intel

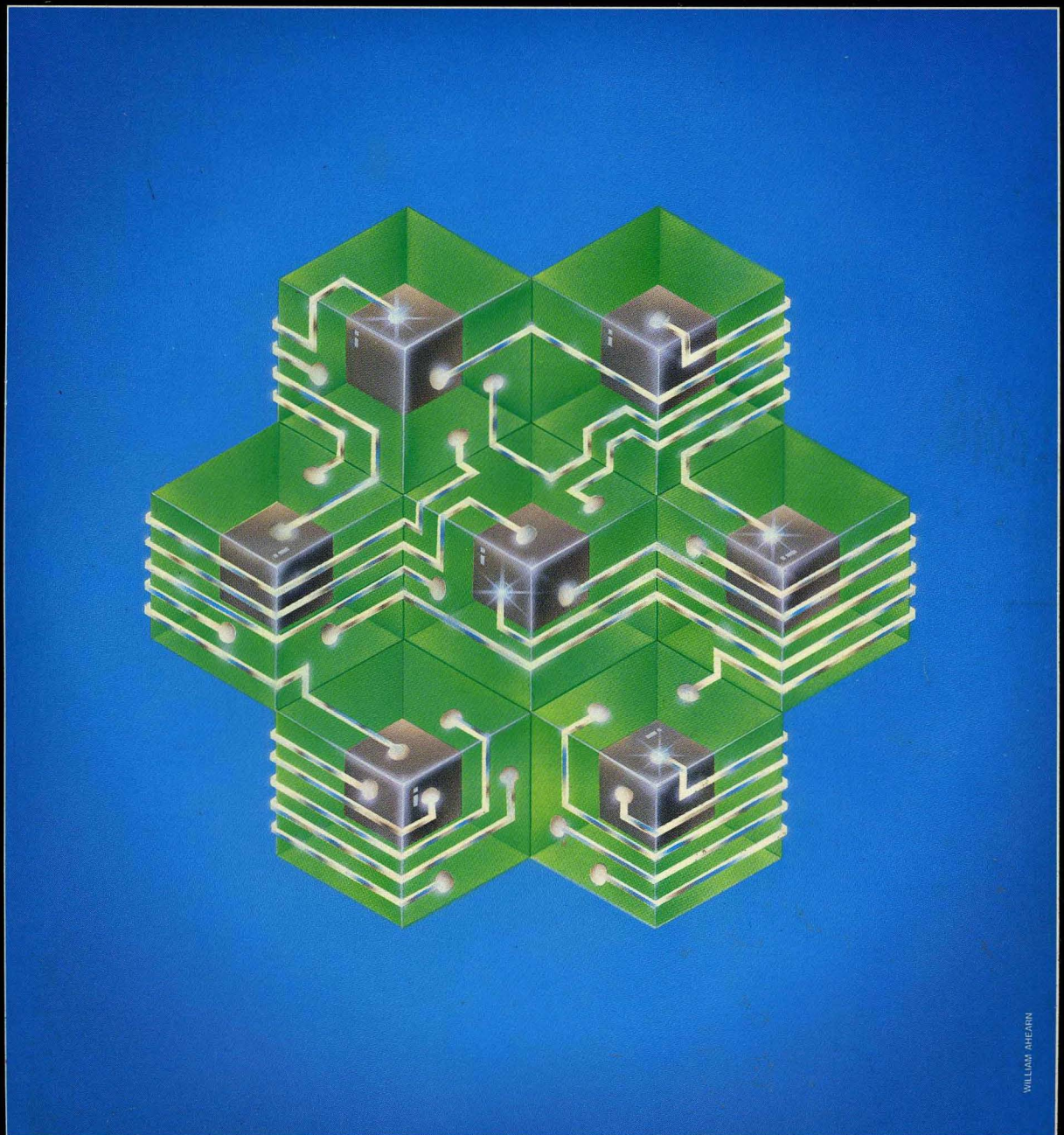
Microsystem Components Handbook, Volume I

1985

intel

Microsystem Components Handbook

Microprocessors and Peripherals Volume I



WILLIAM AHEARN

Order Number: 230843-002

LITERATURE

In addition to the product line handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846-003) provides an overview of Intel's complete product lines and customer services.

Consult the INTEL LITERATURE GUIDE (Order No. 210620) for a listing of Intel literature. TO ORDER literature in the U.S., 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).

Use the order blank on the facing page or call our TOLL FREE number listed above to order literature. Remember to add your local sales tax.

1985 HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information.

	*U.S. PRICE
QUALITY/RELIABILITY HANDBOOK (Order No. 210997-001) Contains technical details of both quality and reliability programs and principles.	**$15.00**
CHMOS HANDBOOK (Order No. 290005-001) Contains data sheets only on all microprocessor, peripheral, microcontroller and memory CHMOS components.	**$12.00**
MEMORY COMPONENTS HANDBOOK (Order No. 210830-004)	**$18.00**
TELECOMMUNICATION PRODUCTS HANDBOOK (Order No. 230730-003)	**$12.00**
MICROCONTROLLER HANDBOOK (Order No. 210918-003)	**$18.00**
MICROSYSTEM COMPONENTS HANDBOOK (Order No. 230843-002) Microprocessors and peripherals—2 Volume Set	**$25.00**
DEVELOPMENT SYSTEMS HANDBOOK (Order No. 210940-003)	**$15.00**
OEM SYSTEMS HANDBOOK (Order No. 210941-003)	**$18.00**
SOFTWARE HANDBOOK (Order No. 230786-002)	**$12.00**
MILITARY HANDBOOK (Order No. 210461-003) Not available until June.	**$15.00**
COMPLETE SET OF HANDBOOKS (Order No. 231003-002) Get a 25% discount off the retail price of $160.	**$120.00**

***U.S. Price Only**

intel®

U.S. LITERATURE ORDER FORM

NAME: _____ TITLE: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: (____) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____
☐☐☐☐☐☐-☐☐☐		____	× ____	= ____

POSTAGE AND HANDLING:
Add appropriate postage and handling to subtotal
10% U.S.
20% Canada

Subtotal _____

Your Local Sales Tax _____

_____

Allow 4–6 weeks for delivery

Total _____

Pay by Visa, MasterCard, Check or Money Order, payable to Intel Literature. Purchase Orders have a $50.00 minimum.

☐ Visa Account No. _____ Expiration Date _____
☐ MasterCard

Signature: _____

Mail To: Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.

Customers outside the U.S. and Canada should contact the local Intel Sales Office or Distributor listed in the back of this book.

For information on quantity discounts, call the 800 number below:
TOLL-FREE NUMBER: (800) 548-4725
Prices good until 12/31/85.
Source HB

Mail To: Intel Literature Distribution
Mail Stop SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051.

intel

MICROSYSTEM COMPONENTS HANDBOOK

1985

About Our Cover:
The design on our front cover is an abstract portrayal of microprocessors and associated peripherals as the building blocks which provide total systems development solutions. Intel superior technology and reliability provide easier solutions to specific development problems thereby cutting "time-to-market" and creating a greater market share.

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, $\hat{i}$, ICE, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMDDX, iMMX, Insite, Intel, int_el, int_elBOS, Intelevision, int_eligent Identifier, int_eligent Programming, Intellec, Intellink, iOSP, iPDS, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, OpeNET, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Ripplemode, RMX/80, RUPI, Seamless, SLD, SYSTEM 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, iSBX, 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 is a Trademark of Digital Research, Inc.

CHAPTER 5

MEMORY CONTROLLERS

— VOLUME 2 —

SUPPORT PERIPHERALS

FLOPPY DISK CONTROLLERS

HARD DISK CONTROLLERS

CHAPTER 7

Numeric Index

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. Highest reliability is a tangible goal 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 lower product costs.

HIGHER SYSTEM PERFORMANCE

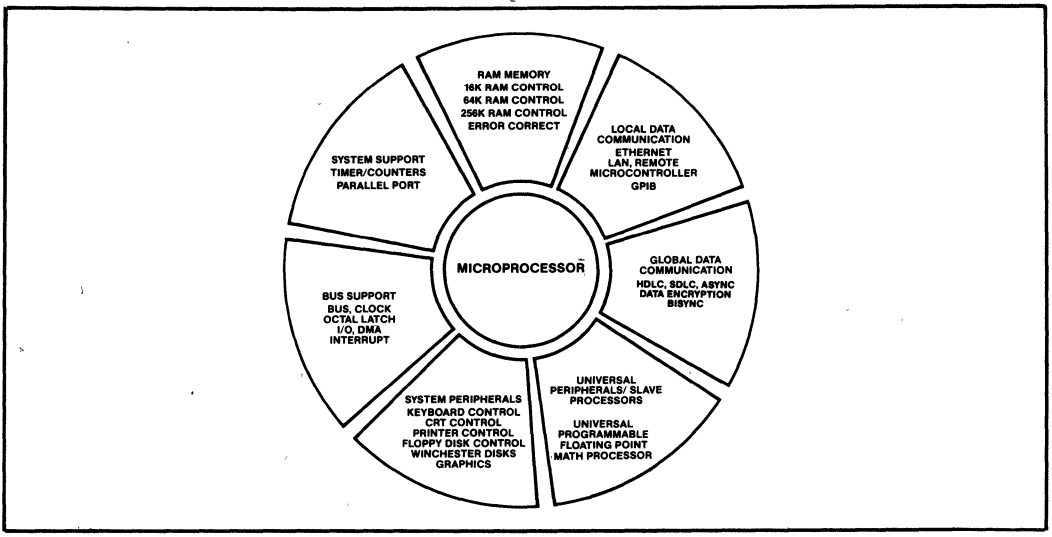
Intel microprocessors and peripherals provide the highest system performance for the demands of today's (and tomorrow's) microprocessor-based applications. For example, the iAPX 286 CPU, with its on-chip memory management and protection, offers the highest performance for multitasking, multiuser systems.

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 controllers, 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	82588	82586/501/502	8044/8744	8042/8742	8272A	82062/064	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	82588	82586/501/502	8044/8744	8042/8742	8272A	82062/064	8203	8206	8207	8208	8254	8255	8231A	
TERMINALS																												
Conversational			Y			X	X	X	X	X	X													Y	Y			
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		X		X	X	Y	Y		
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																Y	Y		
Printing, Laser, Impact	X	X	X	X			X	X	X							X	X	X							Y	Y		
Portable	X	X	Y				X	X	X										X		X				Y	Y		
INDUSTRIAL AUTO																												
Robotics			Y	Y	X							X		X			Y	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	Y	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				Y	X							X	X	X	
Aviation/Navig		X	X	X	X													X							X	X	X	
INDUST/DATA ACQ																												
Laboratory Instr	X	X	X	X	X								Y				Y	X								X	X	
Source Data	X		X														Y									X		
Auto Test	X	X	X	X	Y												Y									X		
Medical	X	X	X	X	X		Y									X	Y	X								X	X	
Test Instr	X	X	X	X	X							X		X		X	Y	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			X	X	X	X	X	X	X	
Financial Transfer		X		X			X	X	X	X					X		Y							X		X		
Automatic Teller		X	X	X		X	X	X	X	X							Y	X						X		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	Y	X	X	X	X	X	Y	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	Y	Y	X	X	X	X	X	Y		
MINI MAINFRAME																												
Processor & Control Store		X		Y	Y	Y						X		X								X	X					
Database Subsys		X		Y	X	X									X							X	X					
I/O Subsystem			Y	Y	Y							X		X			X						X					
Comm Subsystem		X		Y	Y				X			X		X	X		X						X					

MCS®-80/85 Microprocessors

2

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.

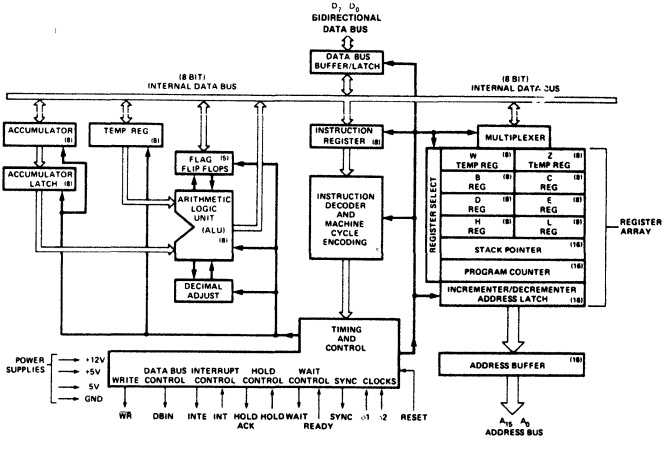


Figure 1. Block Diagram

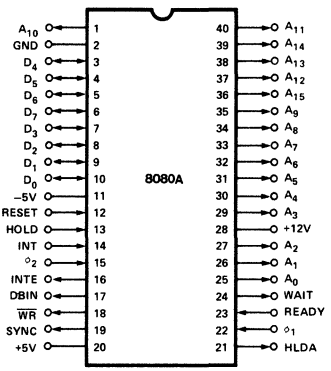


Figure 2. Pin Configuration

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.

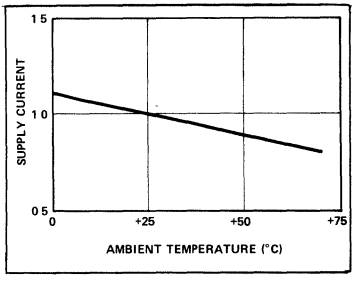
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{DD} = +12V ±5%, V_{CC} = +5V ±5%, V_{BB} = −5V ±5%, V_{SS} =0V; unless otherwise noted)

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.



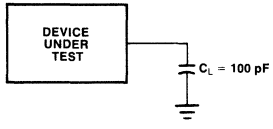
Typical Supply Current vs. Temperature, Normalized[3]

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)

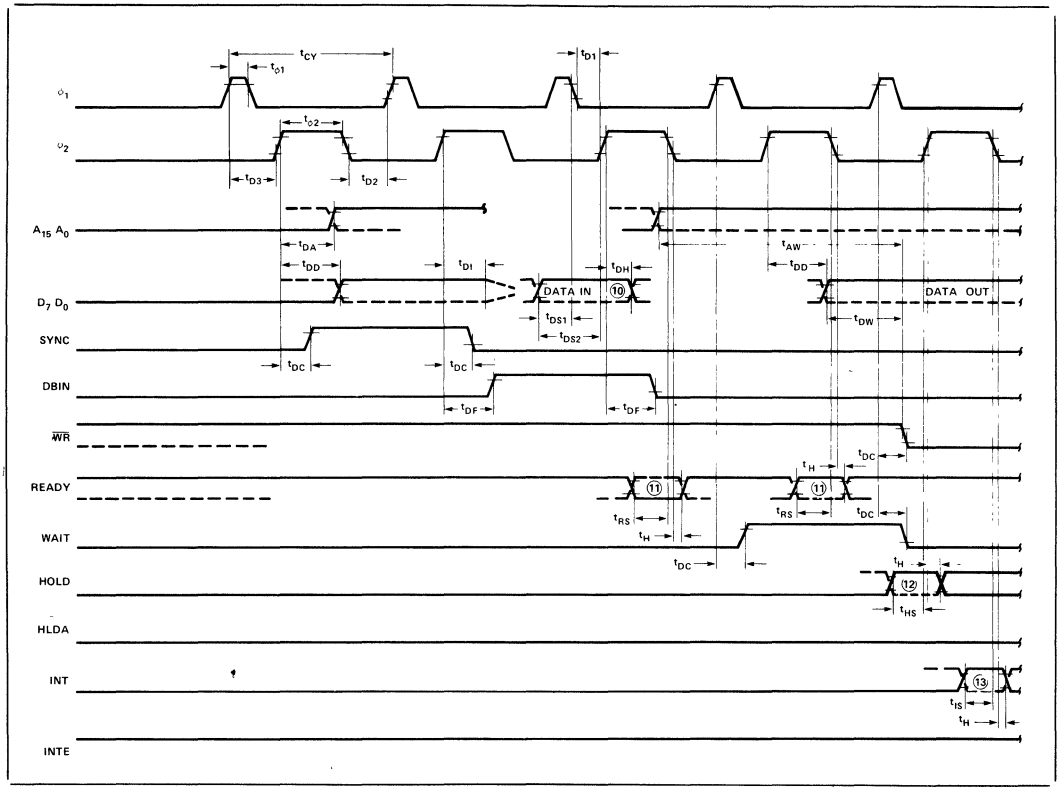
Symbol	Parameter	Min.	Max.	-1 Min.	-1 Max.	-2 Min.	-2 Max.	Unit	Test Condition
t_{CY}[3]	Clock Period	0.48	2.0	0.32	2.0	0.38	2.0	μsec	
t_r, t_f	Clock Rise and Fall Time	0	50	0	25	0	50	nsec	
$t_{\phi 1}$	ϕ_1 Pulse Width	60		50		60		nsec	
$t_{\phi 2}$	ϕ_2 Pulse Width	220		145		175		nsec	
t_{D1}	Delay ϕ_1 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	
t_{WD}	Output Data Stable From WR	[7]		[7]		[7]		nsec	
t_{WA}	Address Stable From WR	[7]		[7]		[7]		nsec	
t_{HF}	HLDA to Float Delay	[8]		[8]		[8]		nsec	
t_{WF}	WR to Float Delay	[9]		[9]		[9]		nsec	
t_{AH}	Address Hold Time After DBIN During HLDA	−20		−20		−20		nsec	

A.C. TESTING LOAD CIRCUIT



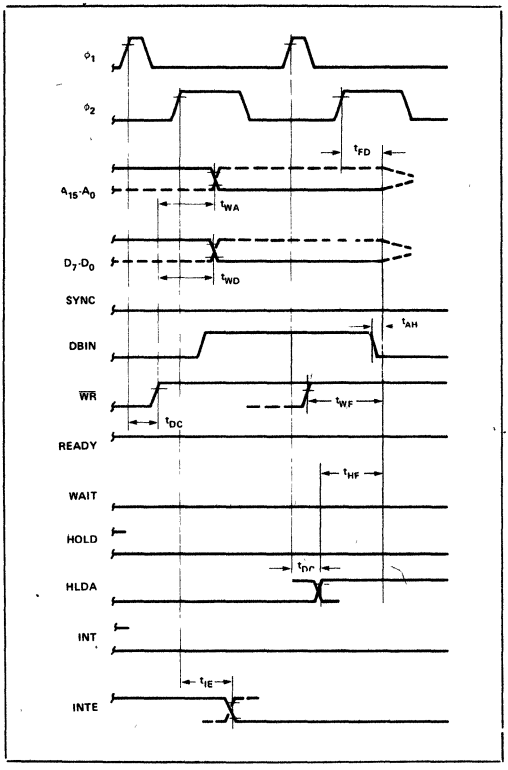
C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

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.

