{-}8}$ are at 2.4V and 0.8 V. All other pins are measured at 2.0V and 0.8V. All times in nsec unless otherwise indicated. AC testing done with specified test load.

CLOCK AND PROGRAMMING

Ref.	Symbol	Parameter	8208 Min.	8208 Max.	8208-5 Min.	8208-5 Max.	Units	Notes
—	tF	Clock Rise Time		10		15	ns	
—	tF	Clock Fall Time		10		15	ns	
1	TCLCL	Clock Period	125	500	200	500	ns	
2	TCL	Clock Low Time	TCLCL/2-12		TCLCL/2-12		ns	
3	TCH	Clock High Time	TCLCL/3		TCLCL/3		ns	
4	TRTVCL	Reset to CLK↓ Setup	40		65		ns	1
5	TRTH	Reset Pulse Width	4 TCLCL		4 TCLCL		ns	
6	TPGVRTL	$\overline{PCTI}$, PDI, RFRQ to RESET↓ Setup	125		200		ns	2
7	TRTLPGX	$\overline{PCTI}$, RFRQ to RESET↓ Hold	10		10		ns	
8	TCLPC1	PCLK from CLK↓ Delay		45		65	ns	
9	TPDVCL	PDI to CLK↓ Setup	60		100		ns	
10	TCLPDX	PDI to CLK↓ Hold	40		65		ns	3

SYNCHRONOUS μP INTERFACE

11	TKVCH	$\overline{RD}$, $\overline{WR}$, PCTL TO CLK↑ Setup	20		30		ns	
12	TCLKX	$\overline{RD}$, $\overline{WR}$, $\overline{PE}$, PCTL to CLK↓ Hold	0		0		ns	
13	TPEVCL	$\overline{PE}$ to CLK↓ Setup	30		50		ns	

ASYNCHRONOUS μP INTERFACE

14	TRWVCL	$\overline{RD}$, $\overline{WR}$ to CLK↓ Setup	20		30		ns	
15	TRWL	$\overline{RD}$, $\overline{WR}$ Pulse Width	2TCLCL+30		2TCLCL+50		ns	
16	TRWLPEV	$\overline{PE}$ from $\overline{RD}$, $\overline{WR}$↓ Delay		TCLCL-30		TCLCL-50	ns	
17	TRWLPEX	$\overline{PE}$ to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+50		ns	
18	TRWLPT	PCTL from $\overline{RD}$, $\overline{WR}$↓ Delay		TCLCL-30		TCLCL-50	ns	
19	TRWLPTX	PCTL to $\overline{RD}$, $\overline{WR}$↓ Hold	2TCLCL+30		2TCLCL+50		ns	

RAM INTERFACE

20	TAVCL	AL, AH, BS to CLK↓ Setup	35+tASR		55+tASR		ns	4
21	TCLAX	AL, AH, BS to CLK↓ Hold	0		0		ns	
22	TCLRSL	$\overline{RAS}$↓ from CLK↓ Delay		35		55	ns	
23	tRCD	$\overline{RAS}$ to $\overline{CAS}$ Delay	TCLCL/2-25 75		 60		ns ns	5,7,8 6,7,8
24	TCLRSH	$\overline{RAS}$↑ from CLK↓ Delay		50		70	ns	
25	tASR	Row AO to $\overline{RAS}$↓ Setup						4,10
26	tRAH	Row AO to $\overline{RAS}$↓ Hold	TCLCL/4-10 40		 30		ns ns	5,7,9 6,7,9
27	tASC	Column AO to $\overline{CAS}$↓ Setup	5		5		ns	7,11,12
28	tCAH	Column AO to $\overline{CAS}$ Hold		(See DRAM Interface Tables)				13

A.C. CHARACTERISTICS (Continued)

RAM Interface (Continued)

Ref.	Symbol	Parameter	8208		8208-5		Units	Notes
			Min.	Max.	Min.	Max.		
29	TCLCSL	$\overline{CAS}$↓ from CLK↓ Delay	TCLCL/2	TCLCL/1.8+53	TCLCL/2	TCLCL/1.8+78	ns	
30	TCLCSH	$\overline{CAS}$↑ from CLK↓ Delay		50		70	ns	
31	TCLWH	WE↑ from CLK↓ Delay	TCLCL/2	TCLCL/1.8+53	TCLCL/2	TCLCL/1.8+78	ns	
32	TCLWL	WE↓ from CLK↓ Delay		35		55	ns	
33	TCLTKL	$\overline{XACK}$↓ from CLK↓ Delay		35		55	ns	
34	TRWLTKH	$\overline{XACK}$↑ from $\overline{RD}$↑, $\overline{WR}$↑ Delay		50		60	ns	
35	TCLAKL	$\overline{AACK}$↓ from CLK↓ Delay		35		55	ns	
36	TCLAKH	$\overline{AACK}$↑ from CLK↓ Delay		50		70	ns	

REFRESH REQUEST

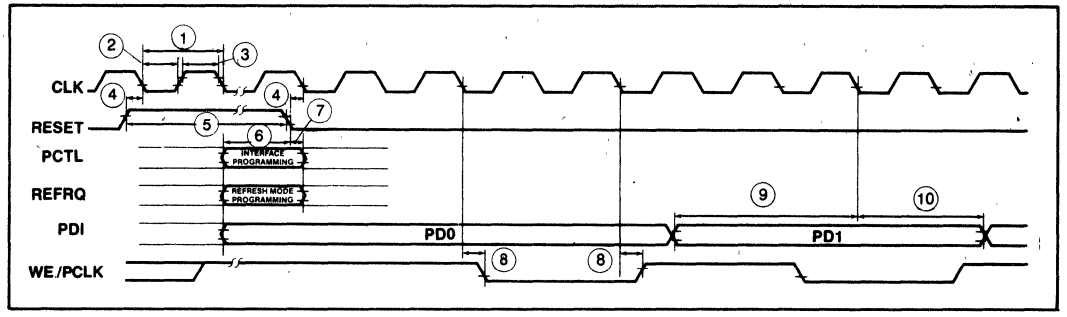
37	TRFVCL	RFRQ to CLK↓ Setup	20		30		ns	
38	TCLRFX	RFRQ to CLK↓ Hold	10		10		ns	
39	TFRFH	Failsafe RFRQ Pulse Width	TCLCL+30		TCLCL+50		ns	14
40	TRFXCL	Single RFRQ inactive to CLK↓ Setup	20		30		ns	15
41	TBRFH	Burst RFRQ Pulse Width	2TCLCL+30		2TCLCL+50		ns	14

NOTES:

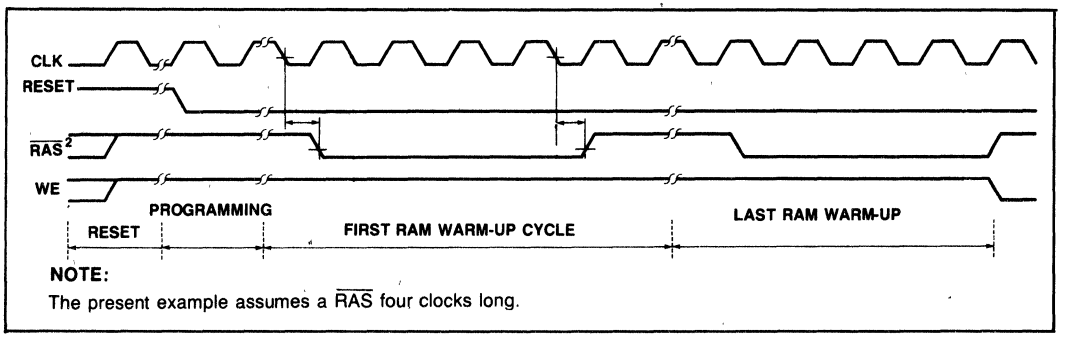
1. RESET is internally synchronized to CLK. Hence a set-up time is required only to guarantee its recognition at a particular clock edge.
2. The first programming bit (PD0) is also sampled by RESET going low.
3. TCLPDX is guaranteed if programming data is shifted using PCLK.
4. tASR is a user specified parameter and its value should be added accordingly to TAVCL.
5. When programmed in Slow Cycle mode and 125 ns $\leqslant$ TCLCL $<$ 200 ns.
6. When programmed in Slow Cycle mode and 200 ns $\leqslant$ TCLCL.
7. Specification for Test Load Conditions.
8. tRCD (actual) = tRCD (specification) + 0.06 (ΔC_{RAS}) − 0.06 (ΔC_{CAS})
   where ΔC = C (test load) − C (actual) in pF.
9. tRAH (actual) = tRAH (specification) + 0.06 (ΔC_{RAS}) − 0.022 (ΔC_{AO})
   where ΔC = C (test load) − C (actual) in pF.
10. tASR (actual) = tASR (specification) + 0.06 (ΔC_{AO}) − 0.025 (ΔC_{RAS})
    where ΔC = C (test load) − C (actual) in pF.
11. tASC (actual) = tASC (specification) + 0.06 (ΔC_{AO}) − 0.025 (ΔC_{CAS})
    where ΔC = C (test load) − C (actual) in pF.
12. tASC is a function of clock frequency and thus varies with changes in frequency. A minimum value is specified.
13. See 8208 DRAM Interface Tables.
14. TFRFH and TBRFH pertain to asynchronous operation only.
15. Single RFRQ cannot be supplied asynchronously.

WAVEFORMS

Clock and Programming Timings

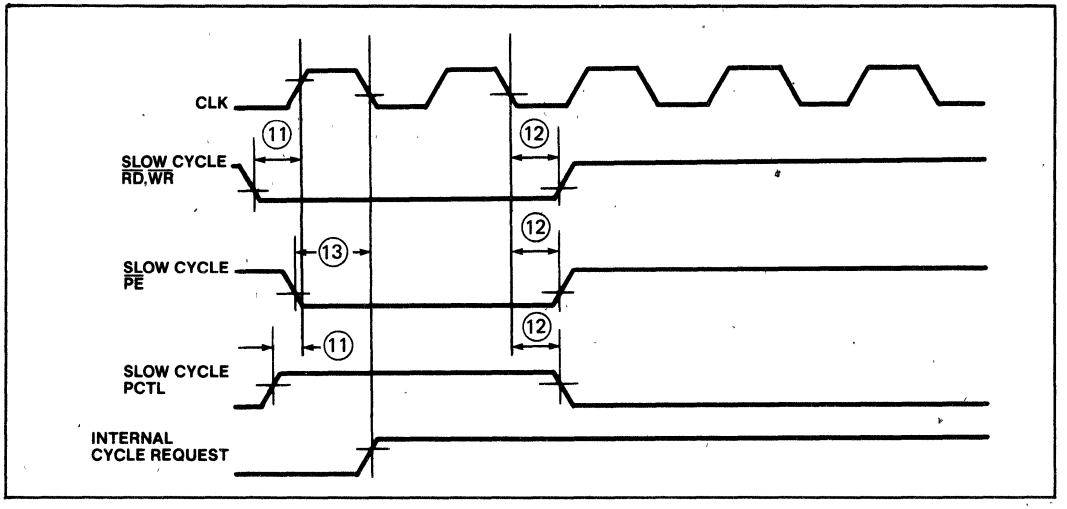


RAM Warm-up Cycles



NOTE:
The present example assumes a $\overline{RAS}$ four clocks long.