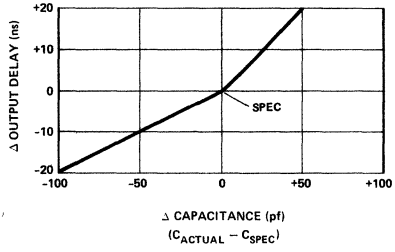
WAVEFORMS (Continued)



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. $t_{DH} \geq 50$ 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. $t_{AW} = 2\,t_{CY} - t_{D3} - t_{r\phi 2} - 140$ 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 8755A (EPROM/IO)] while maintaining total system expandability. The 8085AH-2 and 8085AH-1 are faster versions of the 8085 AH.

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/8755A memory products allow a direct interface with the 8085AH.

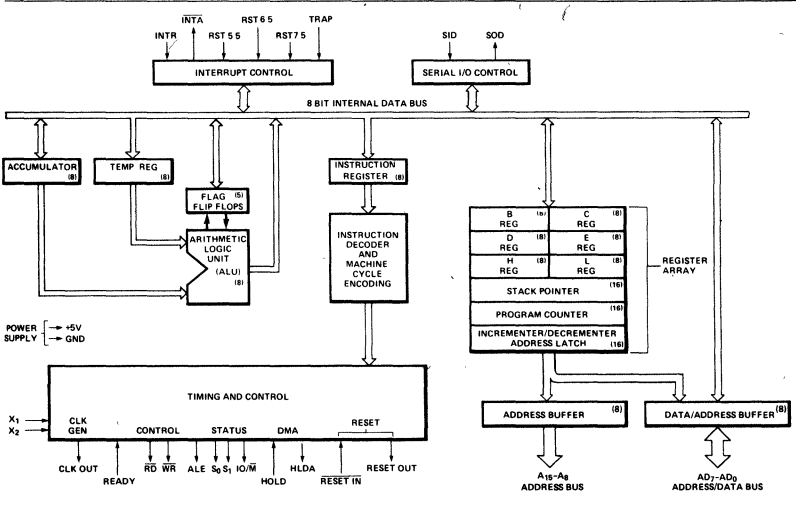


Figure 1. 8085AH CPU Functional Block Diagram

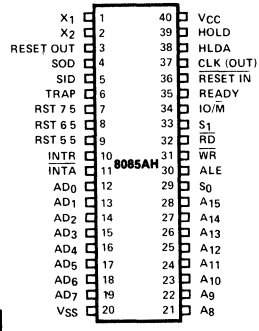


Figure 2. 8085AH Pin Configuration

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.

Symbol	Type	Name and Function
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.

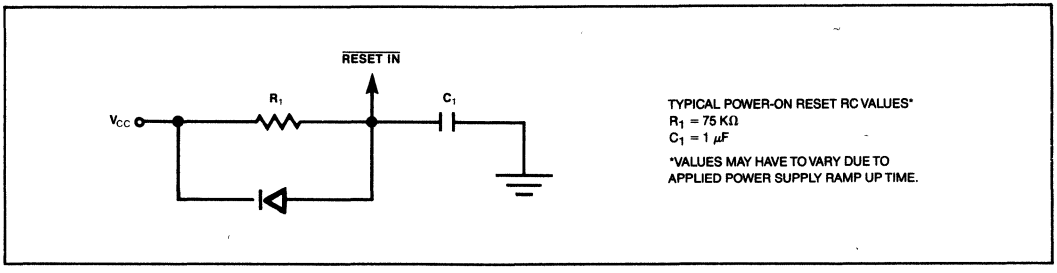


Figure 3. Power-On Reset Circuit

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 an EPROM/IO chip (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.

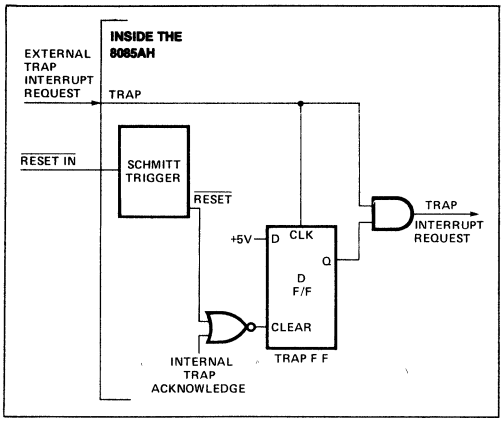


Figure 4. TRAP and RESET IN Circuit

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:

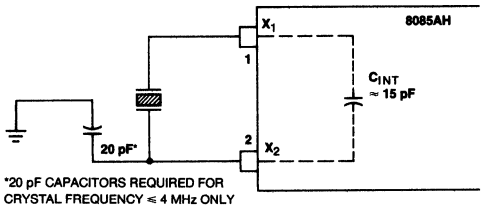
$$f = \frac{1}{2\pi\sqrt{L(C_{ext} + C_{int})}}$$

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.

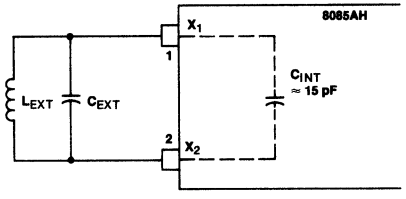
An RC circuit may be used as the frequency-determining network for the 8085AH if maintaining a precise clock frequency is of no importance. Variations in the on-chip timing generation can cause a wide variation in frequency when using the RC mode. Its advantage is its low component cost. The driving 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.

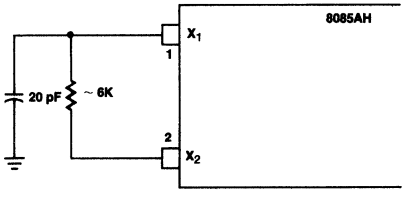
For driving frequencies up to and including 6 MHz you may supply the driving signal to X_1 and leave X_2 open-circuited (Figure 5D). If the driving frequency is from 6 MHz to 12 MHz, stability of the clock generator will be improved by driving both X_1 and X_2 with a push-pull source (Figure 5E). To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



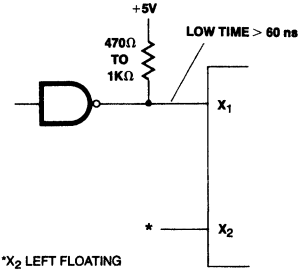
a. Quartz Crystal Clock Driver



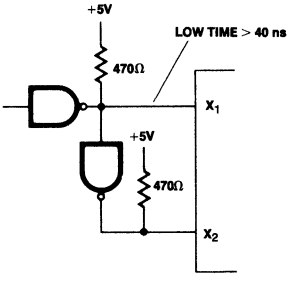
b. LC Tuned Circuit Clock Driver



c. RC Circuit Clock Driver



d. 1-6 MHz Input Frequency External Clock Driver Circuit



e. 1-12 MHz Input Frequency External Clock Driver 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.

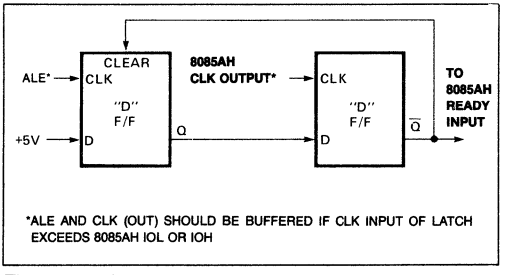


Figure 6. Generation of a Wait State for 8085AH CPU

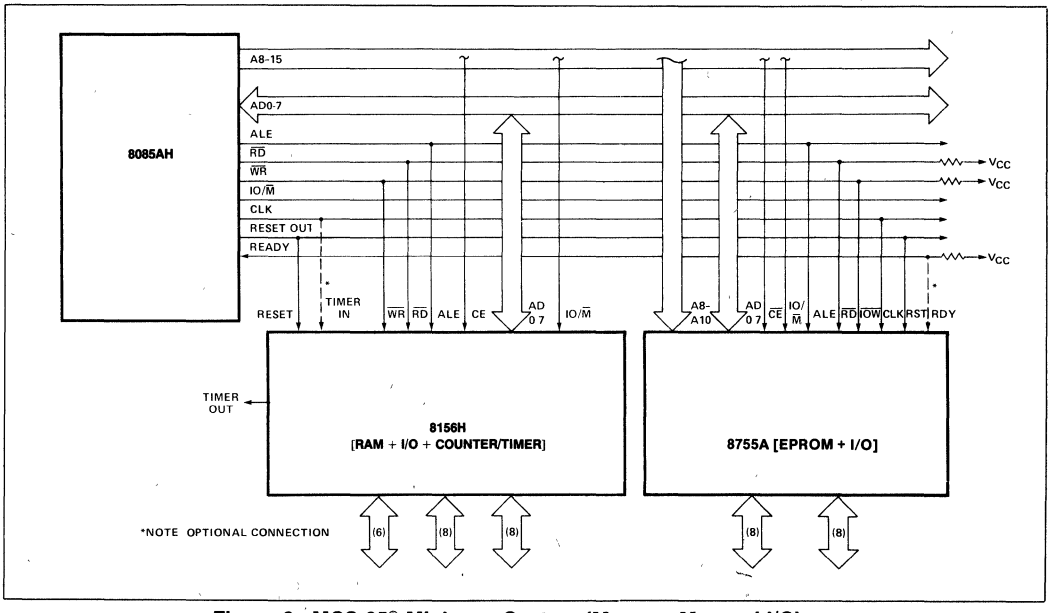


Figure 8. MCS-85® Minimum System (Memory Mapped I/O)

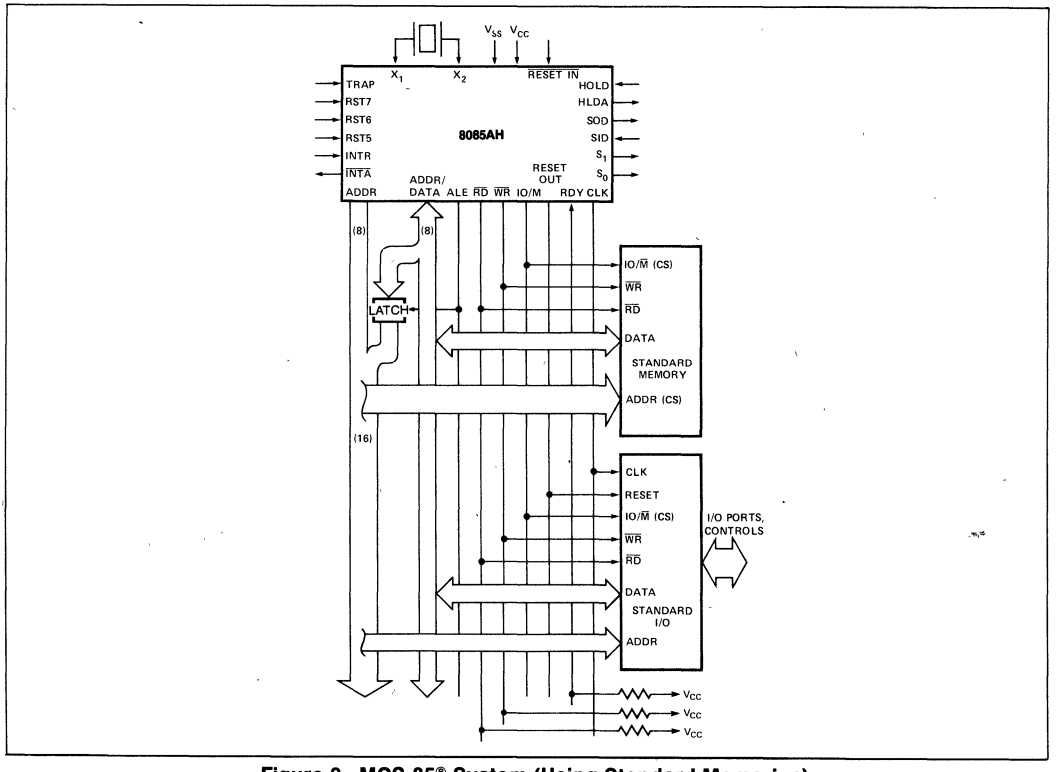


Figure 9. MCS-85® System (Using Standard Memories)

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 8755A will have the following features:

- 2K Bytes EPROM
- 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 8-bit latch as shown in Figure 9.

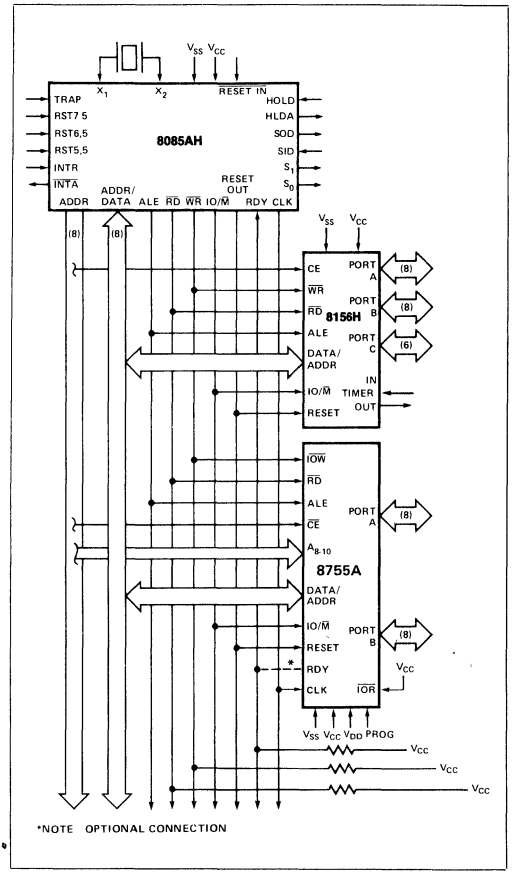


Figure 7. 8085AH Minimum System (Standard I/O Technique)

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"
1 = Logic "1"
TS = High Impedance
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

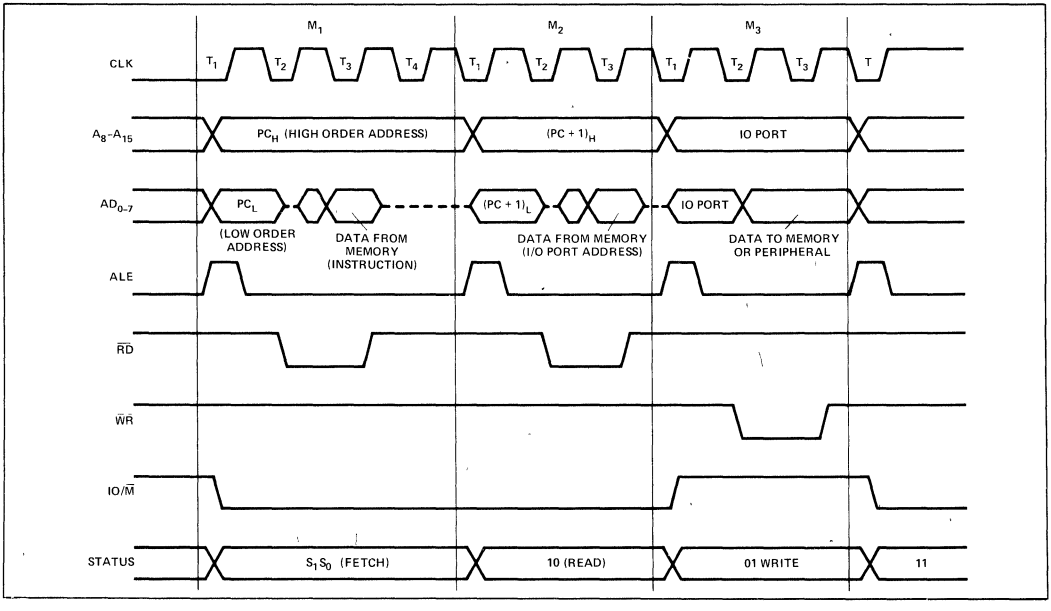


Figure 10. 8085AH Basic System Timing

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_{8-15} Valid to Leading Edge of Control[1]	270		115		70		ns
t_{ACL}	A_{0-7} Valid to Leading Edge of Control	240		115		60		ns
t_{AD}	A_{0-15} 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_{8-15} Valid Before Trailing Edge of ALE [1]	115		50		25		ns

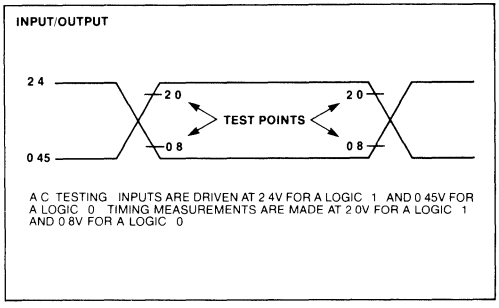
*Note: For Extended Temperature EXPRESS use M8085AH Electricals Parameters.

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

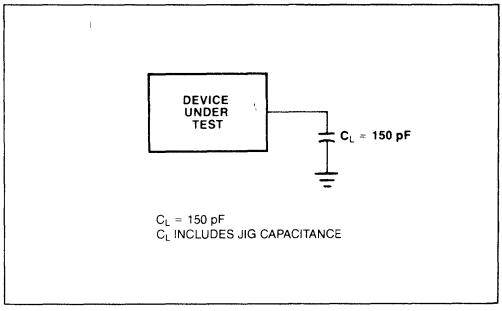


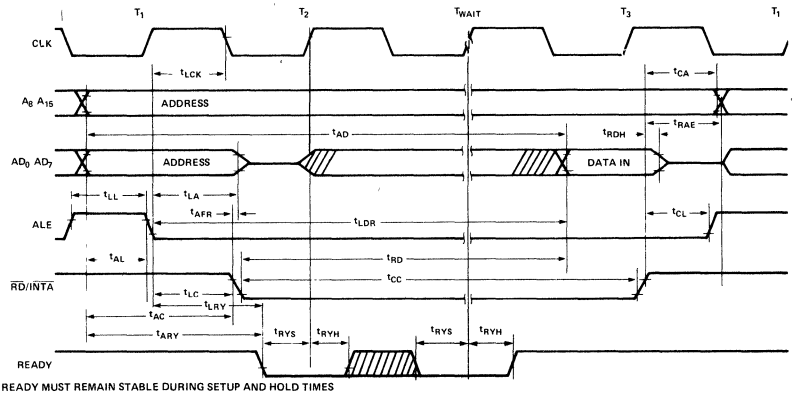
Table 5. Bus Timing Specification as a T_{CYC} Dependent

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}.

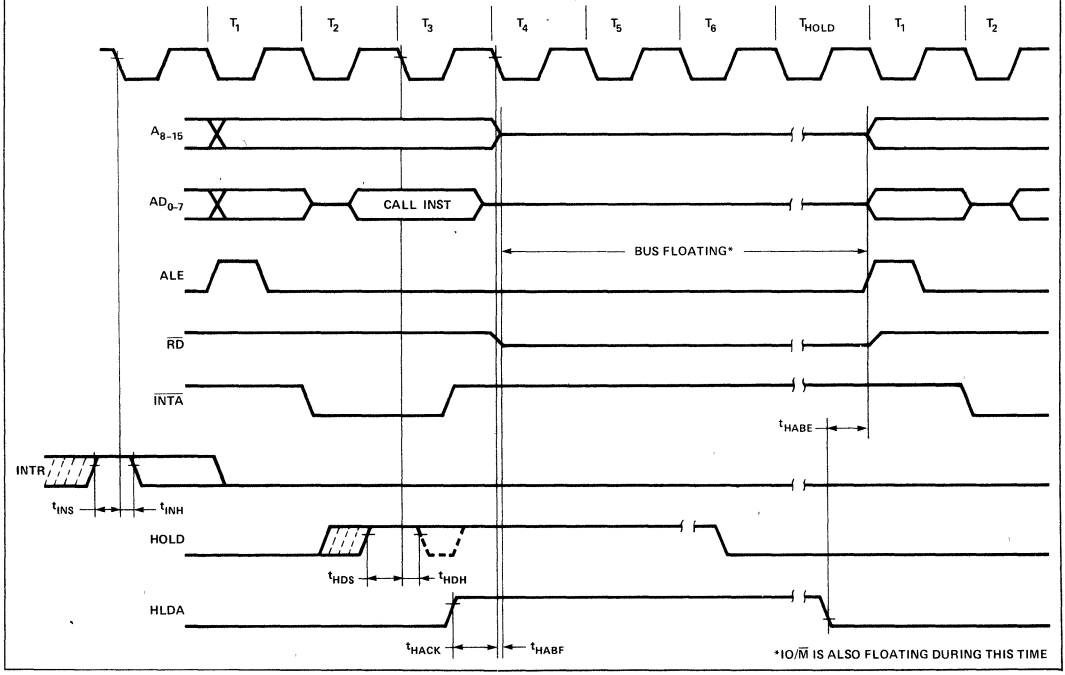
WAVEFORMS (Continued)

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
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

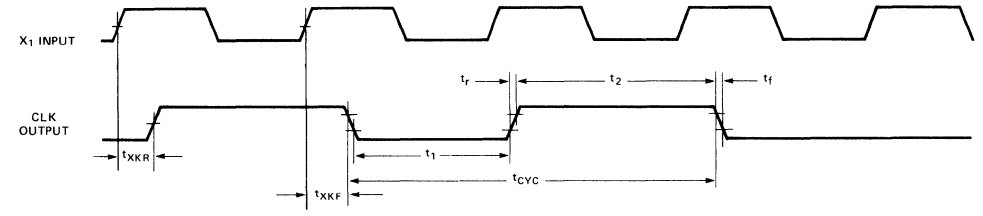
Mnemonic	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Operations Description
SPECIALS									
CMA	0	0	1	0	1	1	1	1	Complement A
STC	0	0	1	1	0	1	1	1	Set carry
CMC	0	0	1	1	1	1	1	1	Complement carry
DAA	0	0	1	0	0	1	1	1	Decimal adjust A
CONTROL									
EI	1	1	1	1	1	0	1	1	Enable Interrupts
DI	1	1	1	1	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 8085AH 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:

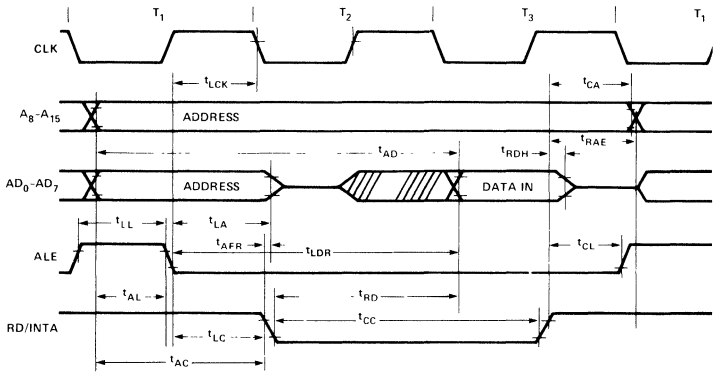
1. DDS or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111
2. Two possible cycle times (6/12) indicate instruction cycles dependent on condition flags

WAVEFORMS

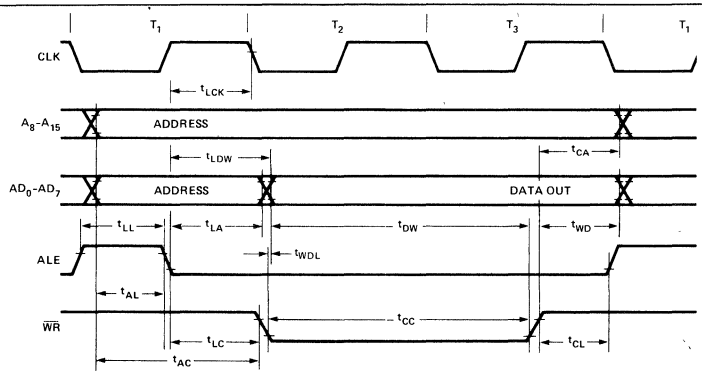
CLOCK



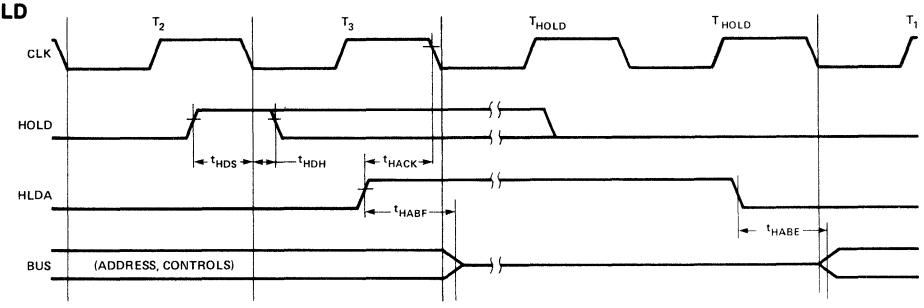
READ



WRITE



HOLD



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.

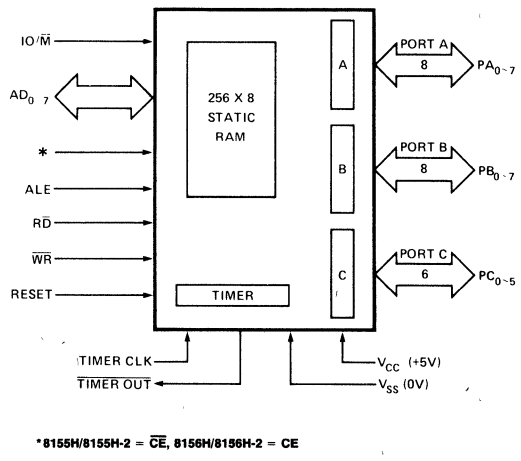


Figure 1. Block Diagram

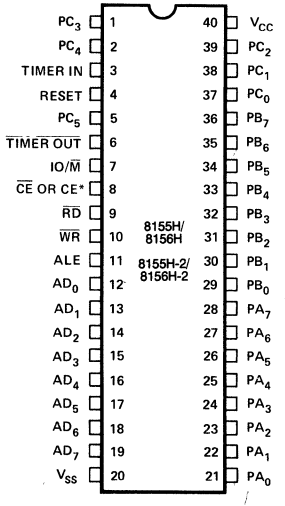


Figure 2. Pin Configuration

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

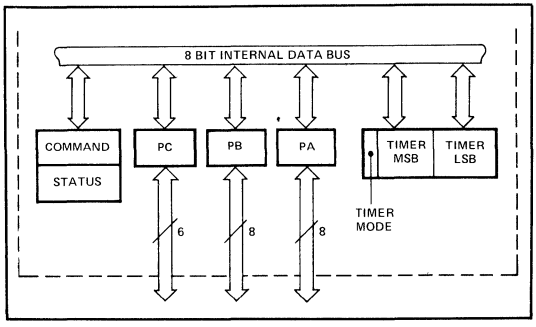


Figure 3. 8155H/8156H Internal Registers

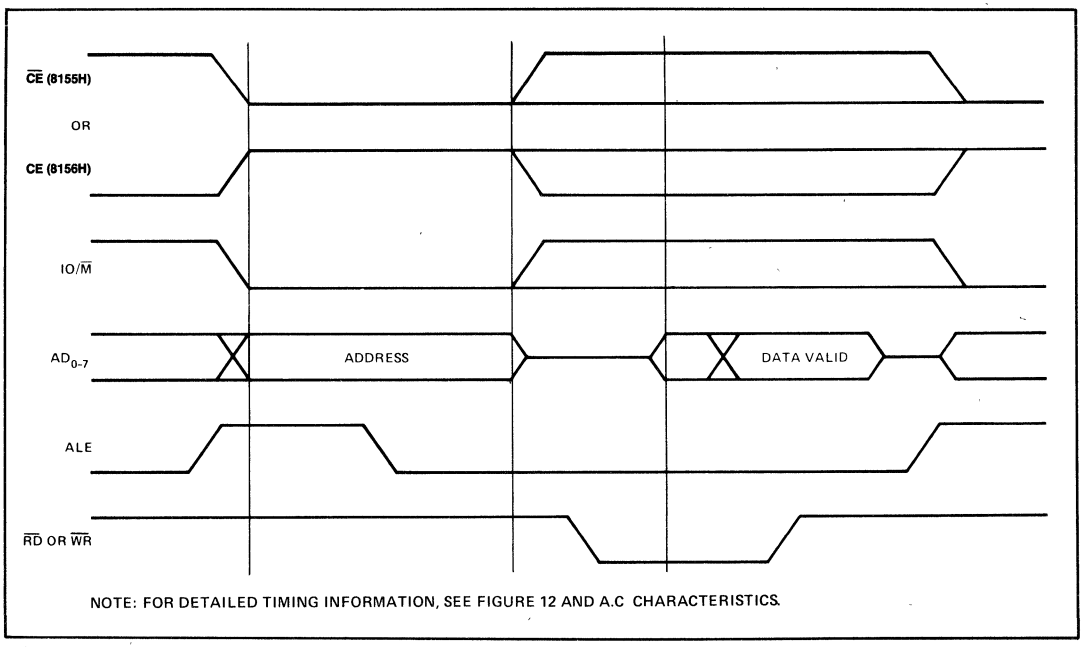


Figure 4. 8155H/8156H On-Board Memory Read/Write Cycle

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

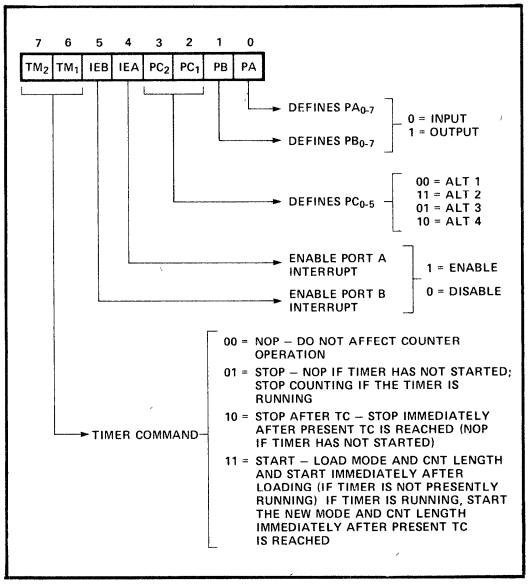


Figure 5. Command Register Bit Assignment

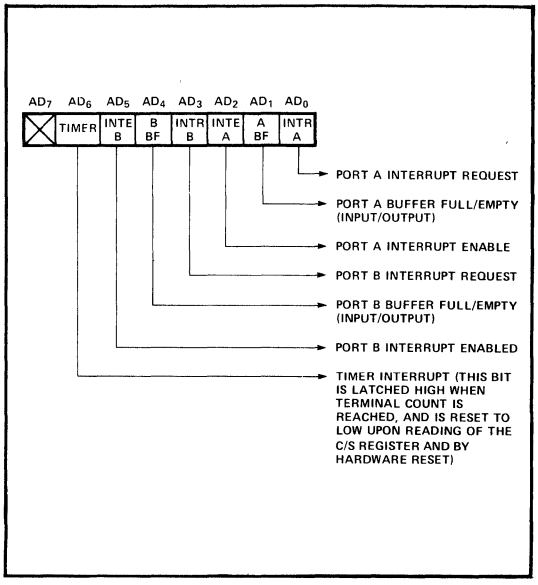


Figure 6. Status Register Bit Assignment

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:

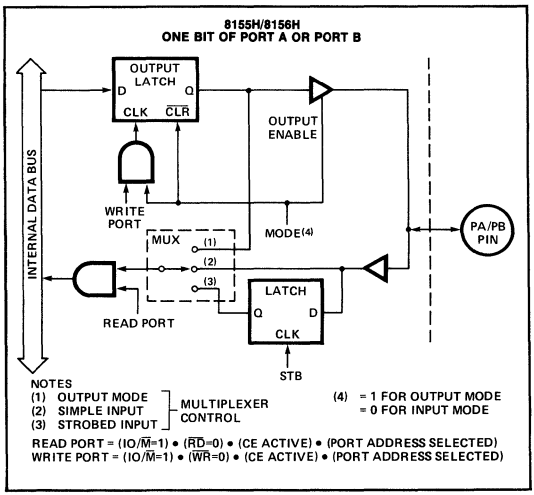


Figure 8. 8155H/8156H Port Functions

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.