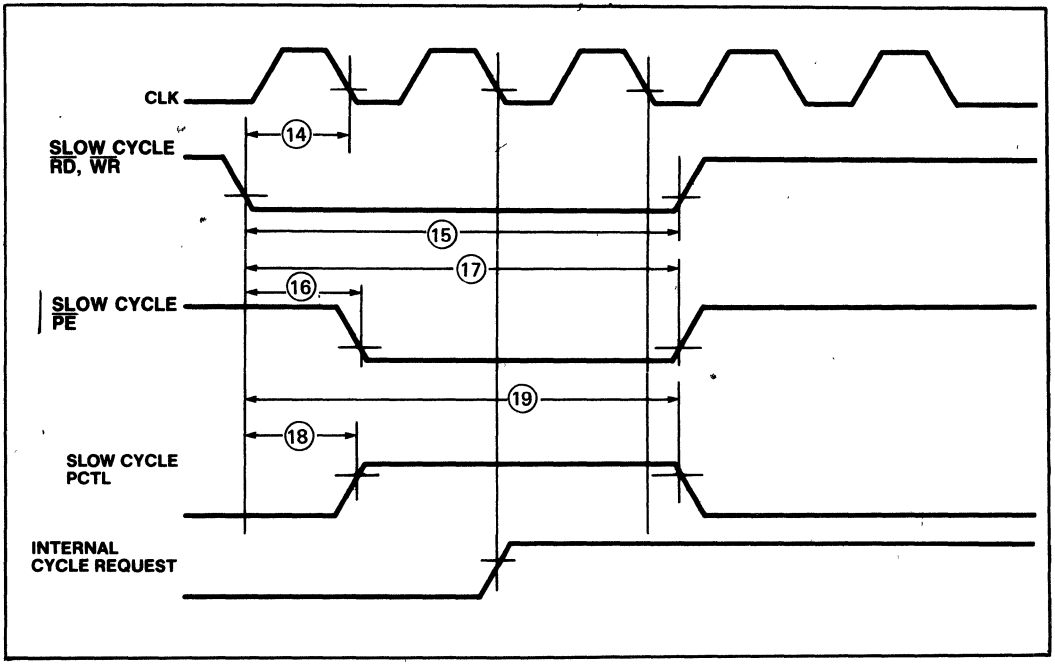
Synchronous Port Interface



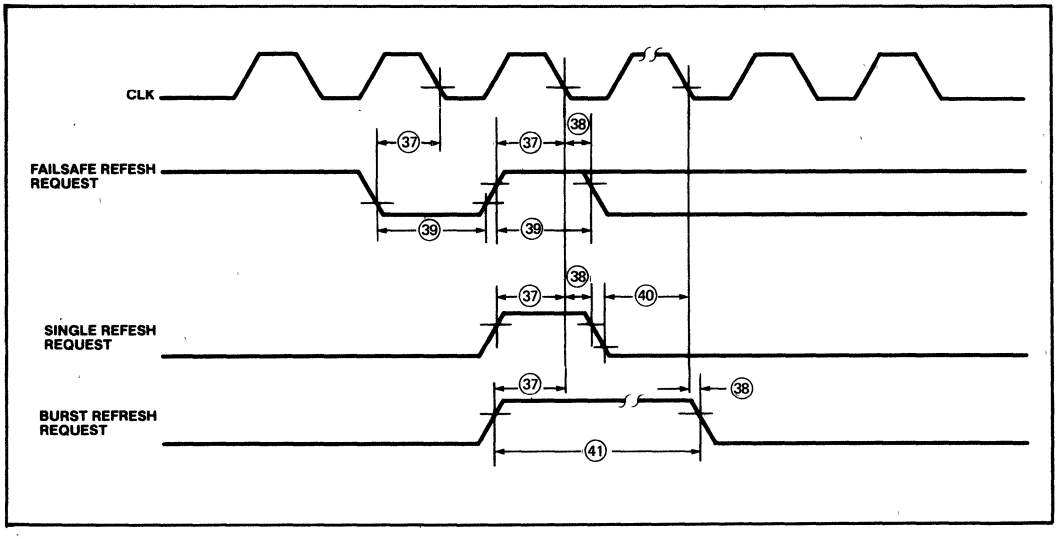
230734-001

WAVEFORMS (Continued)

Asynchronous Port Interface



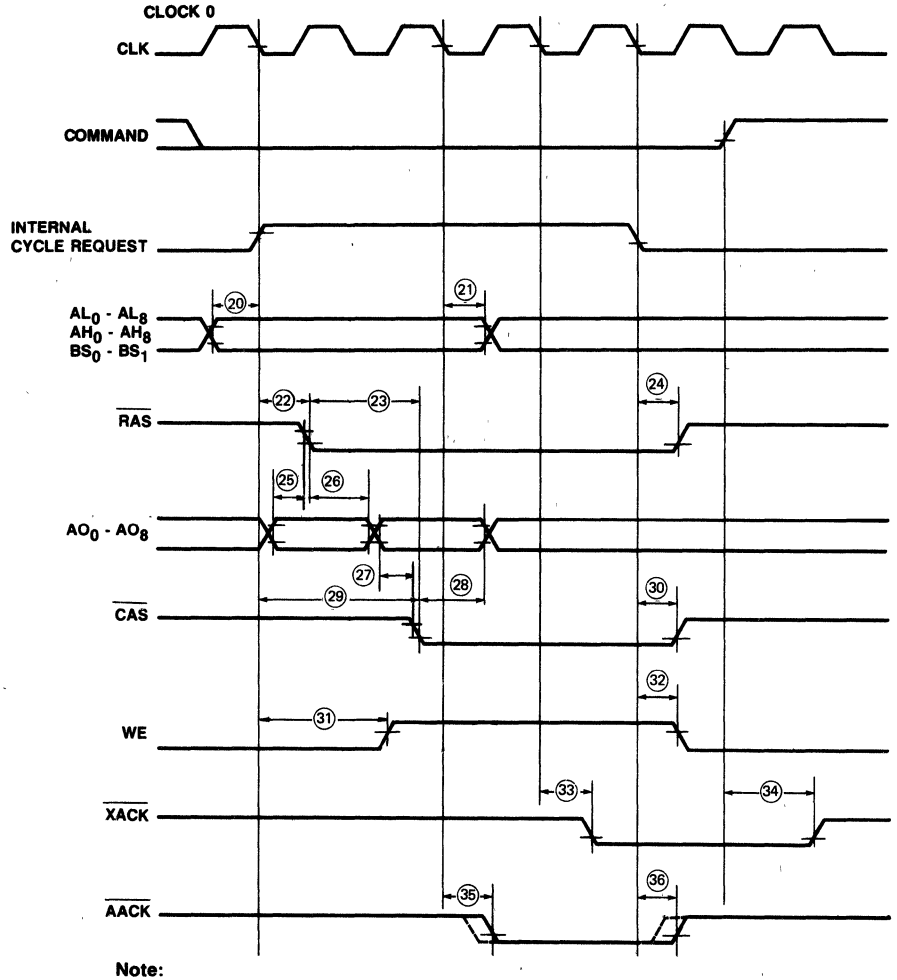
Refresh Request Timing



230734-001

WAVEFORMS (Continued)

RAM Interface Timing



Note:
Dashed waveform indicates that either clock edge may cause the signal transition

8207 User's Manual

AUGUST 1983

NOVEMBER 1983
ORDER NUMBER: 230822-001

CHAPTER 1
INTRODUCTION

This guide is a supplement to the 8207 Data Sheet[1] and is intended as a design aid and not a stand-alone description of the 8207. The reader should already have read and have a copy of the 8207 Data Sheet, 8206 Error Detection and Correction Unit Data Sheet (EDCU), a microprocessor Data Sheet, or a Multibus bus specification for interfacing to the 8207, and a dynamic RAM Data Sheet[2].

The Intel 8207 Advanced Dynamic RAM Controller is a high performance, highly integrated device designed to interface 16k, 64k, and 256k dynamic RAMS to Intel microprocessors. The 8207, with the 8206, provides complete control for memory initialization, error correction, and automatic error scrubbing.

The 8207 has several speed selected versions. The standard part is specified for clock speeds up to 16 MHz in "fast cycle" configurations, and up to 8 Mhz in "slow cycle" configurations. The -2 part can only be used in slow cycle configurations up to 8 Mhz. The -5 is limited to slow cycle configurations of 5 MHz or less and, as a result, has some relaxed A.C. timings.

NOTE:
(1) The most current Data Sheet is dated July, 1983
(2) All RAM cycle timings and references are based on Intel's 2164A Dynamic RAMs, APR '82 Data Sheet.

CHAPTER 2
PROGRAMMING THE 8207

The many configurations of bus structures, RAM speeds, and system requirements that the 8207 supports require the 8207 to be programmable. The 8207 will modify its outputs to provide the best performance possible. The 8207 must be told what type of interface the memory commands will arrive on, what type of RAM (speed, refresh rate) is being used, the clock rate, and others.

The 8207 uses two means to be informed of the user's requirements. It reads in a 16 bit serial program word and examines the logic states on several input pins. The pins that are sampled for a logic level give the user options on the types of refresh and memory command input timing.

Input Pin Options

The three input pins that configure part of the 8207 are: PCTLA, PCTLB, and REFRQ. Let's examine the options in refresh types the REFRQ pin provides.

Refresh types:

The 8207 gives the user a choice of the following refresh types.

1) Internal Refresh: All refresh cycles are generated internally — based on an internal programmable time.

2) External Refresh with Failsafe: If the external logic does not generate a refresh cycle within the programmed period, the 8207 will.

3) External Refresh - No Failsafe or No Refresh; All refresh cycles are generated at times by the user. This is for systems that cannot tolerate the random delay imposed by refresh (i.e. graphics memory).

4) Burst Refresh: The 8207 generates up to 128 consecutive refresh cycles and must be requested by external logic. Memory requests will be performed when the burst is completed.

The 8207 examines the state of the REFRQ pin when RESET goes inactive. This timing is shown in the "Clock and Programming Timings" waveforms in the Data Sheet.

If REFRQ is sampled active by the falling edge of RESET, the 8207's internal timer is enabled. The timer's period is determined by the CI0, CI1, and PLS bits in the program word. External refresh cycles are generated by a low to high transition on the REFRQ input. This transition, besides generating a refresh cycle, also resets the internal timer to zero. Simply tie REFRQ to Vcc if internal refresh is required.

If REFRQ is seen low at the falling edge of RESET, the internal timer is deactivated. All refresh cycles must either be done by external logic or by accessing all RAM (internal) rows within a 2 ms period.

Once the no failsafe option is programmed, the 8207 will generate a burst of up to 128 refresh cycles when the REFRQ input goes from low to high and sampled high for two consecutive clock edges. These cycles are internally counted and the 8207 stops when the refresh address counter reaches the value $XX1111111_2$ (X = don't care; see ***Refresh Counter*** section). If prior to the burst request the counter is at $XX1111110_2$ then only 2 refresh cycles would be generated.

For a single refresh cycle to be generated via external logic, the REFRQ input will have to go from low to high and then sample high by a falling 8207 clock edge. Since external refresh requests typically arrive asynchronously with respect to the 8207's clock, this requires the REFRQ to be synchronized to the 8207 clock when programmed in the failsafe mode. This is to ensure that the request is seen for one clock - no more, no less. If no external synchronization is performed, then the 8207 could do random burst cycles.

Processor Interface Options:

The PCTLA, PCTLB input pins will program the 8207 to accept either the standard demultiplexed $\overline{RD}$ and $\overline{WR}$ inputs, or to directly decode the status outputs of Intel's iAPX86, 88 family of microprocessors. The state definitions of the status lines and their timings, relative to the processor clock, differ for the 8086 family and the iAPX286 processor. Table 1 illustrates how the 8207 interprets these inputs after the PCTL pins are programmed.

If PCTL is seen high, as RESET goes inactive, and 8086 status interface is enabled. The commands arriving at the 8207 are sampled by a rising clock edge. When PCTL is low, the 80286 status and Multibus command interface is selected. These commands are sampled by the 8207 by a falling clock edge.

More information on interfacing to processors is contained in the *Microprocessor Interface section.*

Table 1. Status Coding of 8086, 80186 and 80286

Status Code			Function	
$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	8086/80186	80286
0	0	0	Interrupt	Interrupt
0	0	1	I/O Read	I/O Read
0	1	0	I/O Write	I/O Write
0	1	1	Halt	Idle
1	0	0	Instruction Fetch	Halt
1	0	1	Memory Read	Memory Read
1	1	0	Memory Write	Memory Write
1	1	1	Idle	Idle

8207 Response

8207 Command			Function	
PCTL	$\overline{RD}$	$\overline{WR}$	8086 Status Interface	Command Interface
0	0	0	Ignore	Ignore
0	0	1	Ignore	Read
0	1	0	Ignore	Write
0	1	1	Ignore	Ignore
1	0	0	Read	Ignore
1	0	1	Read	Inhibit
1	1	0	Write	Inhibit
1	1	1	Ignore	Ignore

Programming Word

The 8207 requires more information to operate in a wide variety of systems. The 8207 alters its timings and pin functions to operate with the 8206 ECC chip. The programming options allow the designer to use asynchronous or synchronous buses, various clock rates, various speeds and types of RAM, and others. This is detailed in Table 2.

This data is supplied to the 8207 over the PDI input pin. There are two methods of supplying this data. One is to strap the PDI pin high or low with the subsequent restrictions on your system. Table

3 shows the required system configuration. Note that your only option when strapping this pin high or low is error correction or not.