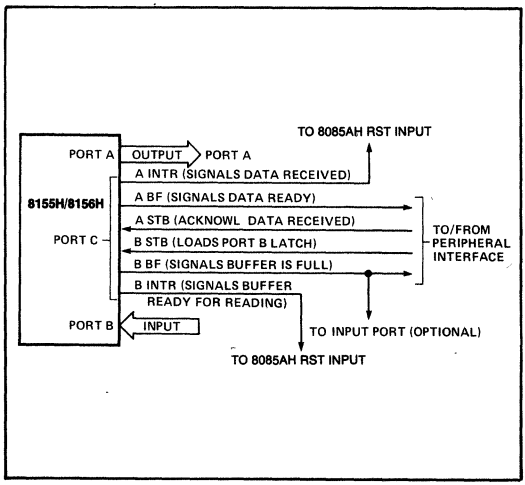


Figure 9. Example: Command Register = 00111001

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 3FFFH in Bits 0-13.

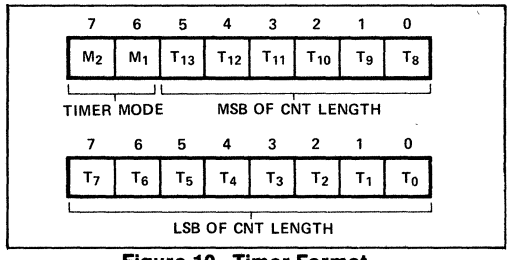


Figure 10. Timer Format

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

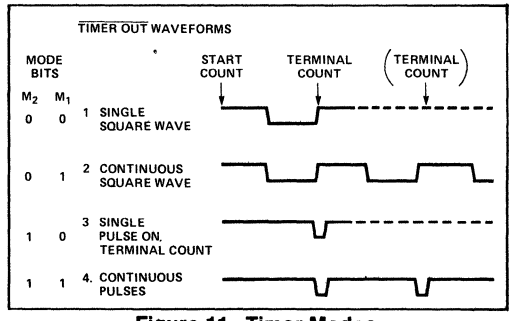


Figure 11. Timer Modes

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.

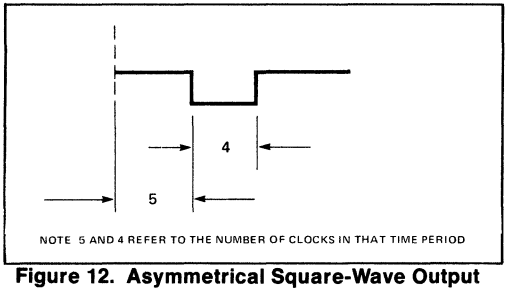


Figure 12. Asymmetrical Square-Wave Output Resulting from Count of 9

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 EPROM
- 38 I/O Pins
- 1 Interval Timer
- 4 Interrupt Levels

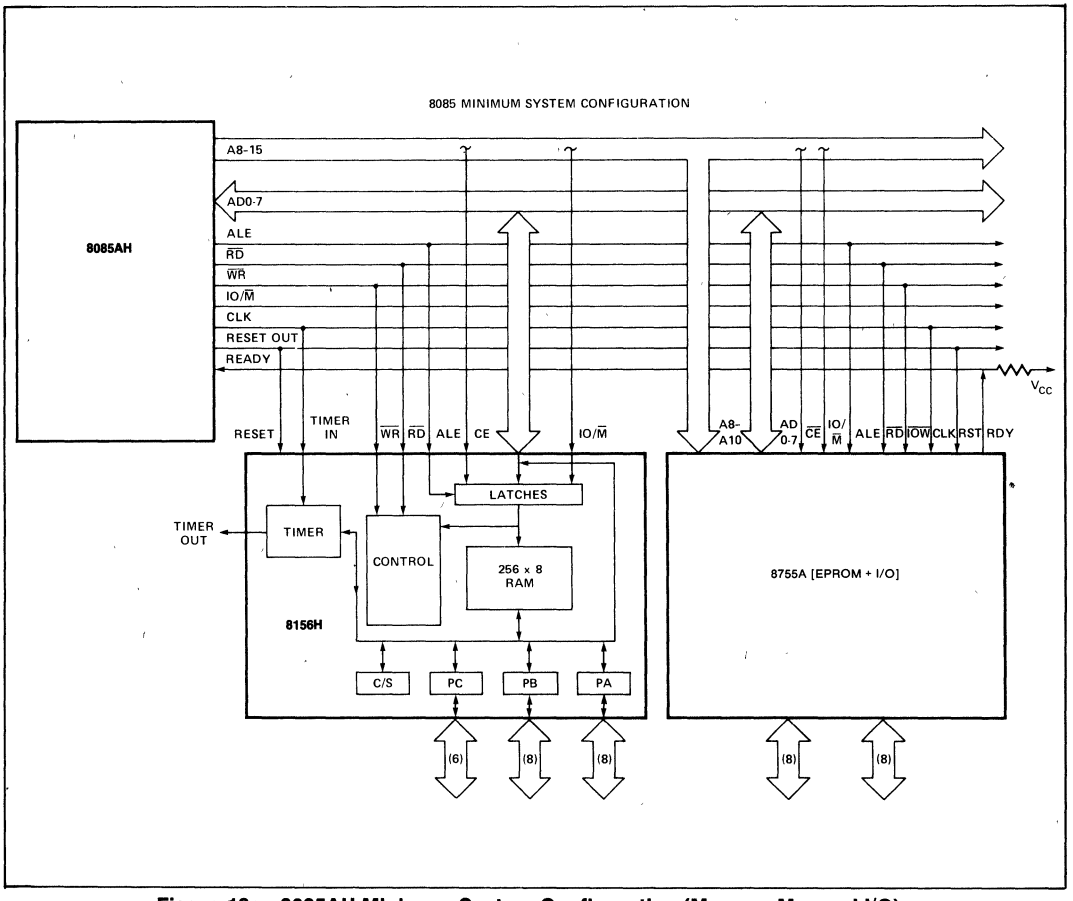


Figure 13a. 8085AH Minimum System Configuration (Memory Mapped I/O)

8088 FIVE CHIP SYSTEM

Figure 13b shows a five chip system containing:

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

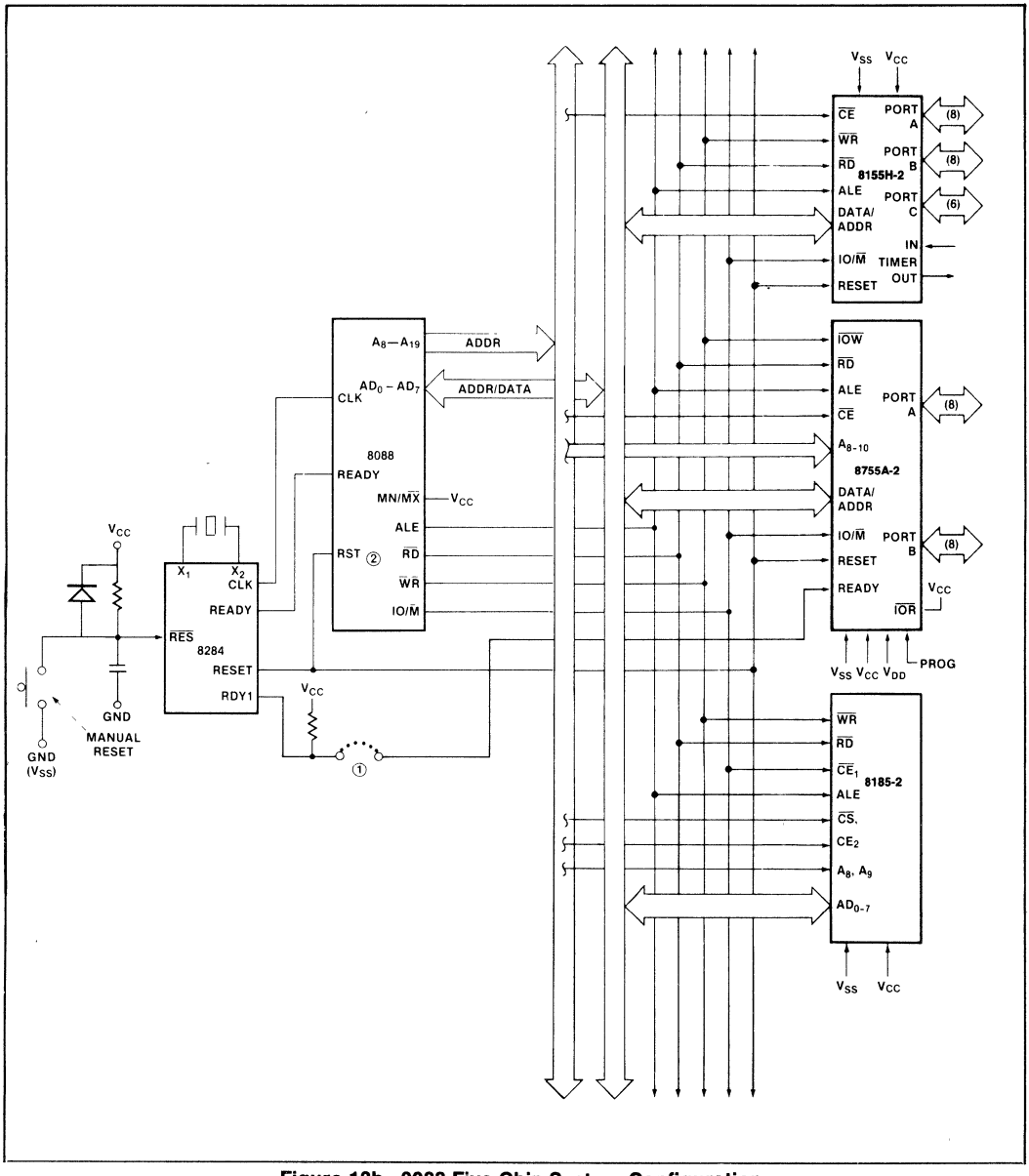


Figure 13b. 8088 Five Chip System Configuration

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	$0V \leq V_{IN} \leq V_{CC}$

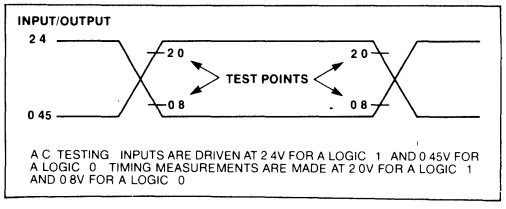
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

		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_{LD}	Latch to Data Out Valid		350		270	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%)

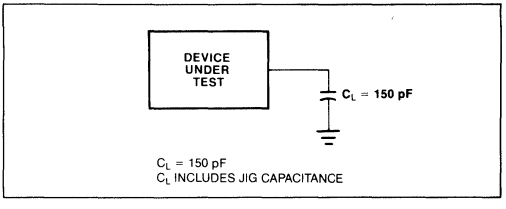
		8155H/8156H		8155H-2/8156H-2		
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer 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
t_{WT}	WRITE to TIMER-IN (for writes which start counting)	360		200		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



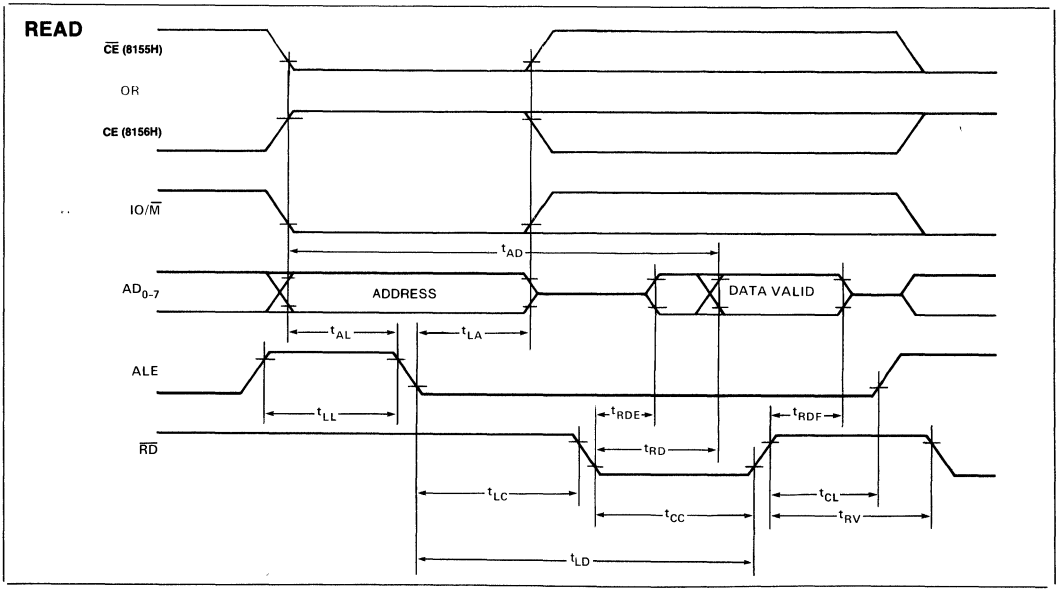
A C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC 1 AND 0 45V FOR A LOGIC 0 TIMING MEASUREMENTS ARE MADE AT 2 0V FOR A LOGIC 1 AND 0 8V FOR A LOGIC 0

A.C. TESTING LOAD CIRCUIT



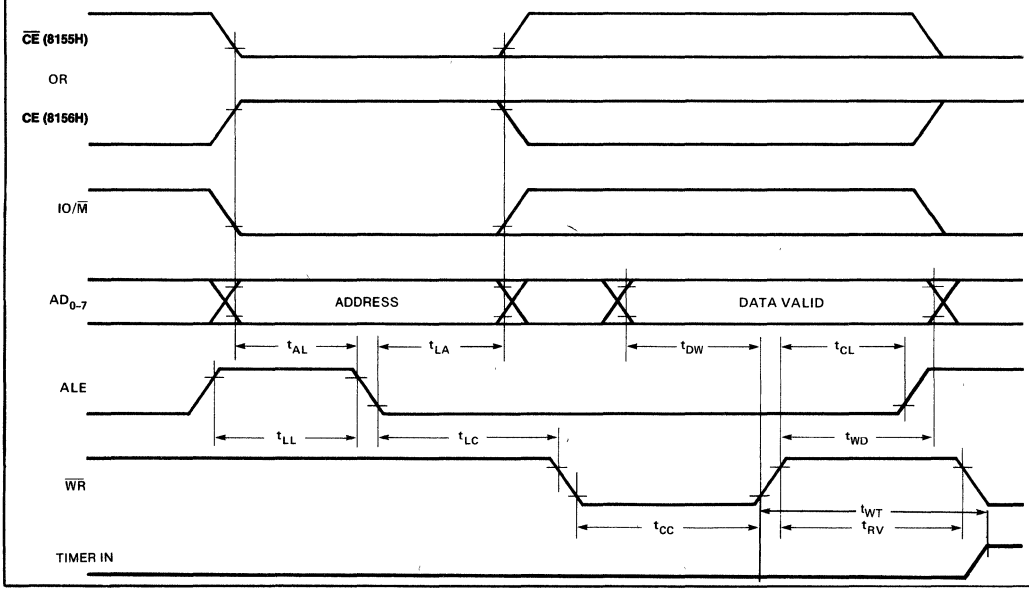
C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

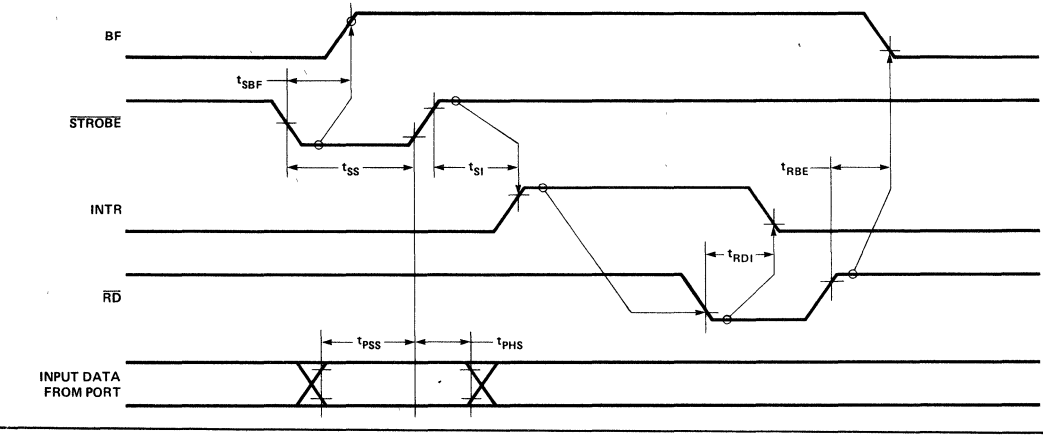


WAVEFORMS (Continued)