If any other configurations are required, then the programming data will have to be supplied by one or two 74LS165 type shift registers. Note that the sense of the bits in the program word change between ECC and non-ECC configurations.

Table 2a.
Non-ECC Mode Program Data Word

PD15								PD8 PD7							PD0
0	0	TM1	$\overline{PPR}$	$\overline{FFS}$	EXT	$\overline{PLS}$	CI0	CI1	$\overline{RB1}$	$\overline{RB0}$	$\overline{RFS}$	$\overline{CFS}$	SB	$\overline{SA}$	0

Program Data Bit	Name	Polarity/Function
PD0	ECC	ECC = 0 For non-ECC mode
PD1	$\overline{SA}$	$\overline{SA}$ = 0 Port A is synchronous $\overline{SA}$ = 1 Port A is asynchronous
PD2	SB	SB = 0 Port B is asynchronous SB = 1 Port B is synchronous
PD3	$\overline{CFS}$	$\overline{CFS}$ = 0 Fast-cycle iAPX 286 mode $\overline{CFS}$ = 1 Slow-cycle iAPX 86 mode
PD4	$\overline{RFS}$	$\overline{RFS}$ = 0 Fast RAM $\overline{RFS}$ = 1 Slow RAM
PD5 PD6	$\overline{RB0}$ $\overline{RB1}$	RAM bank occupancy See Table 4
PD7 PD8	CI1 CI0	Count interval bit 1: see Table 6 in 8207 data sheet Count interval bit 0: see Table 6 in 8207 data sheet
PD9	$\overline{PLS}$	$\overline{PLS}$ = 0 Long refresh period $\overline{PLS}$ = 1 Short refresh period
PD10	EXT	EXT = 0 Not extended EXT = 1 Extended
PD11	$\overline{FFS}$	$\overline{FFS}$ = 0 Fast CPU frequency $\overline{FFS}$ = 1 Slow CPU frequency
PD12	$\overline{PPR}$	$\overline{PPR}$ = 0 Most recently used port priority $\overline{PPR}$ = 1 Port A preferred priority
PD13	TM1	TM1 = 0 Test mode 1 off TM1 = 1 Test mode 1 enabled
PD14	0	Reserved must be zero
PD15	0	Reserved must be zero

Table 2b
ECC Mode Program Data Word

PD15								PD8 PD7							PD0
$\overline{TM2}$	RB1	RB0	PPR	FFS	$\overline{EXT}$	PLS	$\overline{CI0}$	$\overline{CI1}$	XB	$\overline{XA}$	RFS	CFS	$\overline{SB}$	SA	1

Program Data Bit	Name	Polarity/Function	
PD0	ECC	ECC = 1	ECC mode
PD1	SA	SA = 0 SA = 1	Port A is asynchronous (late AACK) Port A is synchronous (early AACK)
PD2	$\overline{SB}$	$\overline{SB}$ = 0 $\overline{SB}$ = 1	Port B is synchronous (early AACK) Port B is asynchronous (late AACK)
PD3	CFS	CFS = 0 CFS = 1	Slow-cycle iAPX 86 mode Fast-cycle iAPX 286 mode
PD4	RFS	RFS = 0 RFS = 1	Slow RAM Fast RAM
PD5	$\overline{XA}$	$\overline{XA}$ = 0 $\overline{XA}$ = 1	Multibus-compatible XACKA AACKA not multibus-compatible
PD6	XB	XB = 0 XB = 1	AACKB not multibus-compatible Multibus-compatible XACKB
PD7 PD8	CI1 CI0	Count interval bit 1: see Table 6 in 8207 data sheet Count interval bit 0: see Table 6 in 8207 data sheet	
PD9	PLS	PLS = 0 PLS = 1	Short refresh period Long refresh period
PD10	$\overline{EXT}$ $\overline{EXT}$	$\overline{EXT}$ = 0 $\overline{EXT}$ = 1	Master and slave EDCU Master EDCU only
PD11	FFS	FFS = 0 FFS = 1	Slow CPU frequency Fast CPU frequency
PD12	PPR	PPR = 0 PPR = 1	Port A preferred priority Most recently used port priority
PD13 PD14	RB0 RB1	RAM bank occupancy See Table 4	
PD15	TM2	TM2 = 0 TM2 = 1	Test mode 2 enabled Test mode 2 off

Table 3. 8207 Default Programming

Port A is Synchronous—has early AACK
Port B is Asychronous—has late AACK
Fast RAM
Refresh Interval uses 236 clocks
128 Row refresh in 2 ms; 256 Row refresh in 4 ms
Fast Processor Clock Frequency (16 MHz)
"Most Recently Used" Priority Scheme
4 RAM banks occupied

Reset

If Port A is changed to an asynchronous interface (via the SA bit), then one of two precautions must be taken. Either a differentiated reset must be provided, or else software must not access the 8207 controller RAM for a short period. The 8207 is either adding or deleting internal synchronizing circuits. If a command is received during this changing, the 8207 may not perform properly. This is required only if Port A is changed to asynchronous, or if Port B is changed to synchronous.

Several of the bits in the program word determine a particular configuration of the 8207 (reference Tables 10, 11 and the 8207 Data Sheet). The bits are: CFS, CLOCK fast or slow; RFS, RAM access time fast or slow (fast refers to 100 ns - slow is everything greater); and EXT, for memory data word widths greater than 16 (22) bits. Generally speaking, CO is the fastest configuration at clock frequencies up to 16 MHz, both in the ECC or non-ECC charts. 'C3' is the fastest for 8 MHz clocks in non-ECC mode, and 'C4' is the fastest configuration when using ECC.

Take, for example, a 16 MHz 8207 clock with no error correction, a 16 bit word, and 150 ns (slowly) dynamic RAMs. Table 10, in the 8207 data sheet, is used to arrive at the configuration "C1." The Timing chart Table 12 in the 8207 Data Sheet is then used to determine which clock edge to reference all timings from. The Waveforms diagrams then are used to determine the delay from the clock edge.

CHAPTER 3
RAM INTERFACE

The 8207 takes the memory addresses from the microprocessor bus and multiplexes them into row and column addresses as required by dynamic RAMs. The only hardware requirement when interfacing the 8207 to dynamic RAM are series resistors on all the RAM outputs of the 8207, and proper layout of the traces (see Intel's RAM Data Sheets or the Memory Design Handbook). This section mainly details the effects and requirements of input signals to the 8207 on the RAM array.

The 8207 contains an internal address counter used for refresh and error scrubbing (when using the 8206 EDCU) cycles. The 8207 has 18 address inputs (AIL0-AIL8 and AIH0-AIH8) which are multiplexed to form 9 address outputs (A00-A08). There are also 2 bank select (BS0, BS1) inputs for up to 4 banks of RAM. The Bank Select inputs are decoded internally to generate $\overline{RAS}$ and $\overline{CAS}$ outputs.

Refresh Interval

The 8207 supports four different refresh techniques as described in the *Refresh Options* section. In addition, the rate at which refresh cycles are performed is programmable. This is necessary because the refresh period is generated from the CLK input, which may vary over a wide range of frequencies. Programming the Cycle Fast/Slow (CFS) and Frequency Fast/Slow (FFS) bits automatically reprograms the refresh timer to generate the correct refresh interval for a clock frequency of 16, 10, 8, or 5 MHz (CFS, FFS = 11, 10, 01, and 00, respectively). For clock frequencies between those, Count Interval (CI1, CI0) programming bits allow "fine tuning" of the refresh interval. Refresh will always be done often enough to satisfy the RAM's requirements without doing refresh more often than needed and wasting memory bandwidth for all clock frequencies.

Refresh Counter

The internal refresh address counter of the 8207 contains 20 bits as organized in Figure 1.

19 18	17 16 15 14 13 12 11 10 9	8 7 6 5 4 3 2 1 0
Bank	Col addr	Row addr

Figure 1. 8207 Refresh Address Counter

In non-ECC mode, the refresh address counter does not count beyond bit 8. For standard RAMs, this will refresh 128 rows every 2 ms or 256 rows every 4 ms.

In ECC mode, the 8207 automatically checks the RAM for errors during refresh. This requires it to access each of the possible 2^{20} words of memory. The 8207 does not delete any of these bits when used with 16k and 64k dynamic RAMs. Each column would be scrubbed 4 times with 16k RAMs, and twice with 64 RAMs. This will have no detrimental effect on reliability. Banks of RAM that are not occupied, as indicated to the 8207 by the RB0, RB1 programming bits, will not be scrubbed.

Bank Selects BS0, BS1; RB0, RB1

The 8207 is designed to drive up to 88 RAMs in various configurations. The 8207 takes 2 inputs, BS0, BS1, and decodes them based on 2 programming bits, RB0, RB1, to generate the required $\overline{RAS}/\overline{CAS}$ strobes. Additionally, the 8207 will always recognize (not programmable) whether an access is made to the same RAM bank or to a different bank. The 8207 will interleave the accesses resulting in improved performance.

RAS and CAS Reallocation

The 8207's address lines are designed to drive up to 88 RAMs directly (through impedance matching resistors). The 4 $\overline{RAS}$ and $\overline{CAS}$ outputs drive up to 22 RAMs per bank (16 data plus 6 check bits with the 8206). Under these conditions, the 8207 will meet all RAM timing requirements. See Figure 2 for an example.

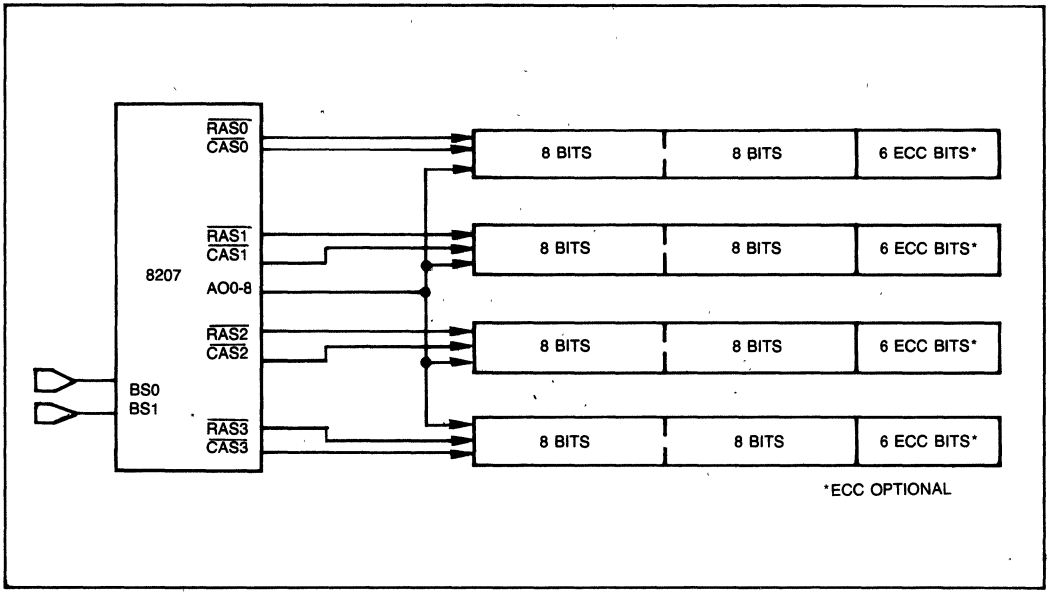


Figure 2. 8207 4 RAM Bank Configuration

The 8207 can accommodate other configurations like a 32 bit error corrected memory system. Each bank would have 39 RAMs (32 + 7 check bits) with the total number of RAMs equal to 78. This is within the address drivers capability, but the 39 RAMs per bank exceeds the $\overline{RAS}$ and $\overline{CAS}$ drivers limits. The loading of the $\overline{RAS}/\overline{CAS}$ drivers should not exceed 22 RAMs per bank, otherwise critical row, column address setup, and hold times would be violated.

In order to prevent these critical timings being violated, the 8207 will re-allocate the $\overline{RAS}$ and $\overline{CAS}$ drivers based on the RB0, RB1 programming bits (see Table 4). If the RB0, RB1 bits are programmed for 2 banks, the 8207 will operate $\overline{RAS0}$ and $\overline{RAS1}$ as a pair along with $\overline{RAS2}$ and $\overline{RAS3}$, $\overline{CAS0}$ and $\overline{CAS1}$, and $\overline{CAS2}$ and $\overline{CAS3}$. Now the address drivers would be loaded by 78 RAMs and the $\overline{RAS}/\overline{CAS}$ drivers by 20 RAMs. This relative loading is almost identical to the first case of four banks of 22 RAMs each. Drive reallocation allows a wide range of memory configurations to be used and still maintain optimal memory timings. Figure 3 shows a 32 bit non-error corrected configuration.