WRITE

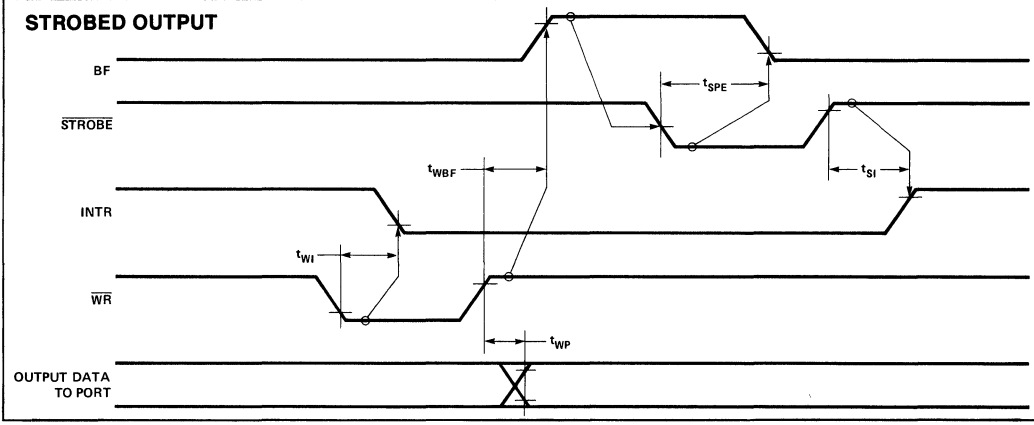


STROBED INPUT

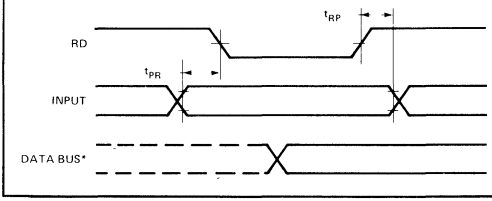


WAVEFORMS (Continued)

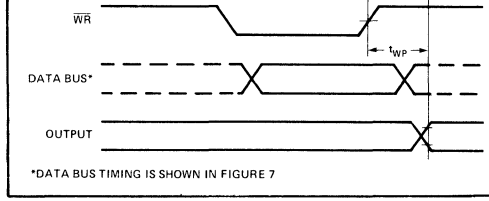
STROBED OUTPUT



BASIC INPUT

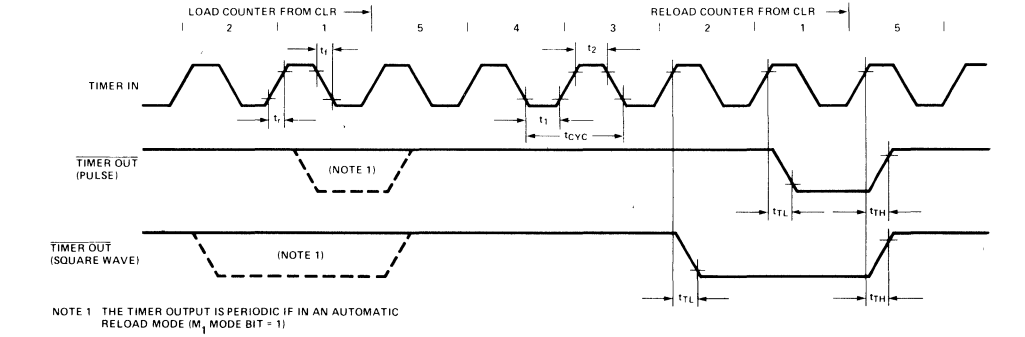


BASIC OUTPUT



*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)

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

- **Multiplexed Address and Data Bus**
- **Directly Compatible with 8085AH 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 8085AH-2 and the 5 MHz iAPX 88.

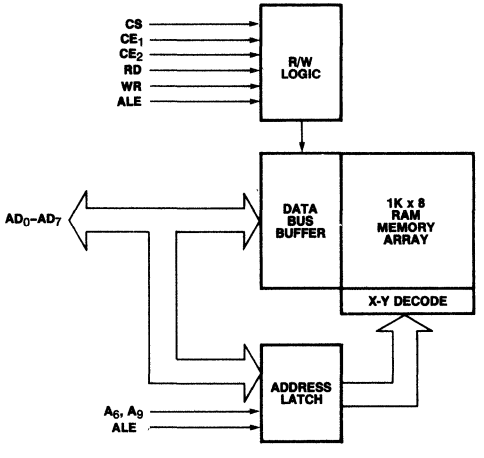
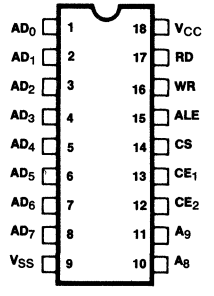


Figure 1. Block Diagram



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.

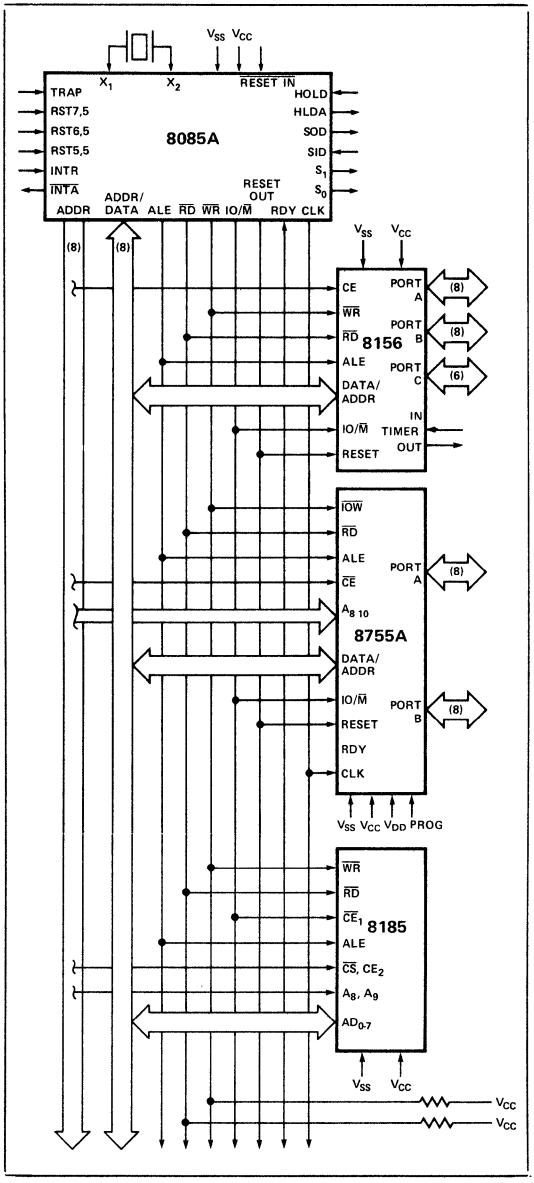


Figure 3. 8185 in an MCS-85 System

4 Chips:
2K Bytes EPROM
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
- 2K Bytes EPROM
- 38 I/O Pins
- 1 Internal Timer
- 2 Interrupt Levels

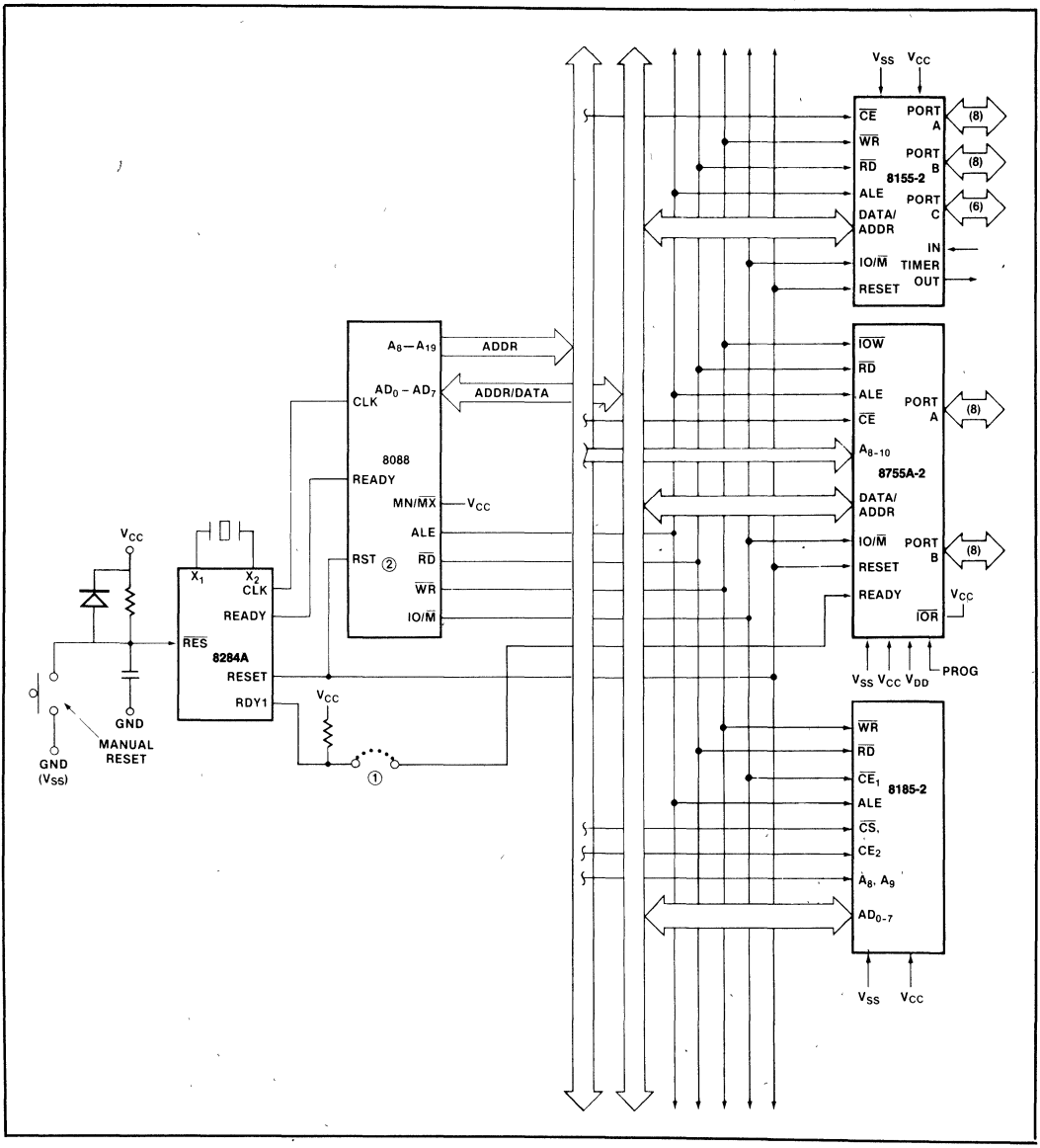


Figure 4. iAPX 88 Five Chip System Configuration

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.*

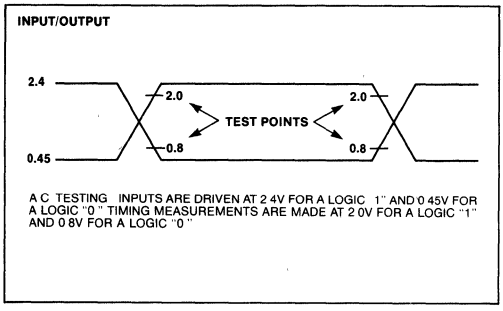
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

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	

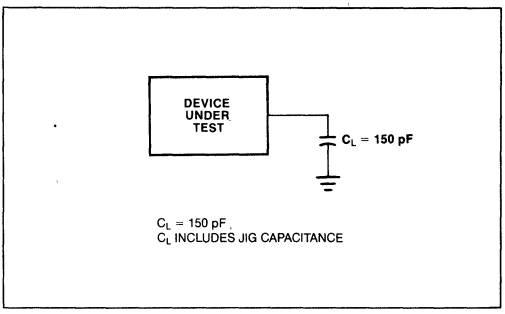
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

		8185		8185-2		
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_{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

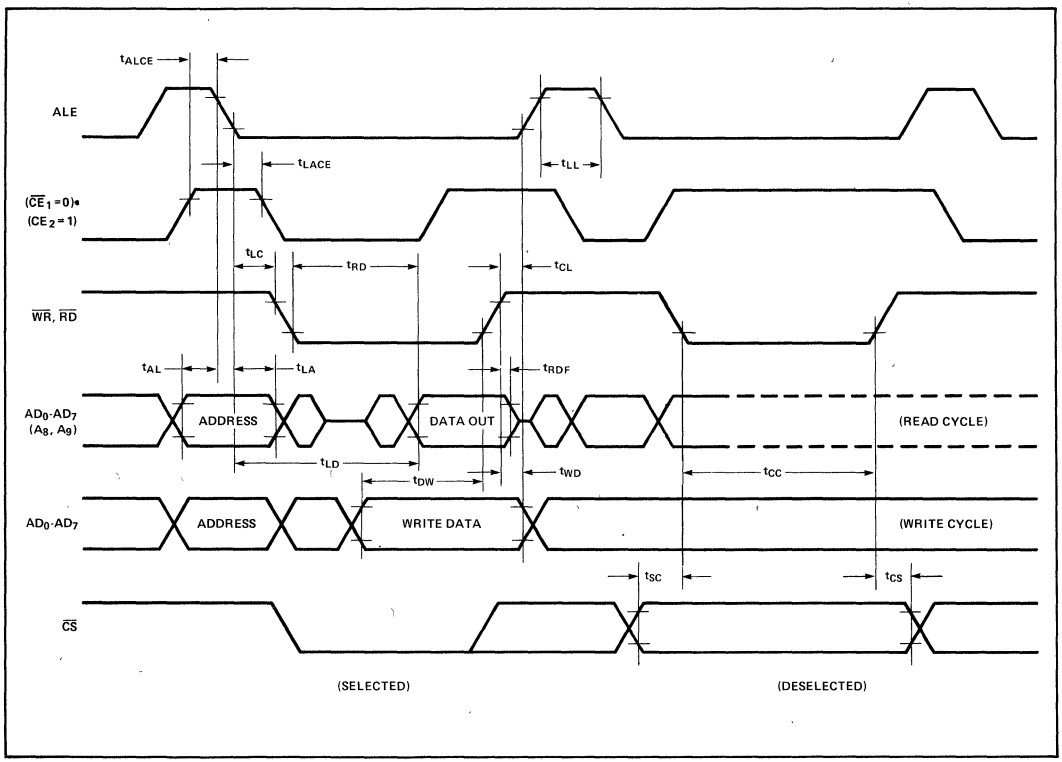
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT



WAVEFORM

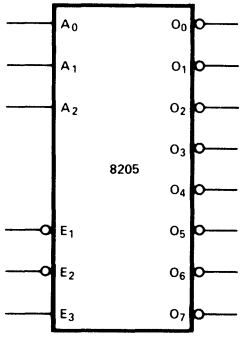


8205
HIGH SPEED 1 OUT OF 8 BINARY DECODER

- **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**

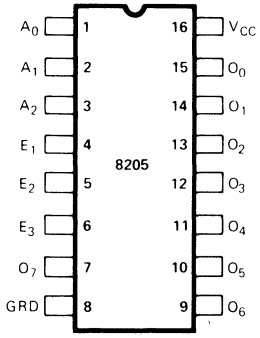
The Intel® 8205 decoder can be used for expansion of systems which utilize input ports, output ports, and memory components with active low chip select input. When the 8205 is enabled, one of its 8 outputs goes "low," thus a single row of a memory system is selected. The 3-chip enable inputs on the 8205 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.

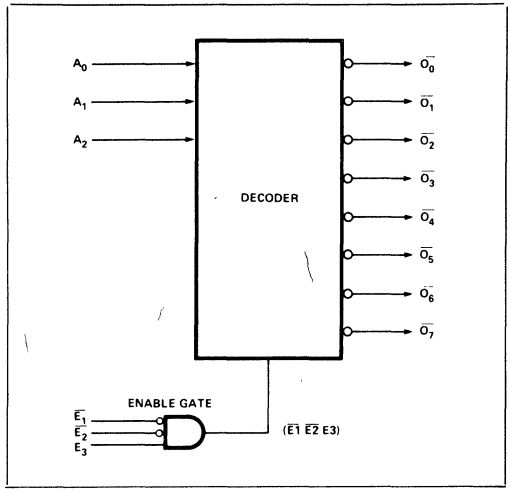


Figure 3. Enable Gate

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).