These programming bits do not help to qualify RAM cycles. Their purpose is to reallocate $\overline{RAS}/\overline{CAS}$ drivers. For example, if there is one bank of RAM and the bank select inputs (BS0, BS1) select any other bank and no provision is made to deselect the 8207 (via PE), the 8207 will do a RAM cycle and issue an acknowledge. This happens irregardless of the RB0, RB1 programmed value. See the *Optional RAM Bank's* section to provide for this.

Table 4. RAM Bank Selection Decoding and Word Expansion

Program Bits RB1	RB0	Bank Input B1	B0	$\overline{RAS}/\overline{CAS}$ Pair Allocation
0	0	0	0	$\overline{RAS}_{0-3}$, $\overline{CAS}_{0-3}$ to Bank 0
0	0	0	1	Illegal Bank Input
0	0	1	0	Illegal Bank Input
0	0	1	1	Illegal Bank Input
0	1	0	0	$\overline{RAS}_{0,1}$, $\overline{CAS}_{0,1}$ to Bank 0
0	1	0	1	$\overline{RAS}_{2,3}$, $\overline{CAS}_{2,3}$ to Bank 1
0	1	1	0	Illegal Bank Input
0	1	1	1	Illegal Bank Input
1	0	0	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	0	0	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1
1	0	1	0	$\overline{RAS}_2$, $\overline{CAS}_2$ to Bank 2
1	0	1	1	Illegal Bank Input
1	1	0	0	$\overline{RAS}_0$, $\overline{CAS}_0$ to Bank 0
1	1	0	1	$\overline{RAS}_1$, $\overline{CAS}_1$ to Bank 1
1	1	1	0	$\overline{RAS}_2$, $\overline{CAS}_2$ to Bank 2
1	1	1	1	$\overline{RAS}_3$, $\overline{CAS}_3$ to Bank 3

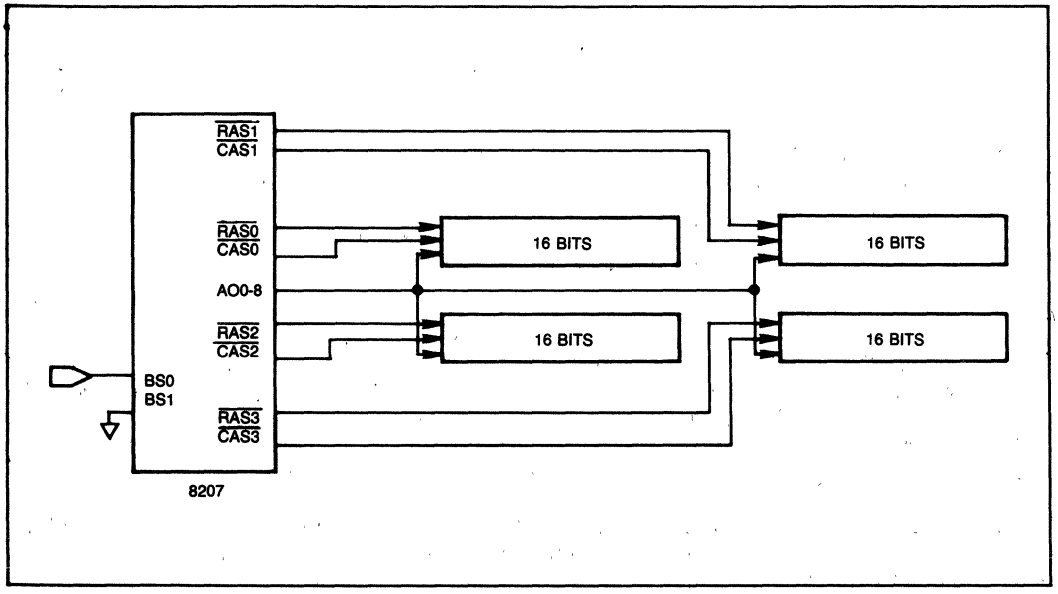


Figure 3. 8207 2 RAM Bank Configuration

Scrubbing

An additional function of the RB0, RB1 bits, besides $\overline{RAS}/\overline{CAS}$ allocation, is to inform the 8207 of how many banks are physically present. The 8207 will, during the refresh cycle, read data from a location and check to see that data and check bits are correct. If there is an error, the 8207 lengthens the refresh cycle and writes the corrected data back into RAM. Scrubbing the entire memory greatly reduces the chance of an uncorrectable error occurring. See the ***Refresh*** section for more detail on scrubbbing.

Refresh Cycles

The 8207 performs $\overline{RAS}$ only refresh cycles in non-ECC systems. It outputs all 8207 control signals except for $\overline{CAS}$ and acknowledges. The real delay in a system due to refresh would be a fraction of that value[1]. The length of the refresh cycle is always 2tRP + tRAS, and varies based upon the programmed 8207 configuration.

In error-corrected systems, the refresh cycle is actually a read cycle. The 8207 outputs a row address, then all $\overline{RAS}$ outputs go active. Next, a column address is output and then $\overline{CAS}$. The $\overline{CAS}$ output is based upon the RB0, RB1 allocation bits. Figure 4a shows the general timing for a four bank system, and Figure 4b shows a two bank system.

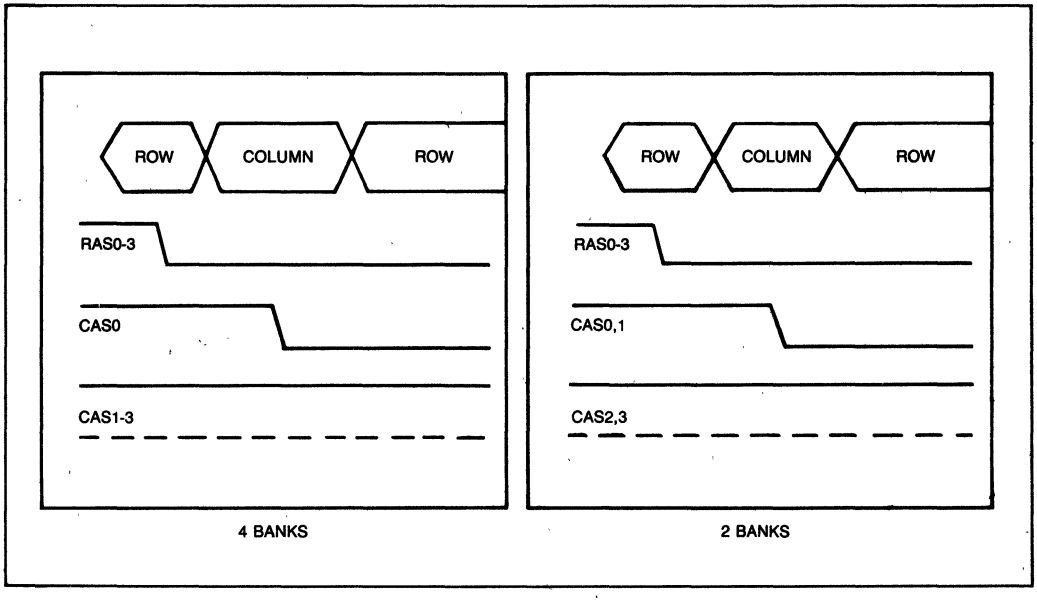


Figure 4. Refresh Cycles for Error Corrected Systems

(1) Measurements have shown a delay of 2-4% on program execution time compared to programs running without refresh.

The 8207 sends the read out word through the 8206 EDCU to check for any errors. If no errors, the refresh cycle ends. If an error is discovered, the 8207 lengthens the cycle. An error is determined if the ERROR output of the 8206 is seen active at the same edge that the 8207 issues the R/W output. The cycle is then lengthened to a RMW cycle. If the error was correctable, the corrected data is written back to the location it was read from. But, if the data is uncorrectable, the cycle is still lengthened to a RMW, but no write pulse is issued. To aid in stabilizing the RAM output data and the Error flag, pullup resistors of 10k ohms on the data out lines are recommended.

Scrubbing removes soft errors that may accumulate until a double-bit error occurs, which would halt the system. Hard single-bit failures will not stop the system, but could slow it down. This is because read and refresh cycles lengthen to correct the data.

For large RAM arrays some form of error logging or diagnostics should be considered.

Interleaving

The term "interleaving" is often used to refer to overlapping the cycle times of multiple banks (or boards or systems) of RAMs. This has the advantage of using relatively slow cycle time banks to achieve a faster perceived cycle time at the processing unit. The drawbacks of interleaving are more logic to handle the necessary control and, for maximum performance, the program should execute sequentially through the addresses.

Dynamic RAM cycles consist of 2 parts — the RAS active time (tRAS in Dynamic RAM Data Sheets) and precharge time (tRP). The sum of these two times are roughly equal to the cycle time of the RAM. The 8207 determines how long these two periods are, based on the configuration the user picked (via the programming bits). Bank interleaving, as used by the 8207, is slightly different than the previous definition. The 8207 will overlap the precharge time of one bank with the access time of another bank. In either case, the advantage is the effective cycle time is reduced without having to use faster RAMs.

For interleaving to take place there must be more than 1 bank of RAM connected to the 8207. Interleaving is not practical with 3 banks of RAM because 3 is not a power of 2 (the 2 bank inputs BS0, BS1). So, interleaving works only for 2 or 4 banks of RAM. Note that it is easy enough to use three banks of RAM where the bank select inputs are connected to the highest-order address line. For instance, if three banks of 2164s are used in an 8086 system, and located at address OH, bank selects BS0 and BS1 would be connected to microprocessor addresses A17 and A18, respectively. Banks 0-2 would be accessed in the address ranges OH - FFFFH, 10000H - 1FFFFH, and 20000H - 2FFFFH, respectively. In this case, consecutive addresses are almost always in the same bank and very little interleaving can take place.

Figure 5 shows the effects on the performance of the processor with and without interleaving. In both examples, consecutive accesses to the same bank will add 1 wait state to the second access, but no wait states to consecutive accesses to different banks. Irregardless of the 8207 configuration, there will always be a minimum 1 wait state added without interleaving. Therefore, interleaving is very highly recommended!

Interleaving is accomplished by connecting the 8207's BS0, BS1 inputs to the microprocessor's low order word address lines. Each consecutive address is then located in a different bank of RAM. About 90% of memory accesses are sequential, so interleaving will occur about 90% of the time in a single port system.

In a dual port system, the advantages of interleaving are a function of the number of banks of memory. Since the memory accesses of the two ports are presumably independent, and both ports are continuously accessing memory, the 8207 arbiter will tend to interleave accesses from each port (i.e., Port A, Port

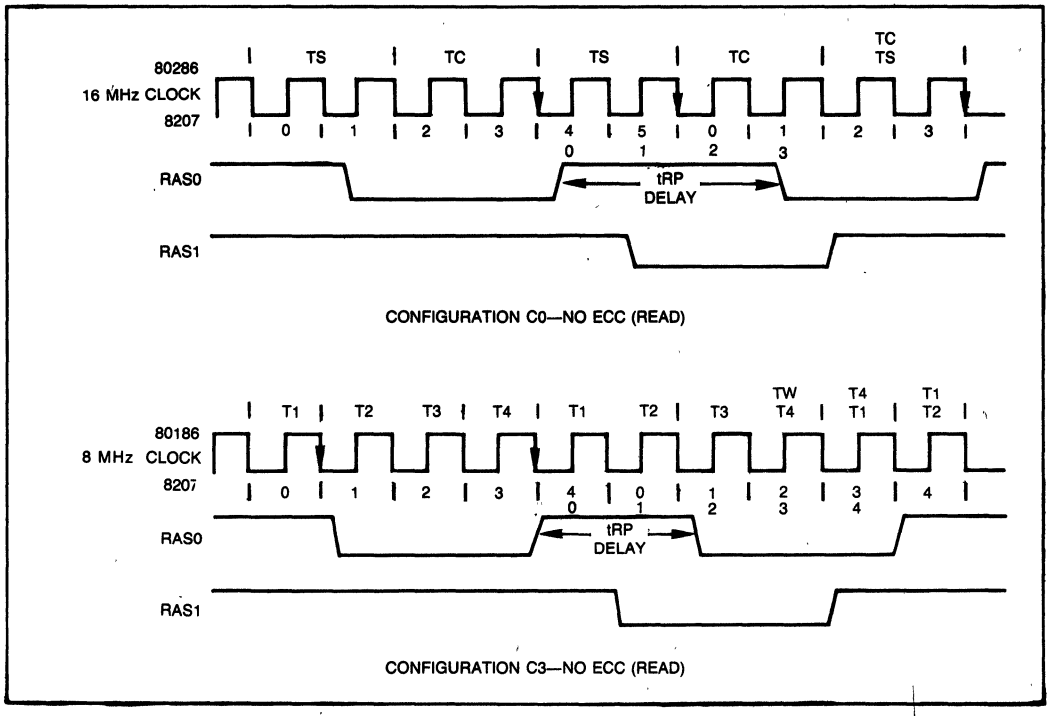


Figure 5. Processor Performance With and Without Interleaving

B, Port A, Port B, ...). If there are two banks of RAM interleaving will occur 50% of the time and, if there are four banks of RAM, interleaving will take place 75% of the time[1]. To the extent that a single port generates a majority of memory cycles, interleaving efficiency will approach 90% as described in the previous paragraph.

(1) Don't get confused here. The paragraph is talking about interleaving memory requests from both ports, and their probability of accessing one of the other banks of RAM where tRP has been satisfied. The 8207 will leave the RAM precharge time out if consecutive accesses go to different banks. The 8207 RAM timing logic does not care which port requests a RAM cycle. requests a RAM cycle.

Optional RAM Banks

Many users allow various RAM array sizes for customer options and future growth. Some care must be taken during the design to allow for this. Three items should be considered to permit optional RAM banks.

The first item is the total RAM size. The 8207 starts a memory cycle based only upon a valid status or command and $\overline{PE}$ active. So some logic will be required to deselect the 8207 (via $\overline{PE}$) when the addressed location does not exist within the current memory size. A 7485 type magnitude comparator works well.

The second item to consider is the BS0, BS1 inputs. With one bank of RAM these inputs are tied to ground. Four banks of RAM require two address inputs. So, if the design ever needs four banks

of RAM, then the BS0, BS1 inputs must be connected to address lines. Selecting a non-existant RAM bank is illegal. Figure 6 shows a non-interleaved method.

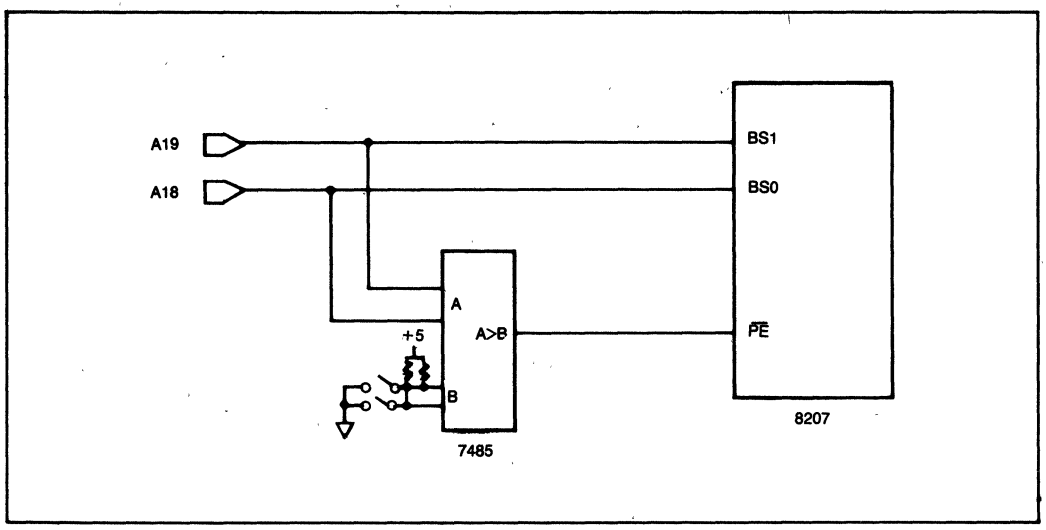


Figure 6. Non-Interleaved 8207 Selection Circuit

With designs using interleaving, the least significant word address lines are connected to the BS0, BS1 inputs. With two banks of RAM, A1 from the Intel processor is connected to BS0. A2 is connected to BS1, but not allowed to function until four banks are present. However, A2 must still be used since addresses increase sequentially. Two possible ways of implementing this are shown in Figure 7 below.

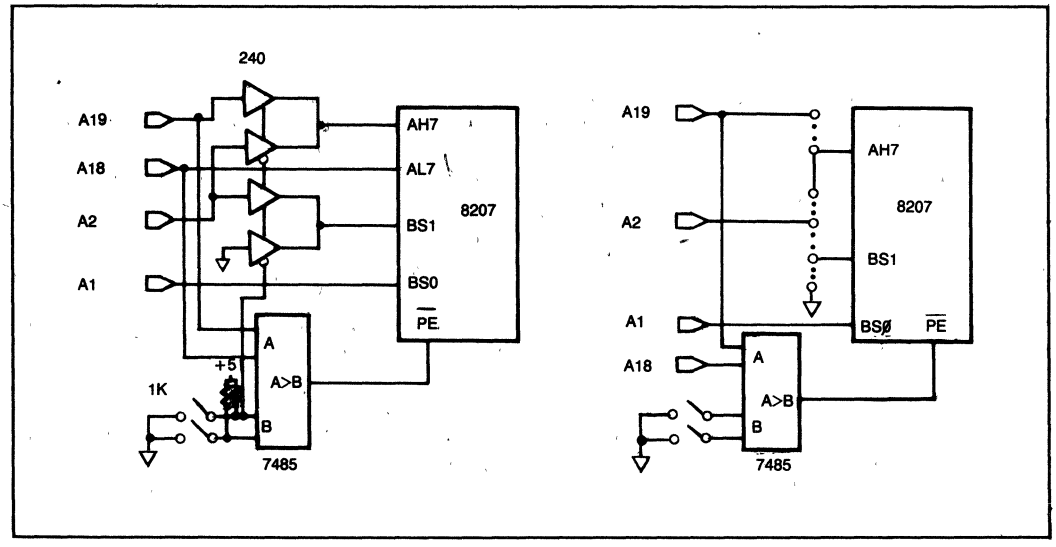


Figure 7. Interleaved 8207 Selection Circuits

The final consideration is for the $\overline{RAS}/\overline{CAS}$ outputs. Remember that when the RB0, RB1 bits are programmed for two banks, then $\overline{RAS0}$, 1 operates in tandem (non-ECC mode/ECC mode - the $\overline{CAS}$ outputs also work in tandem). Figure 8 shows the proper layout.

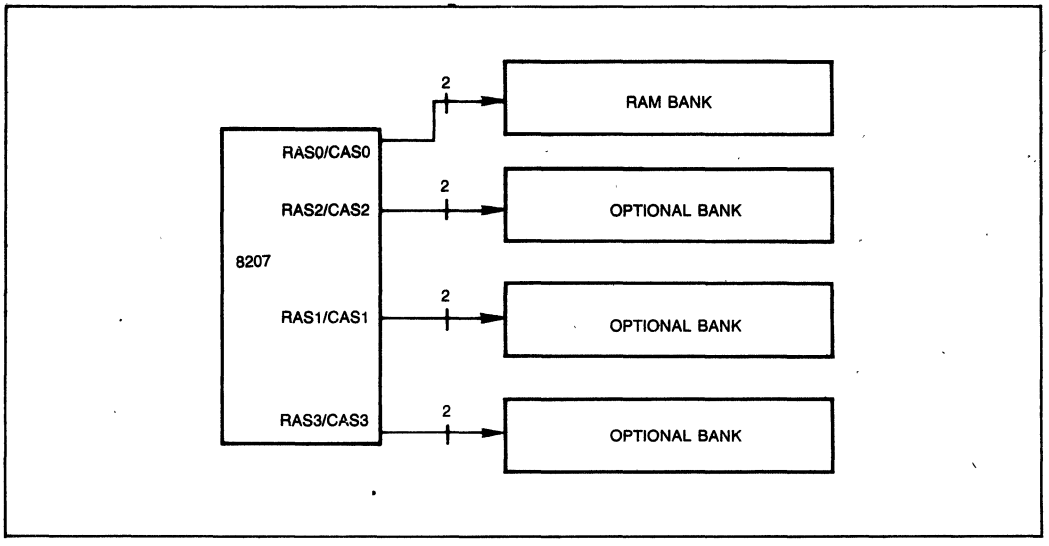


Figure 8. RAM Bank Layout

Write Enables - Byte Marks

The write enable supplied by the 8207 cannot drive the RAM array directly. It is intended to be NAND with the processor supplied byte marks in a non-ECC system. In error-corrected systems, the write enable output should be inverted before being used by RAMs. Only full word read/writes are allowed in ECC systems. The changing of byte data occurs in the 8206 EDCU.

For single and dual port systems, the byte mark data (A0, $\overline{BHE}$) must be latched. The 8207 can (and will) change the input addresses midway through a RAM cycle.

Memory Warm-up and Initialization

After programming, the 8207 performs 8 RAM warm-up cycles. The warm-up cycles are to prepare the RAMs for proper operation. If the 8207 is configured for ECC, it will then prewrite zeros into the entire array.

All RAS outputs are driven active for these cycles, once every 32 clock periods. The prewrite cycles are equivalent to write cycles, except all RAS and CAS will go active, data is generated by the 8206, and the address is generated by the 8207.

RAM Cycles/Timings

Tables 12 and 13 of the 8207 Data Sheet show on what clock edge each of the 8207 outputs are generated. This, together with the timing waveforms and A.C. parameters, allows the user to calculate the timings of the 8207 for each of its configurations. To make the job easier, Tables 14-18 of the 8207 Data Sheet precalculate dynamic RAM timings for each 8207 configuration and type of cycle. All that is required is to plug in numerical values for the 8207 parameters.

Write Cycles

The 8207 always issues WE after $\overline{CAS}$ has gone valid. These types of cycles are known as "late writes." The 8207 does this primarily to interface to the iAPX286 processor bus timings. Late writes require separate data in and data out traces to the RAM array, plus the additional drivers.

Data Latches

The 8207 is designed to meet data setup and hold times for the iAPX86 family processors when using a synchronous status interface (see Microprocessor Interface section). Other types of interfaces will require external data latches. This is because the $\overline{CAS}$ pulse is a fixed length - the user has no control (besides programming options) over lengthening $\overline{CAS}$. When $\overline{CAS}$ goes inactive, data out of the RAMs will disappear. Asynchronous interfaces should use $\overline{XACK}$ or $\overline{LAACK}$ to latch the data.

CHAPTER 4
MICROPROCESSOR INTERFACES

The 8207 is designed to be directly compatible with all Intel iAPX86, 186, 188, and 286 processors. For maximum performance, the 8207 will directly decode the status lines and operate off of the processor's clock. Additionally, the 8207 interfaces easily to other bus types that support demultiplexed address and data with separate read and write strobes.

Bus Interfaces

The 8207 easily supports either an asynchronous or synchronous command timing. The command timing can also be adjusted for various processors via the PCTL pin.

MEMORY COMMANDS

There are four inputs for each port of the 8207 that initiate a memory cycle. The input pins are $\overline{WR}$, $\overline{RD}$, PCTL, and $\overline{PE}$. The first three inputs connect directly to the iAPX 86, 88, 186, 188 $\overline{S0}$-$\overline{S2}$ outputs, respectively. For the 80286, the same connections are used except that PCTL is tied to ground. In all configurations $\overline{PE}$ is decoded from the address bus. Multibus type commands use the same input setup as the 80286.

COMMAND/STATUS INTERFACE

The status interface for the 80186 and the 80286 differ both in timing and meaning. The 8207 can be optimized for either processor by programming the PCTL input pin at RESET time. $\overline{S2}$ in 80186 systems, connects directly to PCTL. When the processor is reset it drives $\overline{S2}$ high for one clock, then tristates it. A pullup resistor to +5 will program the PCTL input for the 80186 status interface when RESET goes inactive. A pullup is required only if no component has this pullup internally.

To optimize the 8207 for the 80286 interface, PCTL is tied to ground and not used in 80286 systems. Multibus commands are similar in meaning to the 80286 status interface, and are programmed the same way. In Multibus type systems, PCTL can be used as an inhibit to allow shadow memory. PCTL would be driven high, when required, to prevent the 8207 from performing a memory cycle. It would be connected to the Multibus INH pin through an inverter.