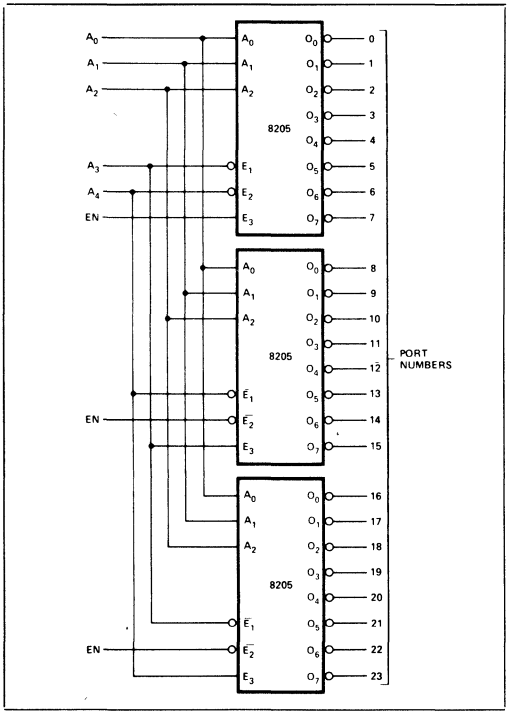


Figure 4. I/O Port Decoder

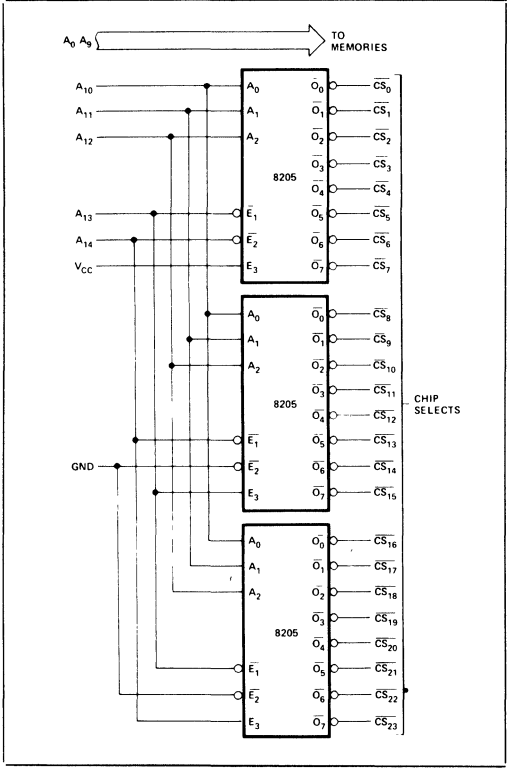


Figure 5. 24K Memory Interface

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%)

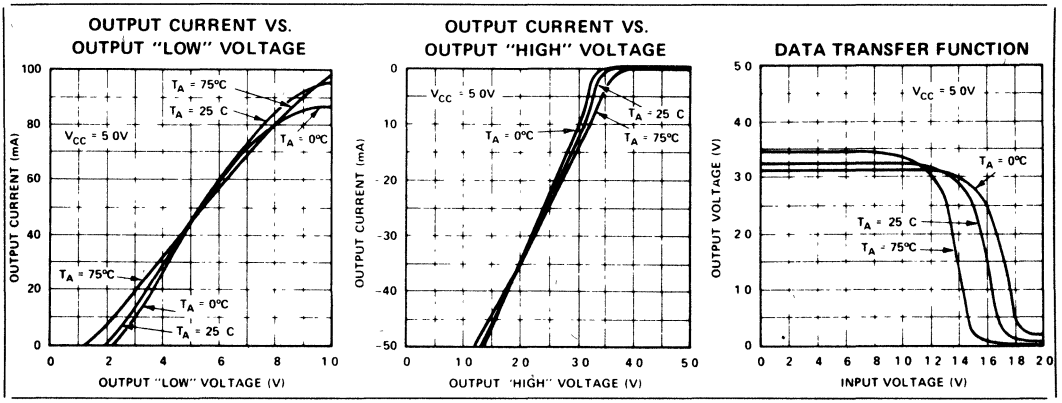
Symbol	Parameter	Limit Min.	Limit Max.	Unit	Test Conditions
I_F	INPUT LOAD CURRENT		−0.25	mA	V_{CC} = 5.25V, V_F = 0.45V
I_R	INPUT LEAKAGE CURRENT		10	μA	V_{CC} = 5.25V, V_R = 5.25V
V_C	INPUT FORWARD CLAMP VOLTAGE		−1.0	V	V_{CC} = 4.75V, I_C = −5.0 mA
V_{OL}	OUTPUT "LOW" VOLTAGE		0.45	V	V_{CC} = 4.75V, 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

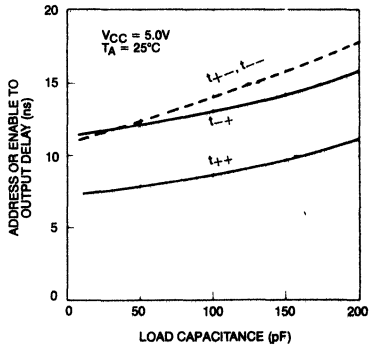
1 This parameter is periodically sampled and is not 100% tested

TYPICAL CHARACTERISTICS

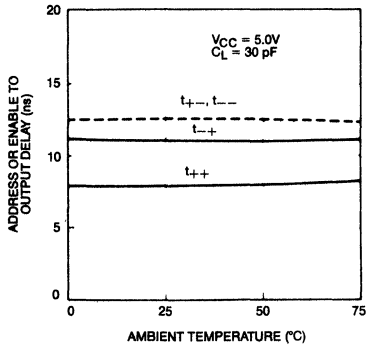


TYPICAL CHARACTERISTICS (Continued)

ADDRESS OR ENABLE TO OUTPUT DELAY VS. LOAD CAPACITANCE



ADDRESS OR ENABLE TO OUTPUT DELAY VS. AMBIENT TEMPERATURE



SWITCHING CHARACTERISTICS

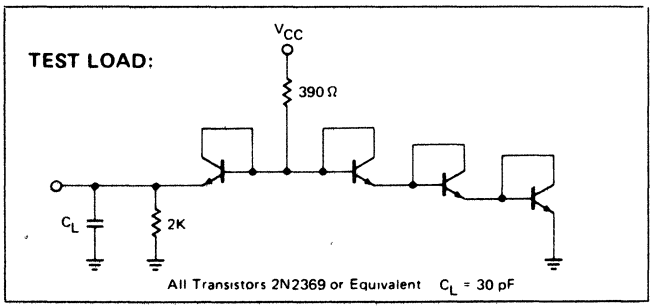
CONDITIONS OF TEST:

Input pulse amplitudes: 2.5V

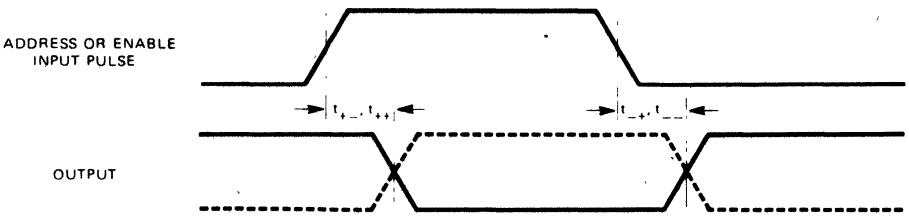
Input rise and fall times: 5 nsec between 1V and 2V

Measurements are made at 1.5V

TEST LOAD



WAVEFORMS



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.

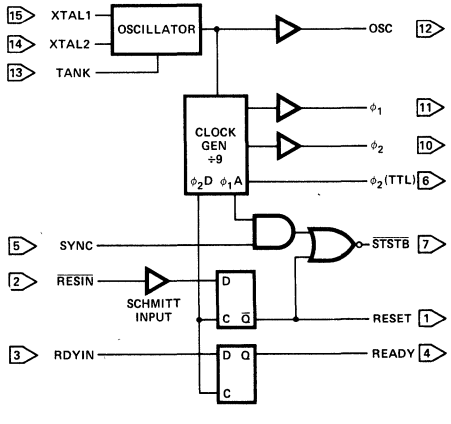


Figure 1. Block Diagram

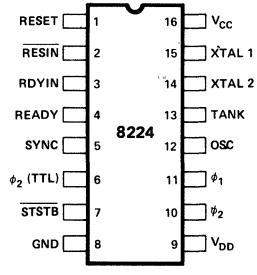


Figure 2. Pin Configuration

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	

XTAL 1	CONNECTIONS FOR CRYSTAL
XTAL 2	
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_{CC}	Power Supply Current			115	mA	
I_{DD}	Power Supply Current			12	mA	

Note 1 For crystal frequencies of 18 MHz connect 510Ω registers between the X1 input and ground as well as the X2 input and ground to prevent oscillation at harmonic frequencies

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)

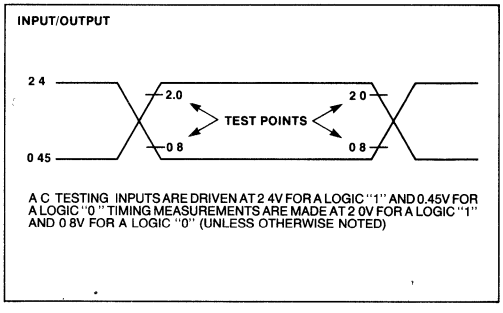
Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	ϕ_1 Pulse Width	$\frac{2tcy}{9}$ - 20ns			ns	C_L = 20pF to 50pF
$t_{\phi 2}$	ϕ_2 Pulse Width	$\frac{5tcy}{9}$ - 35ns				
t_{D1}	ϕ_1 to ϕ_2 Delay	0				
t_{D2}	ϕ_2 to ϕ_1 Delay	$\frac{2tcy}{9}$ - 14ns				
t_{D3}	ϕ_1 to ϕ_2 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)

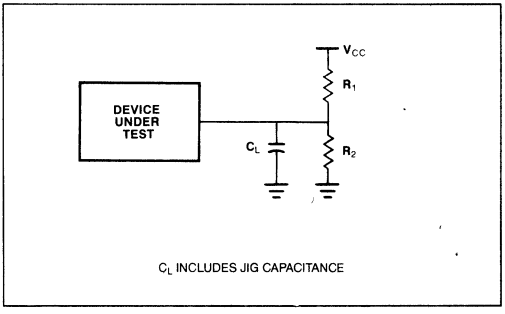
(For t_{CY} = 488.28 ns) (T_A = 0°C to 70°C, V_{DD} = +5V ±5%, V_{DD} = +12V ±5%)

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	

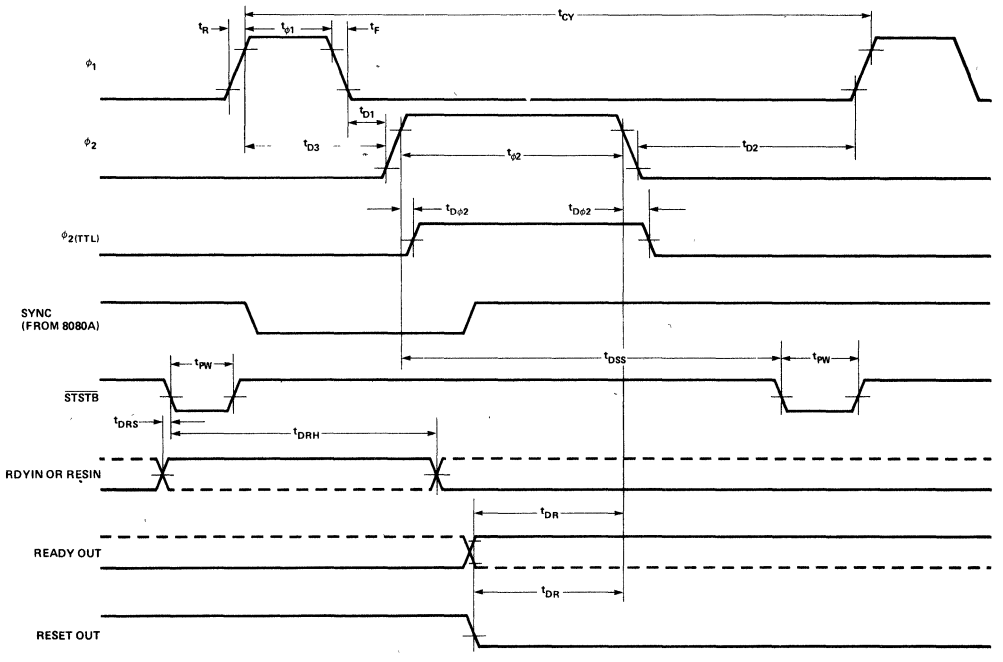
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT

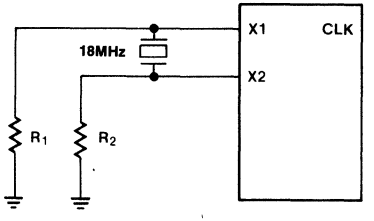


WAVEFORMS



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

CLOCK HIGH AND LOW TIME (USING X1, X2)



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

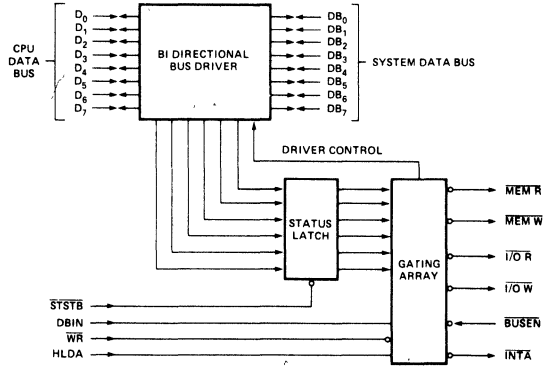
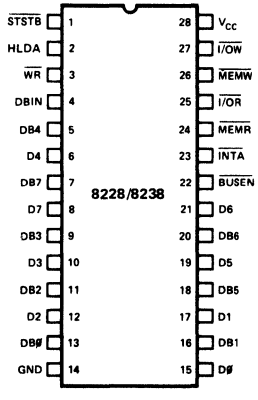


Figure 1. Block Diagram



D7 D0	DATA BUS (8080 SIDE)
DB7 DB0	DATA BUS (SYSTEM SIDE)
I/OR	I/O READ
I/OW	I/O WRITE
MEMR	MEMORY READ
MEMW	MEMORY WRITE
DBIN	DBIN (FROM 8080)

INTA	INTERRUPT ACKNOWLEDGE
HLDA	HLDA (FROM 8080)
WR	WR (FROM 8080)
BUSEN	BUS ENABLE INPUT
STSTB	STATUS STROBE (FROM 8224)
Vcc	+5V
GND	0 VOLTS

Figure 2. Pin Configuration

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)

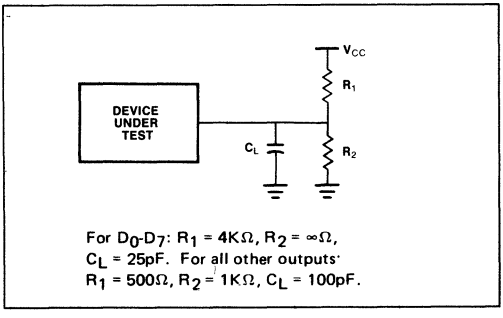
This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits Min.	Typ.[1]	Max.	Unit
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

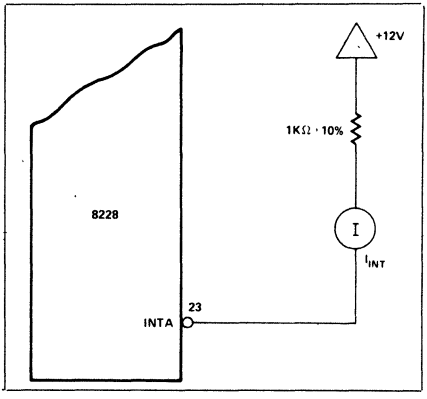
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±5%)

Symbol	Parameter	Limits Min.	Max.	Units	Condition
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

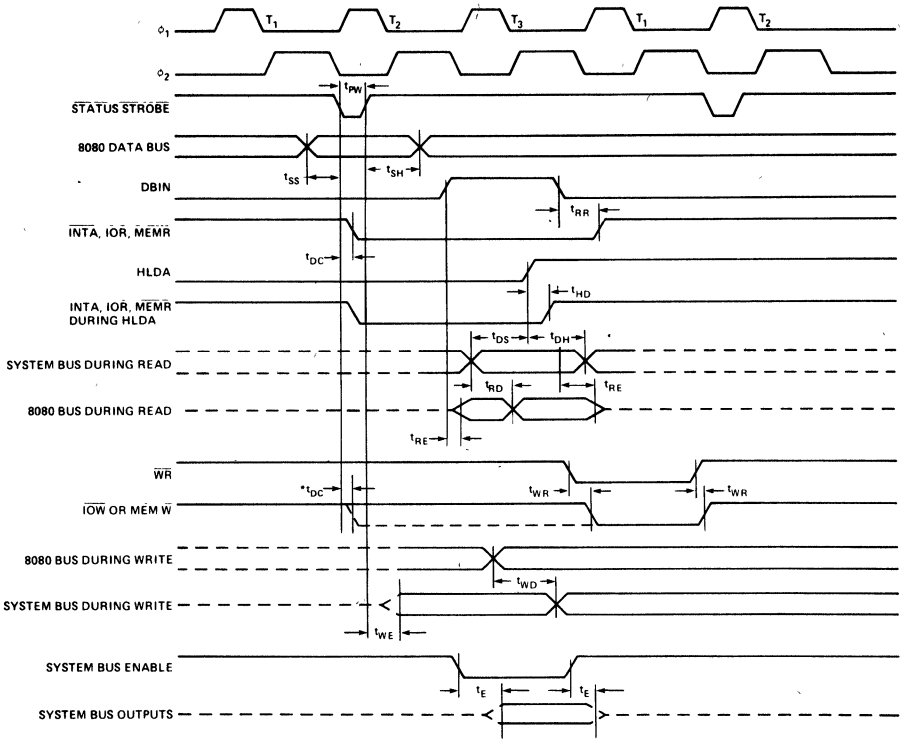


For D_0-D_7: R_1 = 4KΩ, R_2 = ∞Ω, C_L = 25pF. For all other outputs: R_1 = 500Ω, R_2 = 1KΩ, C_L = 100pF.



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 ($\overline{EOP}$).

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.

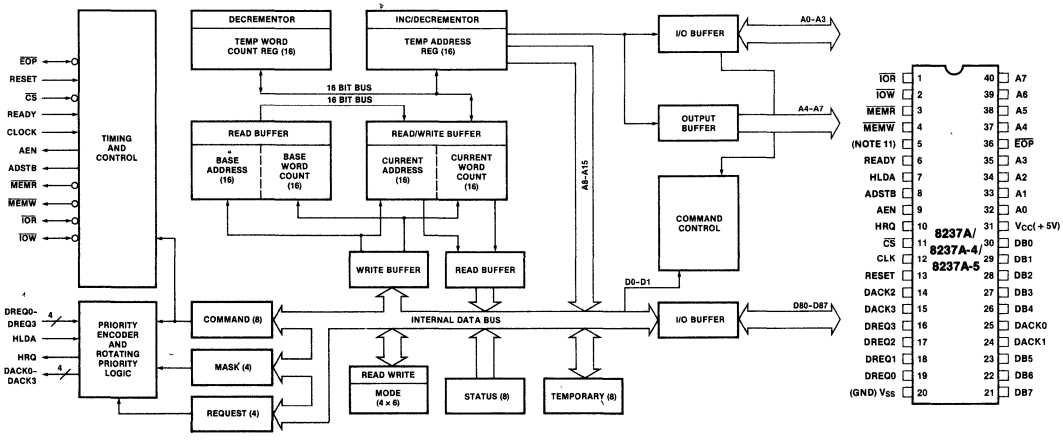


Figure 1. Block Diagram

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.
$\overline{CS}$	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 ($\overline{EOP}$) 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.

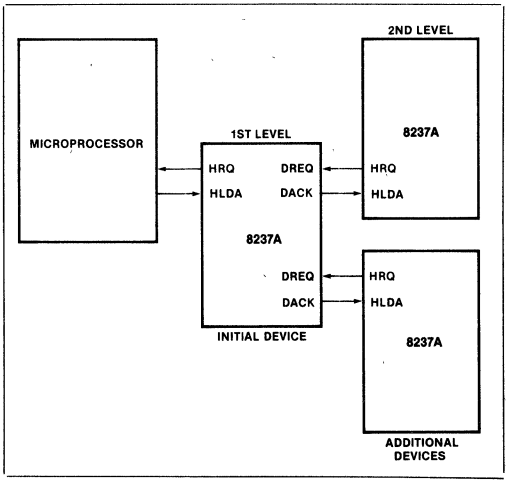


Figure 4. Cascaded 8237As

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.

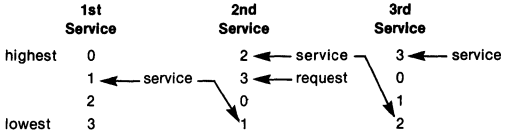
Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory.

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.

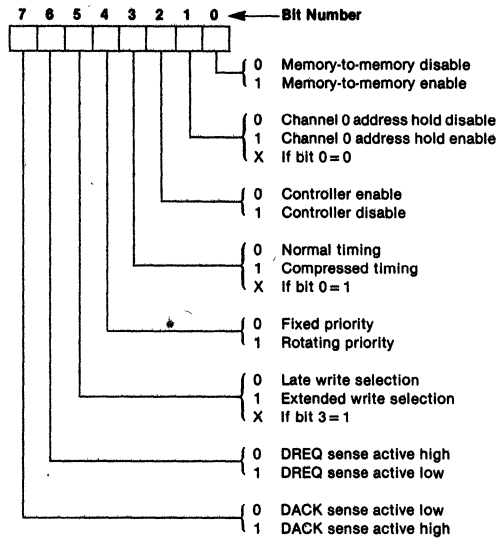
Base Address and Base Word Count Registers — Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original value of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. 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.

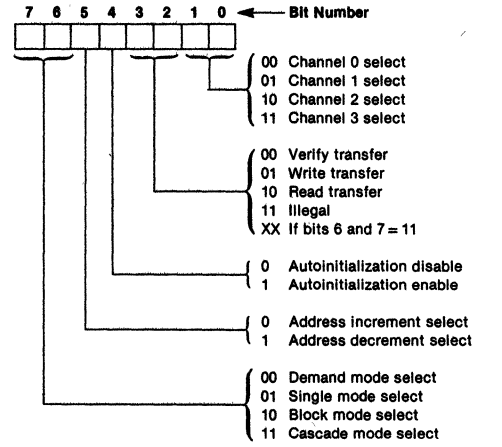
Mode Register — Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register is to be written.

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.

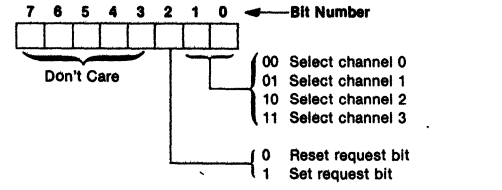
Command Register



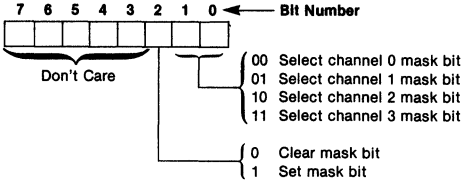
Mode Register



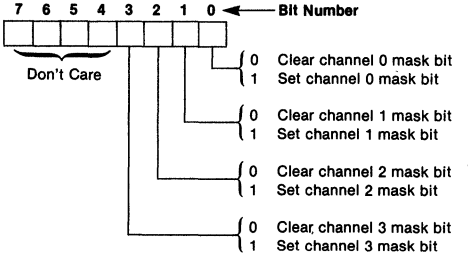
Request Register



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.



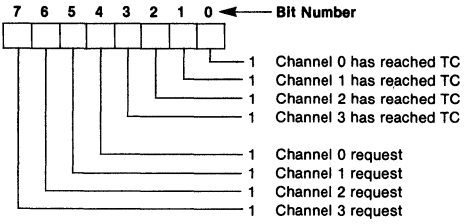
All four bits of the Mask register may also be written with a single command.



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	W0-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	W0-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	W0-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.

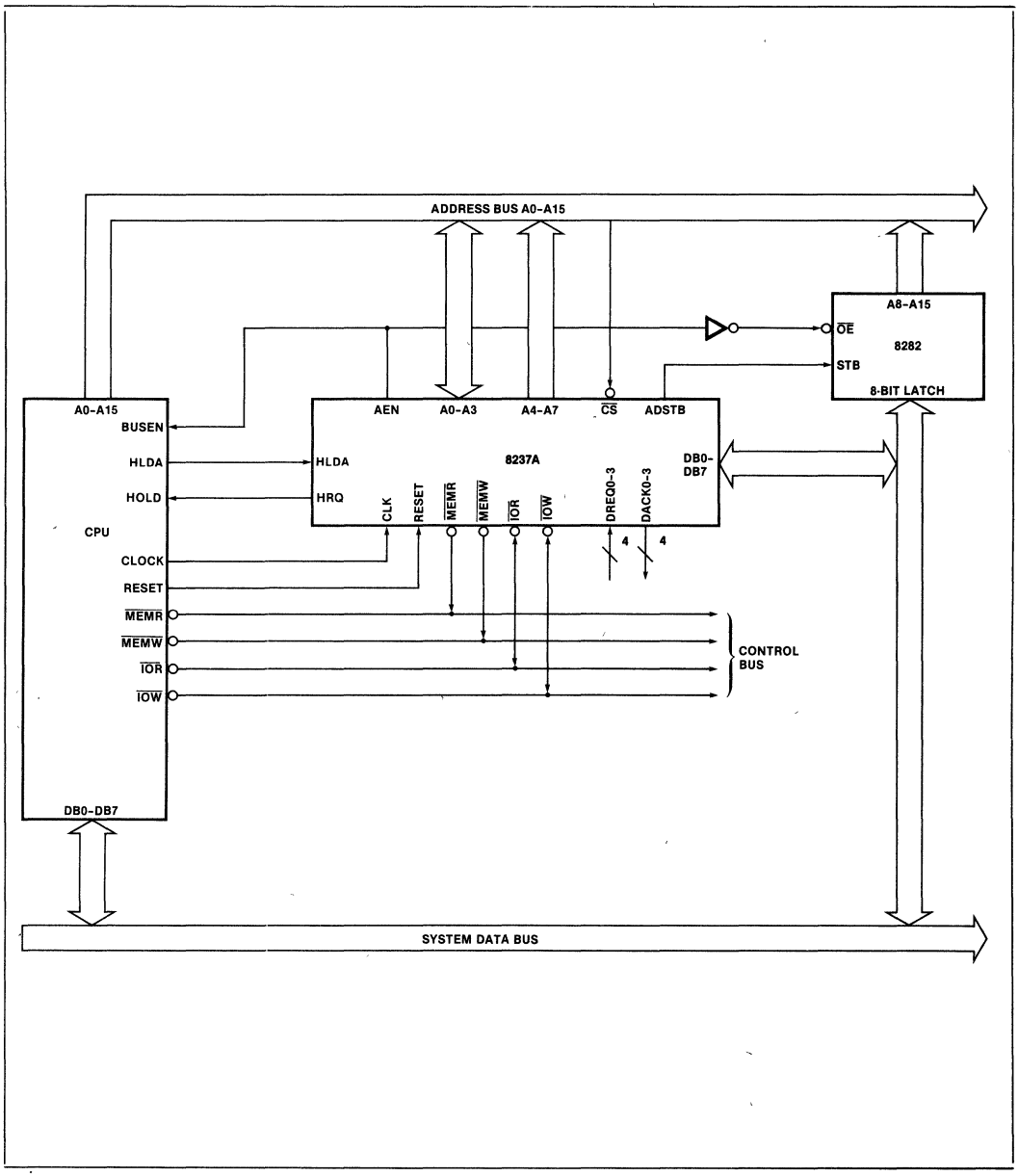


Figure 8. 8237A System Interface

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)

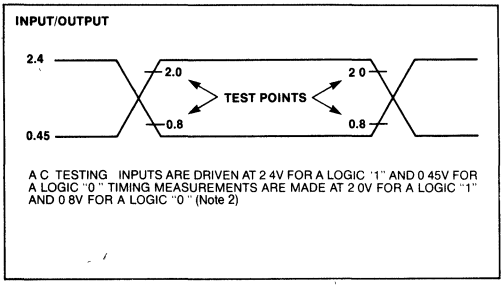
Symbol	Parameter	Min.	Typ.(1)	Max.	Unit	Test Conditions
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -200 μA
		3.3			V	I_{OH} = -100 μA (HRQ Only)
V_{OL}	Output LOW Voltage			.45	V	I_{OL} = 2.0mA (data Bus, $\overline{EOP}$) I_{OL} = 3.2mA (other outputs) (Note 8) I_{OL} = 2.5mA (ADSTB) (Note 8)
V_{IH}	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 1985, 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}
12. Output Loading on the Data Bus is TTL Gate plus 100 pF capacitance

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	40		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 ≤ 10 ns)	120		100		80		ns
TCL	Clock LOW Time (Transitions ≤ 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

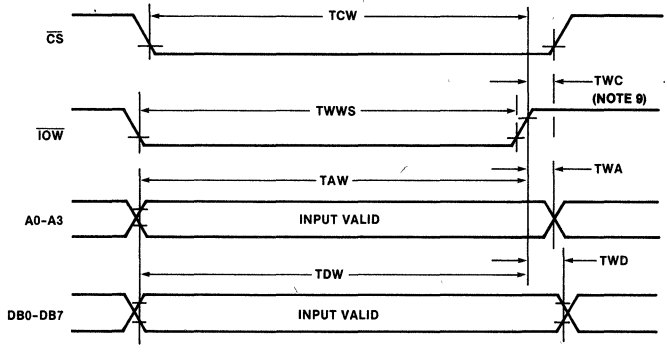


Figure 9. Slave Mode Write

SLAVE MODE READ TIMING

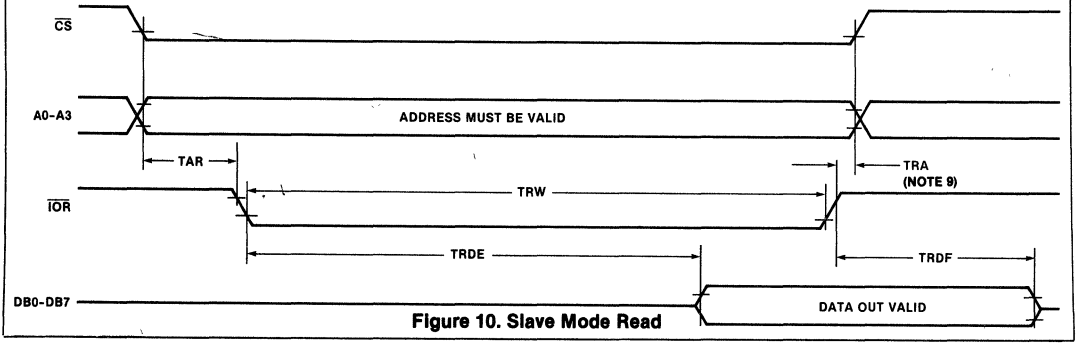


Figure 10. Slave Mode Read

WAVEFORMS (Continued)

DMA TRANSFER TIMING

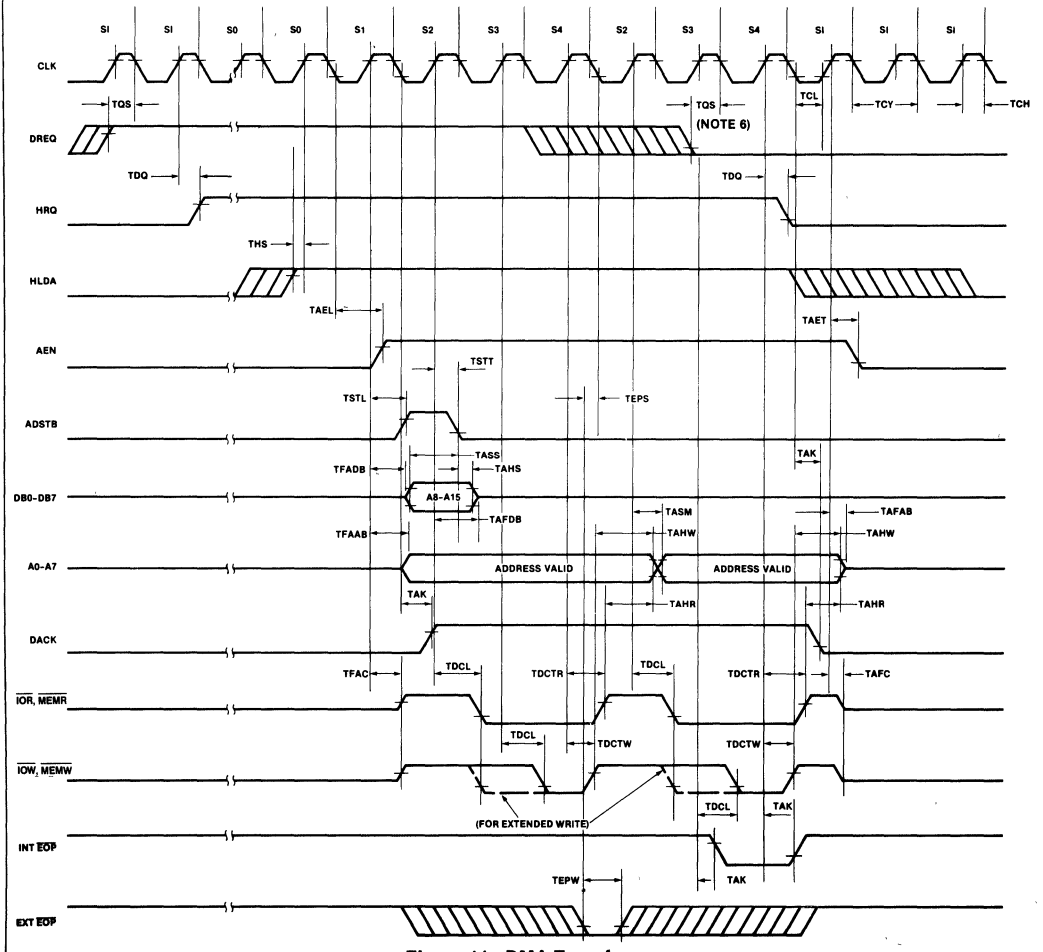


Figure 11. DMA Transfer

WAVEFORMS (Continued)