SYNCHRONOUS/ASYNCHRONOUS COMMANDS

Each port of the 8207 can be configured to accept either a synchronous or asynchronous (via programming bits) memory request. Minimum memory request decode time (and maximum performance) is achieved using a synchronous status interface. This type of interface to the processor requires no logic for the user to implement.

An asynchronous interface is used with Multibus bus interfaces when the setup and hold times of the memory commands cannot be guaranteed. Synchronizers are added to the inputs and will require up to two clocks for the 8207 to recognize the command. It should be obvious that better performance will result if the 8207's clock is run as fast as possible.

Figure 2 of the 8207 Data Sheet shows various combinations of interfaces. The additional logic for the asynchronous interfaces is used to either lengthen the command width, to meet the minimum 8207 spec, or to make sure the command does not arrive too soon before the address has stabilized.

PORT ENABLE

The $\overline{PE}$ inputs serve to qualify a memory request. A RAM cycle, once started, cannot be stopped. A RAM cycle starts if $\overline{PE}$ is seen active at the proper clock edge and a valid command is recognized. If $\overline{PE}$ is activated after a command has gone active and inactive, no cycle will start.

Types of logic that work well are 74138 and 7485. $\overline{PE}$ should be valid as much as possible before the command arrives because, as the address bus switches and settles, glitches on $\overline{PE}$ could either: disqualify a memory cycle; delay a memory cycle; or start a memory cycle when none should have. Refer to the Port Interface Waveforms in the Data Sheet. If Port Enable is not seen active by the next or same clock edge, no memory cycle will occur unless the command is removed and brought active again.

Back to Back Commands

Holding the $\overline{RD}$, $\overline{WR}$ inputs active will not generate continuous memory cycles. Memory commands must go inactive for at least one clock period before another memory request at that port will be considered valid. Holding the inputs active will not keep the other port from gaining access to the RAM. The only signal that can prevent the other port's gaining access to the RAM is LOCK.

Address Inputs (And LOCK)

Two pins control the address inputs on the 8207, MUX and LEN. Neither are used for single port 8086 based systems. MUX is used for dual port configurations, and LEN is used for single and dual port 80286 based systems. MUX is used to gate the proper ports addresses to the 8207. If the output is high, Port A is selected. If it is low, Port B is selected.

The cross coupled NAND gates, shown in the 8207 Data Sheet (Figure 3), are used to minimize contention when switching address buses. Use of a single inverter would have both outputs enabled simultaneously for a short period. The cross coupled hand gates allow only one output enabled.

MUX also allows the single LOCK input to be multiplexed between ports. Figure 9 shows how to multiplex the LOCK input for dual port systems. See the LOCK section for more information.

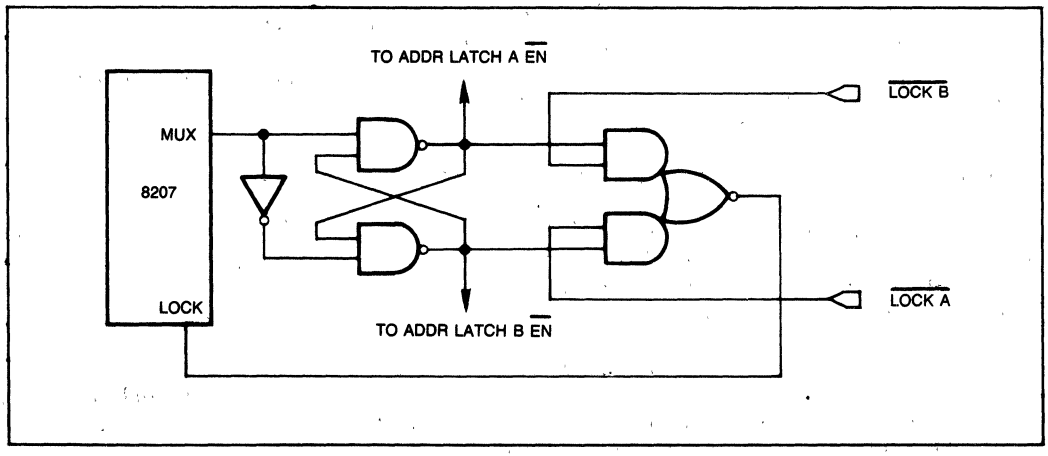


Figure 9. Dual Port LOCK Input Circuit

MUX TIMING

The MUX output is optimized by the Port Arbitration scheme, which is selected in the program word. Figure 10 shows the effects on memory selected in the program word. Figure 10 shows the effects on memory bandwidth with the different schemes. Port A Preferred optimizes consecutive cycles for Port A. Consecutive Port B cycles have at least 1 clock added to their cycle time. There would be no MUX delays for any Port A request.

The Most Recently Used scheme allows either port to generate consecutive cycles without any MUX delays. The first memory cycle for each port would have the 1 clock delay. But all others would not.

With either scheme, if both ports request the memory at their top speed, the 8207 will interleave the requests; Port A, Port B, Port A, Refresh, Port B.

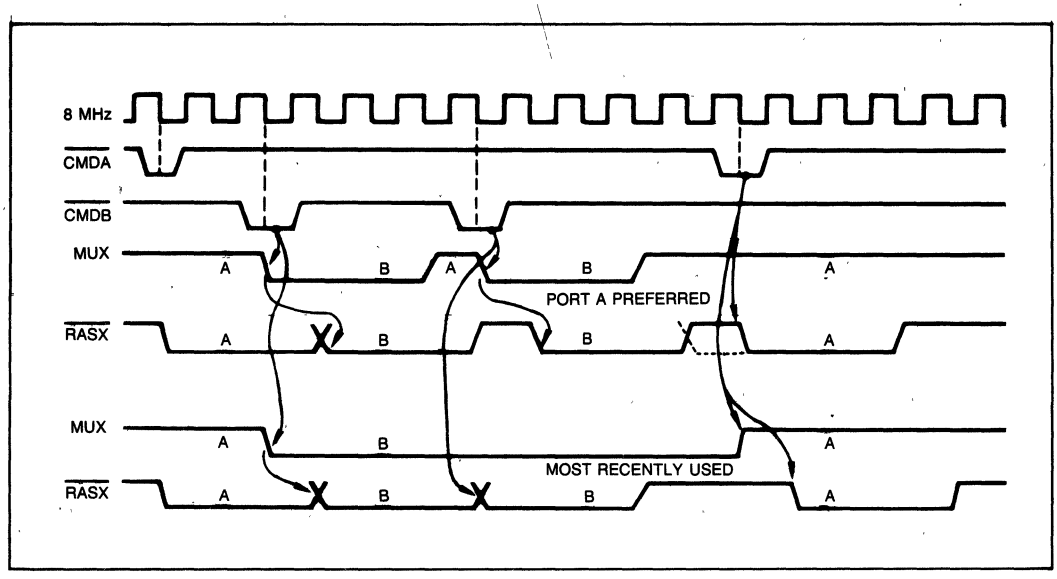


Figure 10. Port Arbitration Effects

LEN

LEN is used to hold the 80286 addresses when the 8207 cannot respond immediately. The 8207 will require a separate address latch, with the ALE input replaced with LEN. LEN optimizes the address setup and hold times for the 8207.

LEN goes from high to low when a valid 8207 command is recognized, which latches the 80286 address. This transition of LEN is independent of a memory cycle starting. The low to high transition will occur in the middle of a memory cycle so that the next address will be admitted and subsequently latched.

If Port B is to interface to an 80286 with the synchronous status interface, then LEN must be created using external logic. Figure 11 shows the equivalent 8207 circuit for Port B.

LOCK

The LOCK input allows each port uninterrupted access to memory. It does this by not permitting MUX to switch. It is not intended as a means to improve throughput of one of the ports. To do so is at the designer's risk[1]. Obviously, LOCK is only used in dual port systems. The 8207 interprets LOCK as originating from the port that MUX is indicating.

(1) The 8207 will not malfunction if this is done. This is a system level concern. For example, a time dependent process may fail if the other port holds LOCK active, preventing its access of memory and relinquishing the bus.

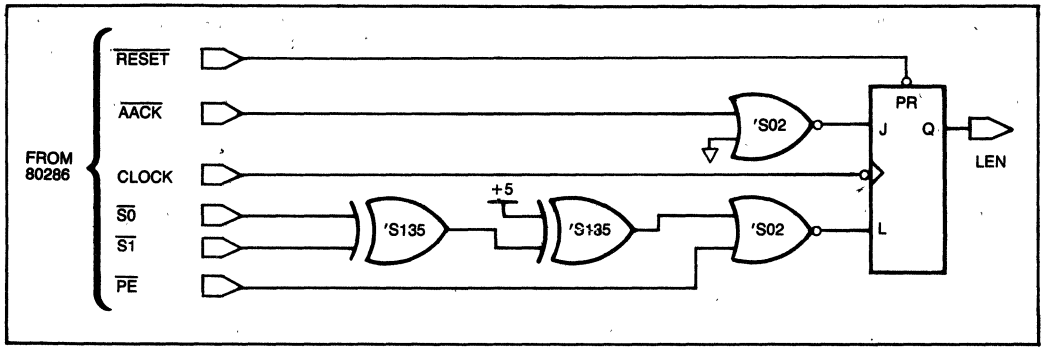


Figure 11. Port B LEN Circuit

LOCK from the 8086 may be connected directly to the 8207 or to the multiplexing logic. The 8207 requires additional logic when interfaced to an 80286. Figure 12 shows both the synchronous and asynchronous circuitry.

For 16 MHz operation, the 8207 ignores the LOCK input during the clock period that MUX switched. During 8 MHz operation, the 8207 will see LOCK as being active during the clock period when MUX switches.

The LOCK issued in Multibus bus systems may not be compatible with the 8207. The 8207 references LOCK from the beginning of a cycle, while Multibus references LOCK from the end of a cycle. The

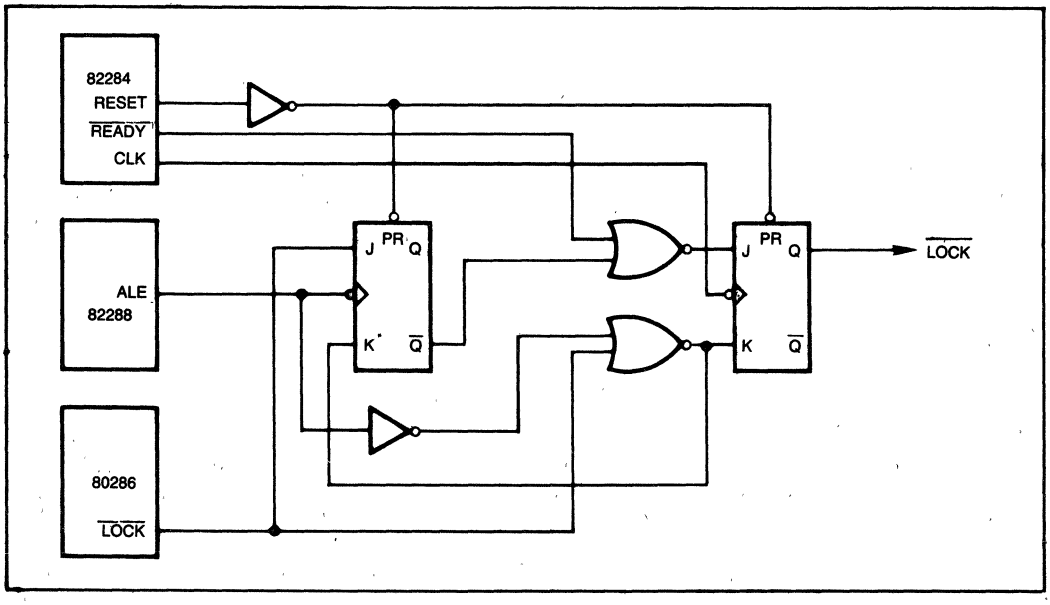


Figure 12a. Synchronous interface

230822-001

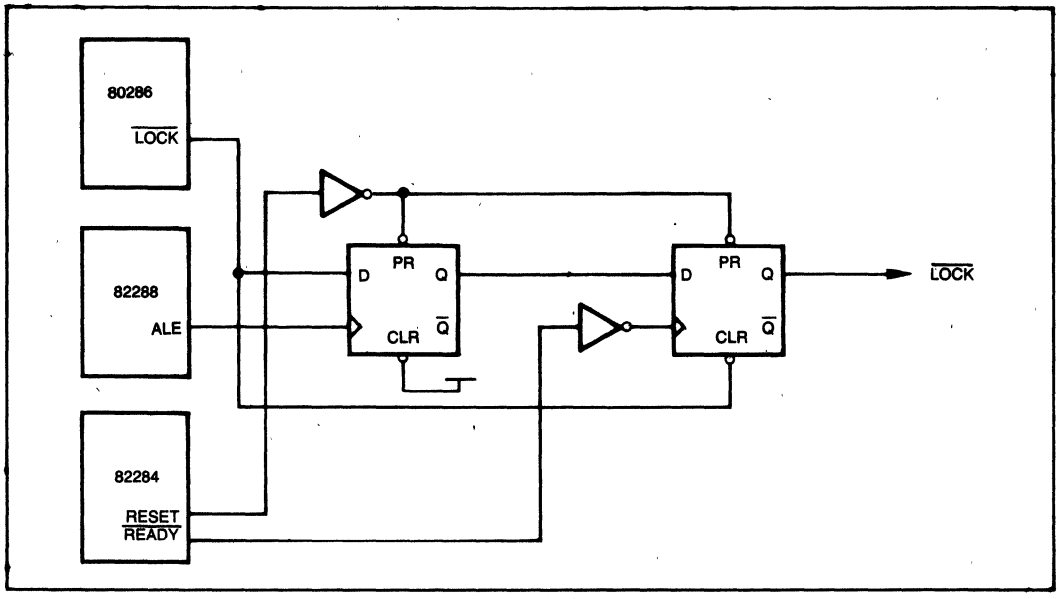


Figure 12b. Asynchronous interface

Multibus LOCK can be used if it meets the 8207 requirements. If the LOCK timing cannot be guaranteed, then additional logic is necessary. The logic would issue LOCK whenever a Multibus command is recognized. The drawback to this is that MUX cannot switch during the RAM cycle. This would delay the other port's memory access by one or two clocks.

DEADLOCK

The designer should ensure that a deadlock hazard has not been created in the design. The simple interfaces shown previously will not create a deadlock condition when the 8207 controls all system memory. If LOCK is issued by both ports, then the above logic would need to be modified to remove LOCK.

Figure 13 shows an illustration of the problem with a single LOCK input.

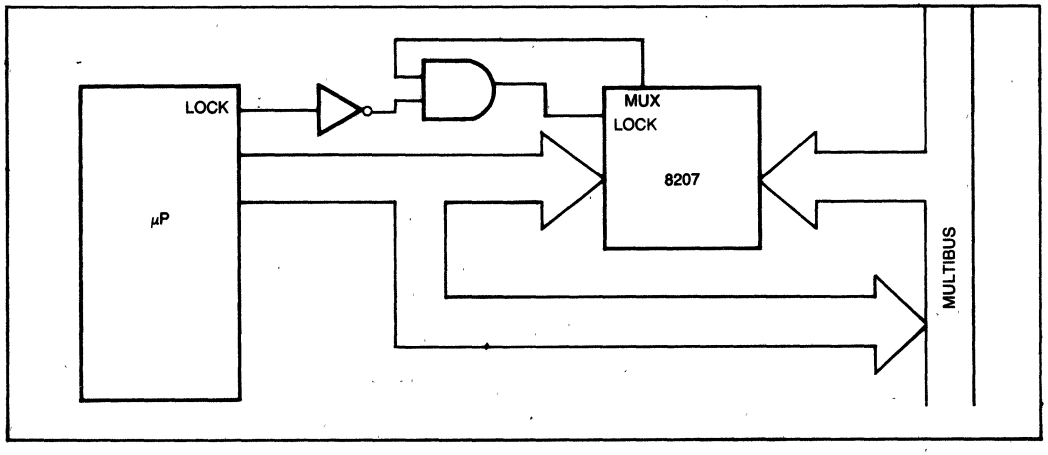


Figure 13. Single LOCK Input Circuit

Suppose the 8207 starts a locked string transfer for the processor. The Multibus bus port requests a memory cycle but must wait for the processor to remove LOCK. But the processor must access Multibus as part of the locked string transfer. We now have a deadlock. The solution is to force LOCK inactive whenever an access is made to non-8207 memory by the processor. By doing this we have now violated the purpose of LOCK, since the Multibus port could change data. Another solution is to ensure that locked data does not exist in physically separate memory.

8207 Acknowledge's

The 8207 in non-ECC mode has two active acknowledge's per port, $\overline{AACK}$ and $\overline{XACK}$. The $\overline{AACK}$ output is configured into either an "early" or "late" $\overline{AACK}$ based on the SA, SB bits in the program data word. In ECC systems there is one Acknowledge per port, and it is configured to any one of the three ($\overline{EAACK}$, $\overline{LAACK}$ or $\overline{XACK}$) by the programming bits.

The $\overline{AACK}$ pin is optimized for either the 80286 or the 8086, based upon the CFS programming bit (fast = 80286; slow = 8086). $\overline{XACK}$ conforms to the Multibus bus specification. $\overline{XACK}$ requires a tri-state buffer and must not drive the bus directly.

In synchronous systems, $\overline{XACK}$ will not go active if the memory command is removed prior to the clock period that issues $\overline{XACK}$. In asynchronous systems, the $\overline{AACK}$ pin can also serve as an advanced RAM cycle timing indicator.

Data out, in synchronous systems, should not have to be latched. The 8207 was designed to meet the data setup and hold times of Intel processors, the 8086 family, and the 80286. In asynchronous systems, the 8207 will remove data before the processor recognizes the Acknowledge ($\overline{LAACK}$ or $\overline{XACK}$). In these systems, the data should be latched with transparent type latches (Intel 8282/8283).

Output Data Control

Non-ECC

In single port systems, Intel processors supply the necessary timing signals to control the input or output of data to the RAMs. These control signals are $\overline{DEN}$ and DT/$\overline{R}$. Refer to the microprocessor handbook for their explanation. If these signals are not available, then PSEN and $\overline{DBM}$ provide the same function. They can be used directly to control the 8286/8287 bus drivers of the 8207.

Because of the single set of data in/out pins of the RAMs, data must be multiplexed between the two ports in dual port systems. The 8207 provides two outputs for contention-free switching. PSEL operates the same as the MUX output, in that a high selects Port A and a low selects Port B. PSEN acts to enable the selected port. The timing is shown in the 8207 Data Sheet, Port *Switching Timing* section.

The easiest means of using PSEL and PSEN is shown in Figure 14. At no time will both ports be enabled simultaneously.

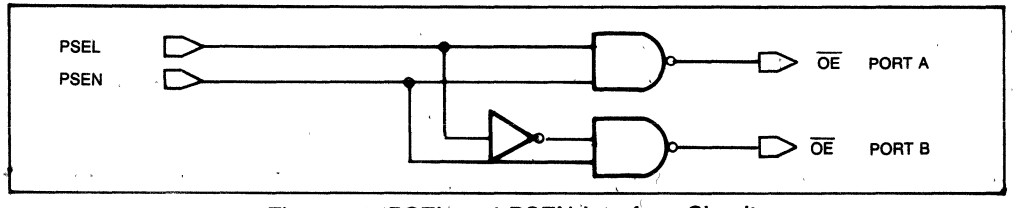


Figure 14. PSEL and PSEN Interface Circuit

Data Bus - Single Port

Recall that the 8207 always performs a late write cycle and that this requires separate data in and out buses. One option for the data bus is shown in Figure 3 of the 8207 Data Sheet. It requires separate data in and out traces on the processor board.

The second option is to keep the processor's combined data, bus but separate the data at the 8207 RAM. This is shown in Figure 15.

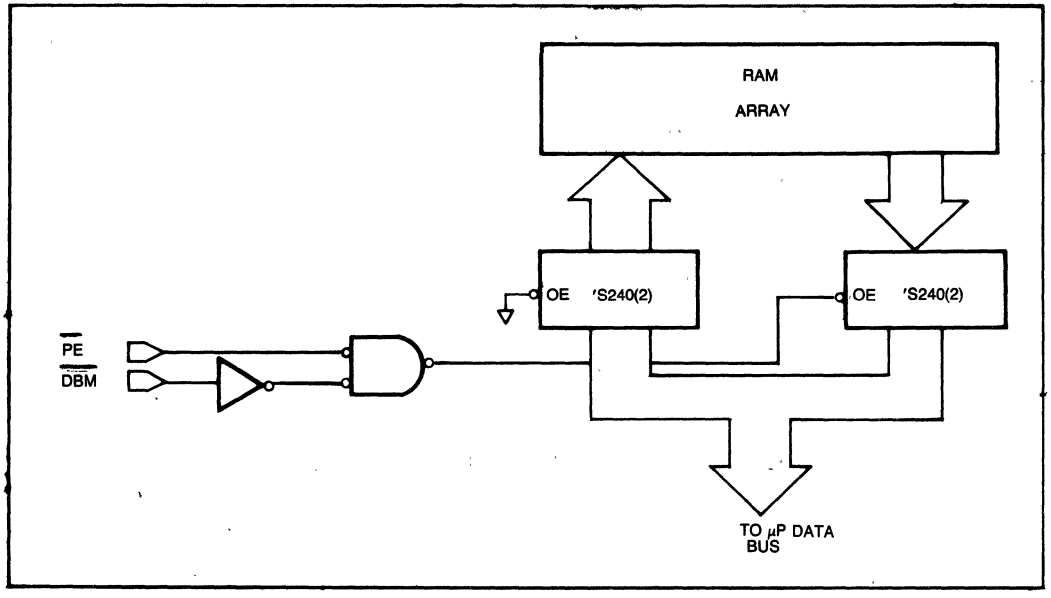


Figure 15. Data Bus Circuit

Data Bus - Dual Port

Non-ECC

The multiplexed data of the 8207 RAM must be kept isolated so that an access by one port does not affect another port. Figure 16 illustrates the control logic.

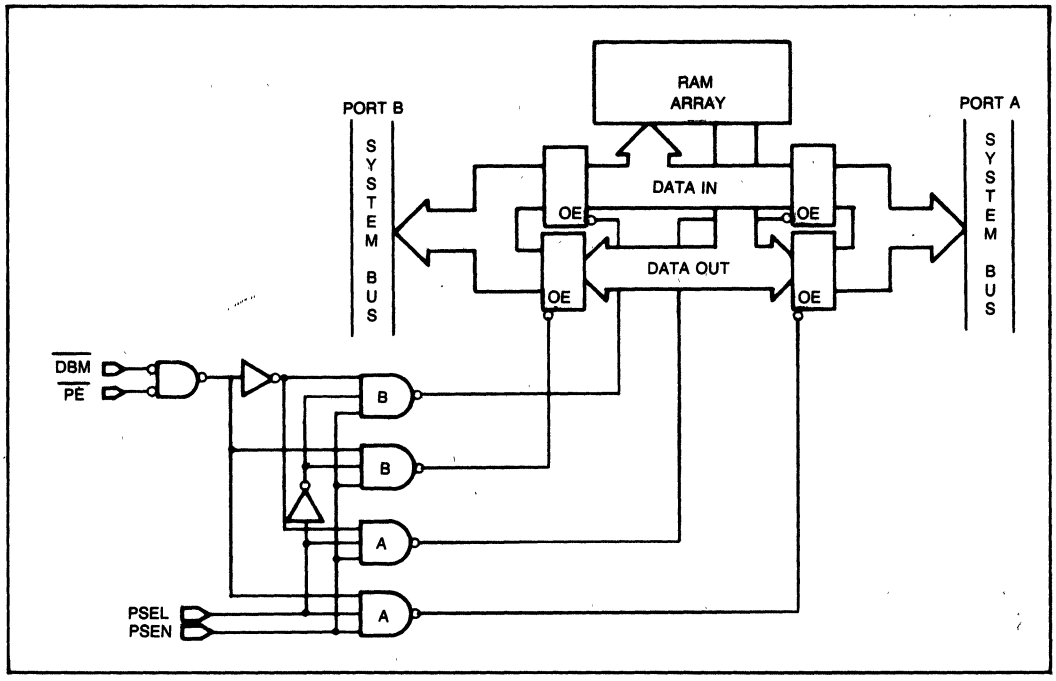


Figure 16. Dual Port Data Bus Control Circuitry

CHAPTER 5
8207 WITH ECC (8206)

This section points out the proper control of the 8206 EDCU by the 8207.

The 8207 performs error correction during read and refresh cycles (scrubbing), and initializes memory after power up to prevent false errors from causing interrupts to the processor. Since the 8207 must refresh RAM, performing scrubbing during refresh allows it to be accomplished without any additional performance penalty. Upon detection of a correctable error during scrubbing, the RAM refresh cycle is lengthened slightly to permit the 8206 to correct the error and for the corrected word to be rewritten into memory. Uncorrectable errors detected during scrubbing are ignored, since the processor may never access that memory location.

Correctable errors detected during a memory read cycle are corrected immediately and written back into memory.

Synchronous/Asynchronous Buses

The many types of configurations that are supported by the 8207/8206 combination can be broken down into two classes: ECC for synchronous or for asynchronous buses.

In synchronous bus systems, performance is optimized for processor throughput. In asynchronous buses, performance is optimized to get valid data onto the bus as quickly as possible (Multibus). While possible to optimize the 8207/8206 for processor throughput in Multibus systems, it is not Multibus compatible. The performance optimization is selected via the XA/XB and SA/SB programming bits.

When optimized for processor throughput, an advanced acknowledge ($\overline{\text{AACK}}$ - early or late) is issued at some point (based on the type of processor) so that data will be valid when the processor needs it.

When optimized for quick data access, an $\overline{\text{XACK}}$ is issued as soon as valid data is known to exist. If the data was invalid (based on the ERROR flag), then the $\overline{\text{XACK}}$ is delayed until the 8206 corrects the data and the data is on the bus.

The first example is known as "correct always" mode. The 8206 $\overline{\text{CRCT}}$ pin is tied to ground and the 8206 requires time to do the correction. Figure 17 shows this implementation. The quick data access method is known as "correct on error." The CRCT pin is tied to the R/$\overline{\text{W}}$ output of the 8207. When CRCT is high, the 8206 does not do correction, but still checks the data. This delay is typically half of the first. If an error happens, the cycle becomes a RMW and $\overline{\text{XACK}}$ is delayed slightly so that data can be corrected.

The correct on error mode is of no real benefit to non-Multibus users. The earliest acknowlege (EAACK) is delayed by one clock to allow for the delays through the 8206. This imposes a 1 wait state delay.

Byte Marks

The only real difference to the 8207 system when adding the 8206 is the treatment of byte writes. Because the encoded check bits apply only to a whole word (including check bits), byte writes must not be permitted at the RAM. Instead, the altering of byte data is done at the 8206. The byte marks previously sent to RAM are now sent to the 8206. These byte marks must also qualify the output enables of the data drivers.

The $\overline{\text{DBM}}$ output of the 8207 is meant to be nanded with the processors byte marks. This output is activated only on reads or refreshes. On write cycles, this output stays high which would force the 8206 byte mark input low. When low, the internal 8206 data out buffers are tristated so that new data may be gated into the device.

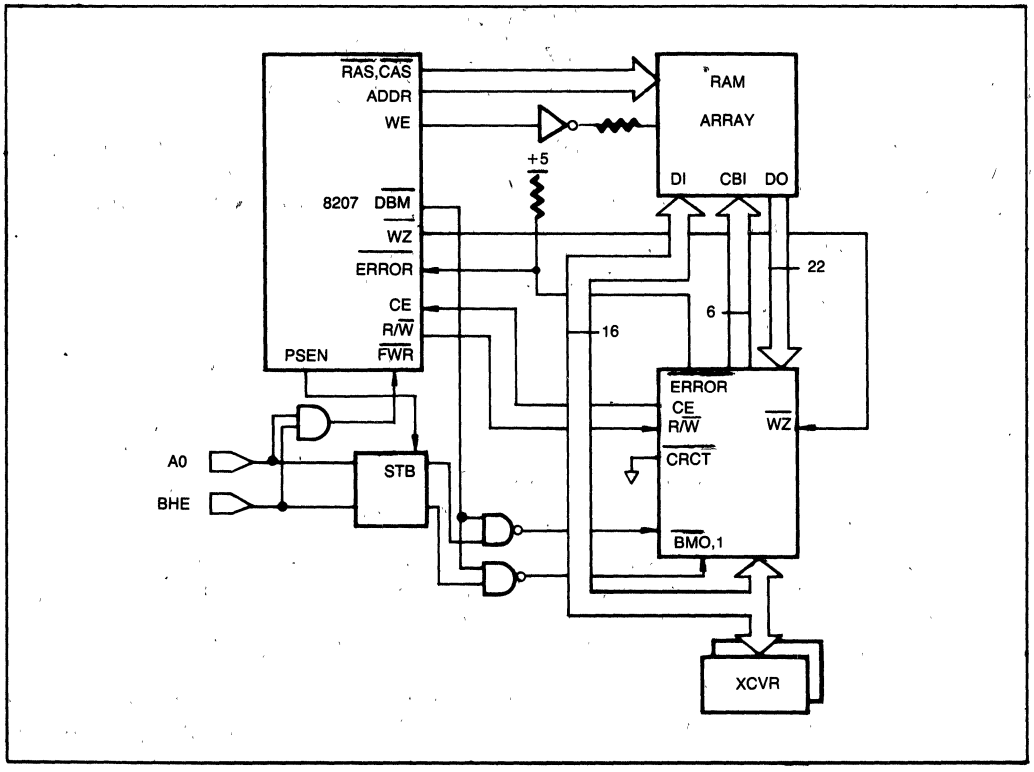


Figure 17. 8206 Interface to the 8207

Read Modify Writes - ECC

A RMW cycle occurs whenever a processor wants to do byte writes or when the 8207 has detected an error during read or refresh (scrubbing) cycles. A byte write is detected by the $\overline{FWR}$ input to the 8207 and is based on the processor supplied byte marks.

At the start of a RMW cycle, $\overline{DBM}$ stays high, which, when qualified with the byte marks, will enable the data out buffer of the 8206 for the unmodified byte, and tristates the buffer for the new byte; R/$\overline{W}$ is high, which tells the 8206 to do error detection and correcting (if $\overline{CRCT}$ is low). The 8206 can latch data and check bits from the RAM via the STB input, but the 8207 does not use this feature. Instead, the 8207 keeps $\overline{CAS}$ active the entire length of the RMW cycle to hold data at the 8206. The new byte data from the processor goes to the 8206 and to the RAM. The 8207 would have corrected any errors just read, so the old and new bytes of data, plus their check bits, are available at the RAM, and the 8207 generates a write pulse. The data driver for the unmodified byte must not have been enabled, otherwise erroneous data would be written to RAM and possibly made valid (if it was stable) by the 8206.

Data Buffer Control - ECC

The control of the data buffers is essentially the same as in non-ECC systems, with a few exceptions.

The processor's byte marks must now qualify the output enable logic. The reason was described earlier in the RMW section. This applies to both single and dual port configurations. A refresh cycle outputs all the control signals that a read cycle will, except for an acknowledge. If complete buffer control is left to the 8207, then it would occasionally (during refreshes) put data on the processor bus. The $\overline{DEN}$ and DT/$\overline{R}$ signals must be qualified by the $\overline{PE}$ input. $\overline{PE}$ would have to be latched for the entire cycle by PSEN.

Test Modes

Neither of the two test modes of the 8207 are to be used in a design. Both test modes reset the refresh address counter to a specific value, which interrupts the refresh sequence and causes loss of data.

In error corrected systems, a reset pulse causes the 8207/8206 to write over the entire RAM array. Test Mode 2 appears to bypass the prewrite sequence. But, the refresh counter is reset to a value of 1F7 (H). So, besides interrupting the refresh sequence, the 8207 still prewrites the 8 locations specified by the counter.

To not overwrite the RAM data, the 8207 RESET will have to be isolated from the system reset logic in ECC systems.

APPENDIX I

8207/8208 Performance

The following performance charts were based upon Figure 3 in the 8207 Data Sheet, and apply to the 8208 as well. All RAM access delays are based upon Intel dynamic RAMs. The charts show the performance of a single cycle with no precharge, refresh, port switching, or arbitration delays.

The read access calculations are: the margin between the 8207 starting a memory cycle to data valid at the processor - 8207 RAS or CAS from clock delay - DRAM RAS or CAS access - 8286 propagation delay - processor setup.

Assume the RAS/CAS drivers are loaded with 150 pf, and the 8286 is driving a 300 pf data bus.

80286 (example)

RAS Access: 3TCLCL - 8207 TCLRSL - 2118 tRAC - 8286 TIVOV - 80286 t8
= (3)62.5 - 35 max - 100 max - 22 - 10
= 20 ns

80186 (example)

CAS Access: 2 TCLCL - 8207 TCLCSL - 2164A tCAC - 8286 TIVOV - 80186 TDVCL
= (2)125 - 115 max - 85 max - 22 - 20
= 8 ns

8207 Performance (EDC synchronous status interface)

Table 5a. Wait States for Different μP and RAM Combinations

Wait states at full CPU speed		RAM speed			
CPU	**Freq**	**100 ns**	**120 ns**	**150 ns**	**200 ns**
80286	8 MHz	1-RD, WR 3-Byte WR CO (3)	1-RD, WR 3-Byte WR CO	2-Read 1-Write 3-Byte WR C2	Not (1) compatible with RAM parameters
80186, 8086/88-2	8 MHz	1-RD, WR 3-Byte WR C4	1-RD,WR 3-Byte WR C4	1-RD,WR 3-Byte WR C4	
8086/88	5 MHz	1 C6	1 C6	1 C6	1-RD, WR 3-Byte WR C4

8207 Performance (EDC synchronous status interface)

Table 5b. μP Clock Frequency for Differenc μP and RAM Combinations

Maximum frequency for one wait-state (4)		RAM speed			
CPU	**Freq**	**100 ns**	**120 ns**	**150 ns**	**200 ns**
80286	8 MHz	FULL SPEED		7.3 MHz C0	6 MHz C0
80186, 8086/88-2	8 MHz				7 MHz C4
8086/88	5 MHz				

8207 Performance (Non-EDC synchronous status interface)

Table 6a. Wait States for Different μP and RAM Combinations

Wait states at full CPU speed		RAM speed			
CPU	**Freq**	**100 ns**	**120 ns**	**150 ns**	**200 ns**
80286	8 MHz	0 CO(3)	1-Read 0-Write C1	1-Read 0-Write C1	Not(1) compatible with RAM parameters
80186, 8086/88-2	8 MHz	0 C3	0 C3	0(2) C3	
8086/88	5 MHz	0 C3	0 C3	0 C3	0 C3

Table 6b. μP Clock Frequency for Different μP and RAM Combinations

Maximum frequency for no wait-state (4)		RAM speed			
CPU	**Freq**	**100 ns**	**120 ns**	**150 ns**	**200 ns**
80286	8 MHz	FULL SPEED	7 MHz	6 MHz	5.3 MHz
80186, 8086/88-2	8 MHz				7 MHz
8086/88	5 MHz				

(1) The 2164A tRAH parameter is not satisfied.

(2) 150 ns 64K DRAMs with tCAC = 100 ns won't run with 0 wait-states, because they have a longer CAS access time than the 2164A-15 (tCAC = 85 ns).

(3) Numbers in lower right corners are the programmed configurations of the 8207.

(4) To meet read access time.

8207 Performance (multibus interface)

This is an *asynchronous, command interface*. Worst case data and transfer acknowledge (XACK#) delays. Including synchronization and data buffer delays, are:

Table 7a. Non-EDC system

	RAM speed			
	100 ns	**120 ns**	**150 ns**	**200 ns**
Data access time	289ns	299ns	322ns	380ns
XACK# access time	333ns			450ns

Table 7b. EDC system

	RAM speed			
	100 ns	**120 ns**	**150 ns**	**200 ns**
Data access time (read)	359ns (324 ns)[1]	369ns (334 ns)	392ns (357 ns)	450ns (415 ns)
XACK# access time	400 ns-RD, WR 588 ns-Byte Write			520 ns-RD, WR 806 ns-Byte WR

(1) Numbers in parentheses are for when 8206 is in check-only mode (8206 doesn't do error correction until after an error is detected.

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Intel International (U.K.) Ltd.
Piper's Way
Swindon, SN3 1RJ
Wiltshire, England

Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

Printed in U.S.A./C-634/120K/RRD/CG-MK
Microprocessor/Peripheral