MEMORY-TO-MEMORY TRANSFER TIMING

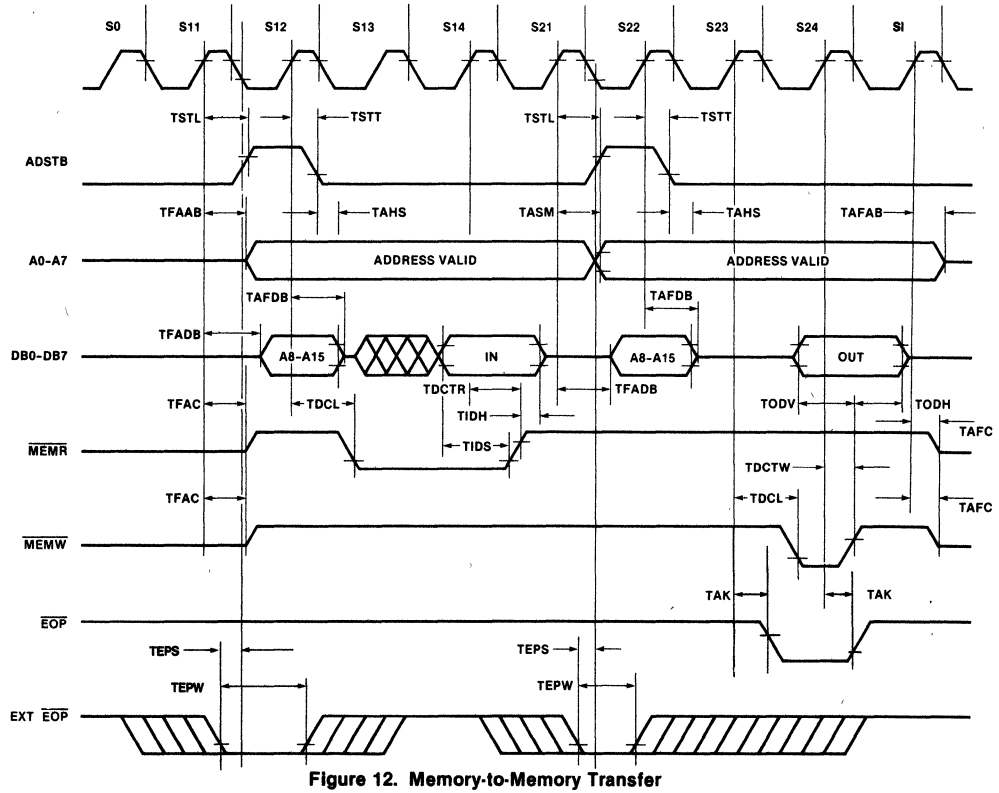


Figure 12. Memory-to-Memory Transfer

READY TIMING

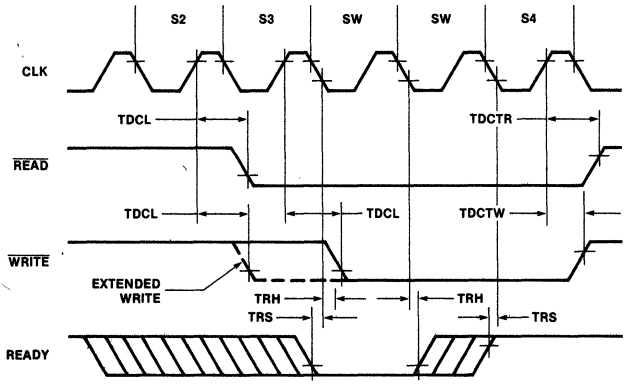


Figure 13. Ready

WAVEFORMS (Continued)

COMPRESSED TRANSFER TIMING

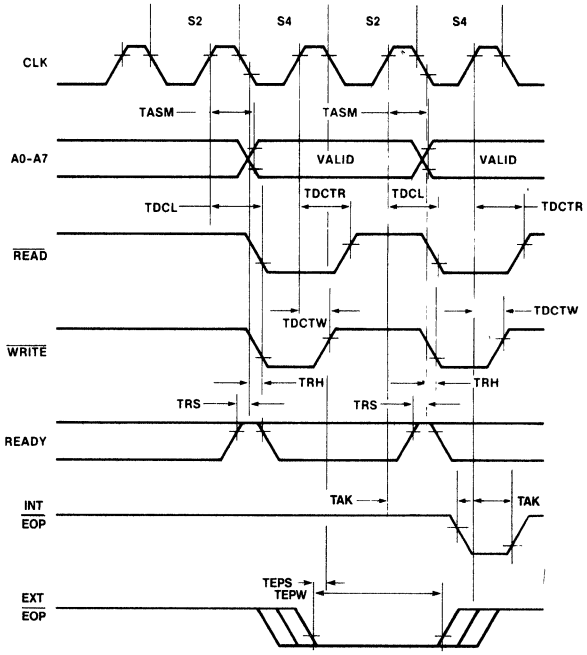


Figure 14. Compressed Transfer

RESET TIMING

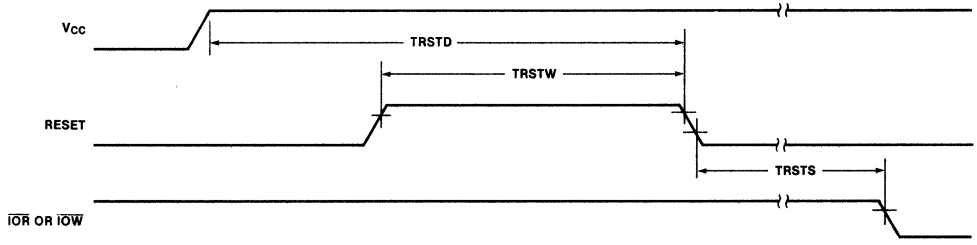


Figure 15. Reset

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.

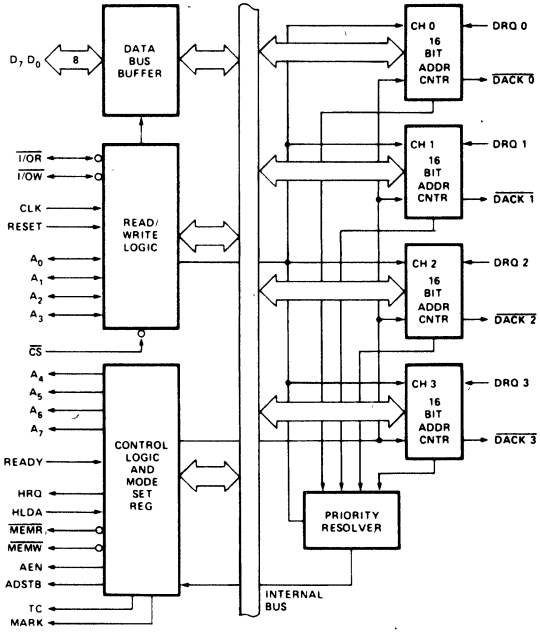


Figure 1. Block Diagram

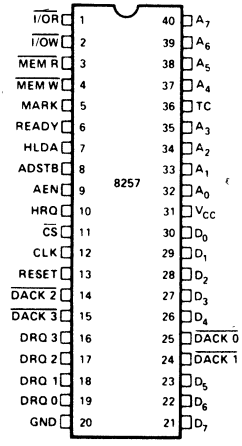


Figure 2. Pin Configuration

FUNCTIONAL DESCRIPTION

General

The 8257 is a programmable, Direct Memory Acess (DMA) device which, when coupled with a single 8-bit latch 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 8-bit latch via the data bus (the outputs of the latch should drive address 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

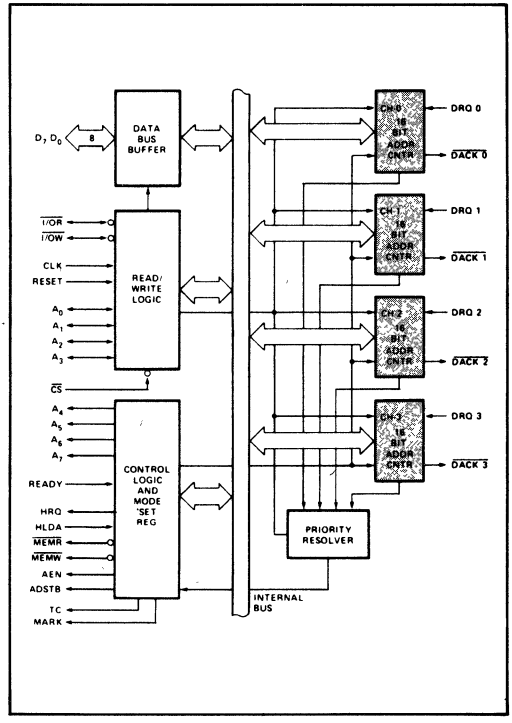


Figure 3. 8257 Block Diagram Showing DMA Channels

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)

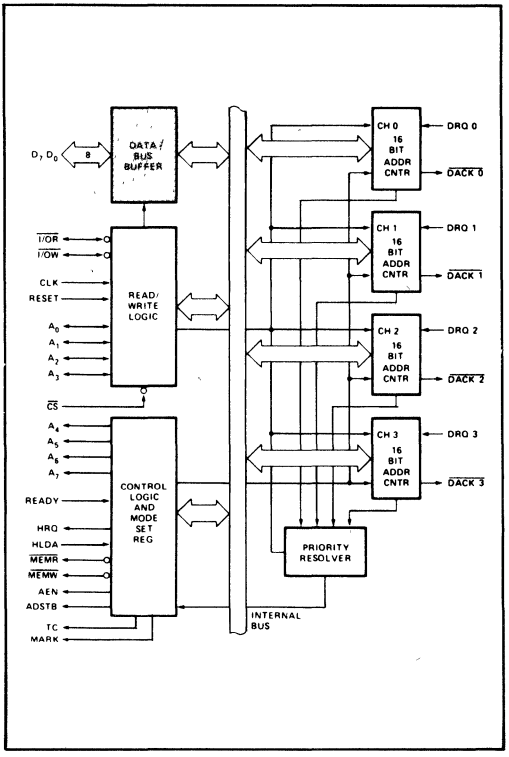


Figure 4. 8257 Block Diagram Showing Data Bus Buffer

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 ® 8085AH 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.

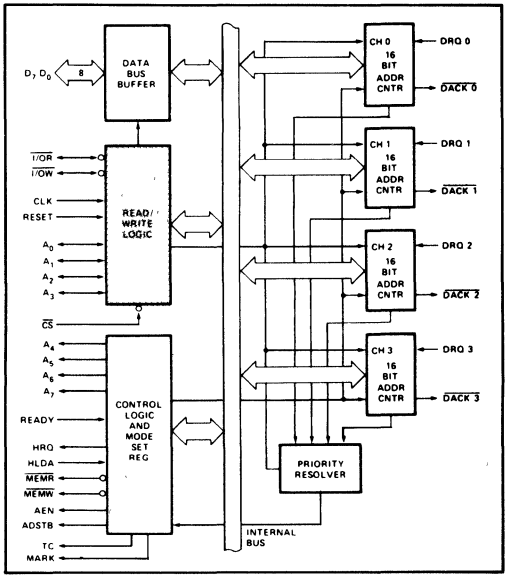


Figure 5. 8257 Block Diagram Showing Read/Write Logic Function

(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, HLDA must remain stable during the specified set-up time.

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

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 latch 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

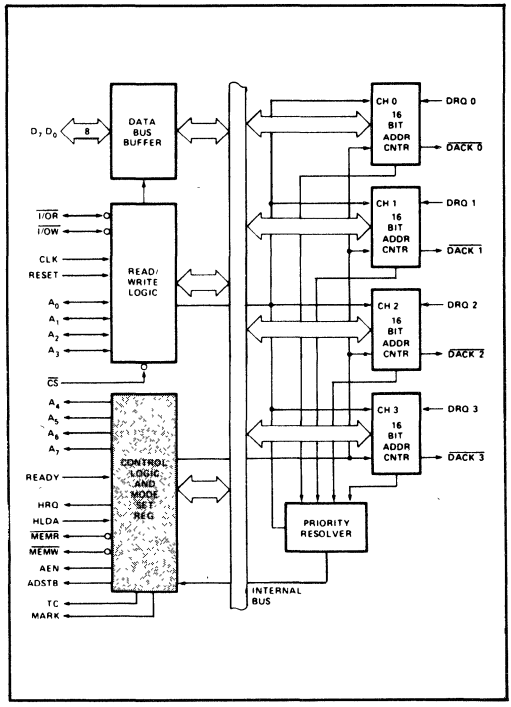
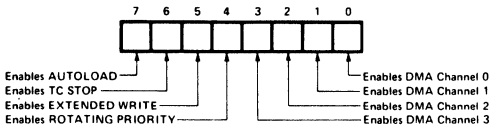


Figure 6. 8257 Block Diagram Showing Control Logic and Mode Set Register

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:

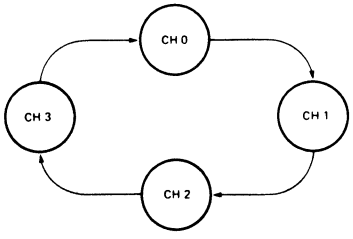


The Mode Set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized. The Mode Set Register is cleared by the RESET input, thus disabling all options, inhibiting all channels, and preventing bus conflicts on power-up. A channel should 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 $\overline{\text{MEMW}}$ and $\overline{\text{I/OW}}$ 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 $\overline{\text{I/OW}}$ or $\overline{\text{MEMW}}$ 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

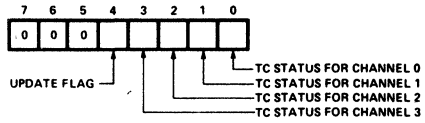
The Auto Load mode permits Channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks Channel 2 registers are initialized as usual for the first data block, Channel 3 registers, however, are used to store the block re-initialization parameters (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.

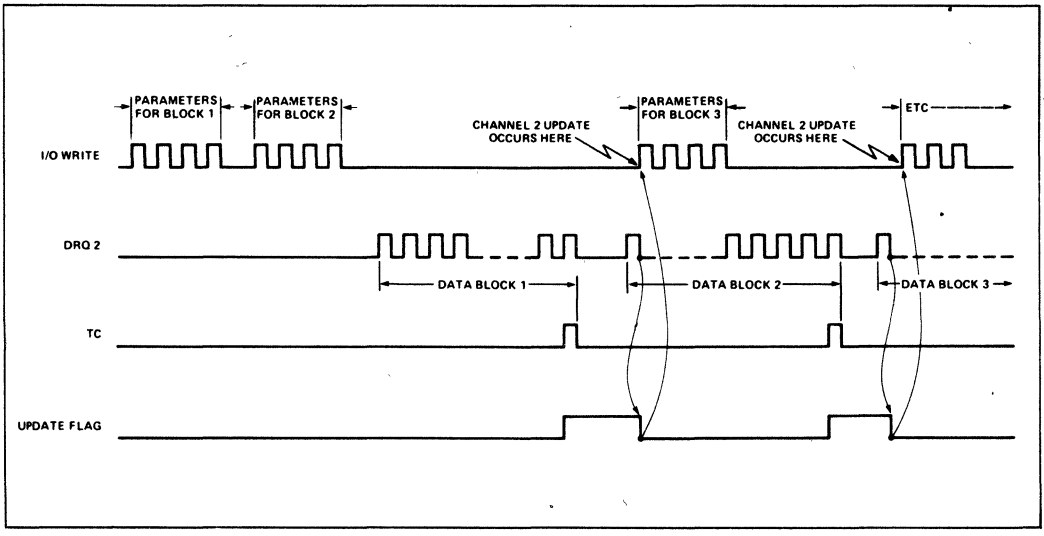


Figure 7. Autoload Timing

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.

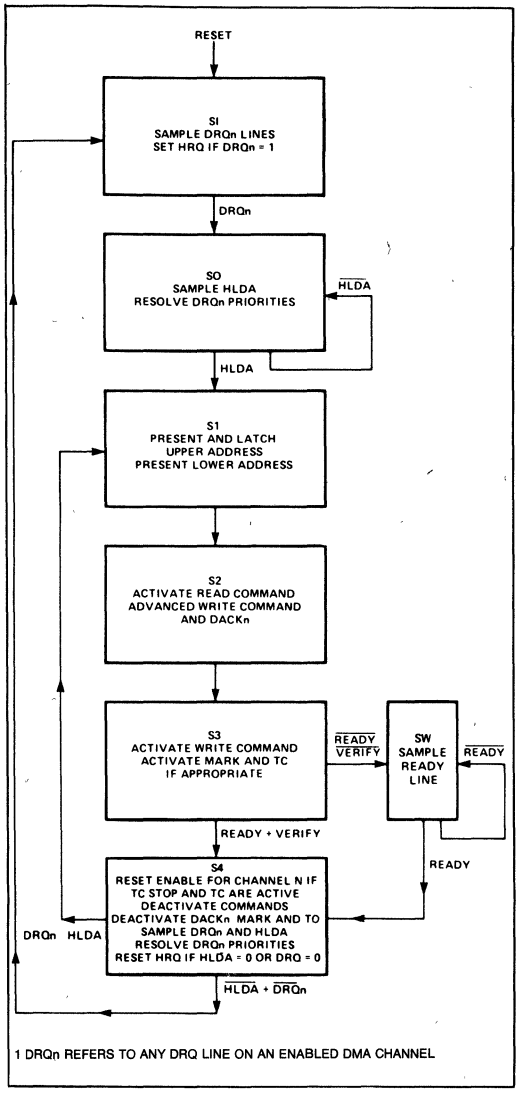


Figure 8. DMA Operation State Diagram

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 8085AH. The Ready line is a 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 of 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

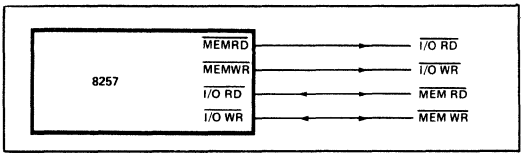


Figure 9. System Interface for Memory Mapped I/O

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

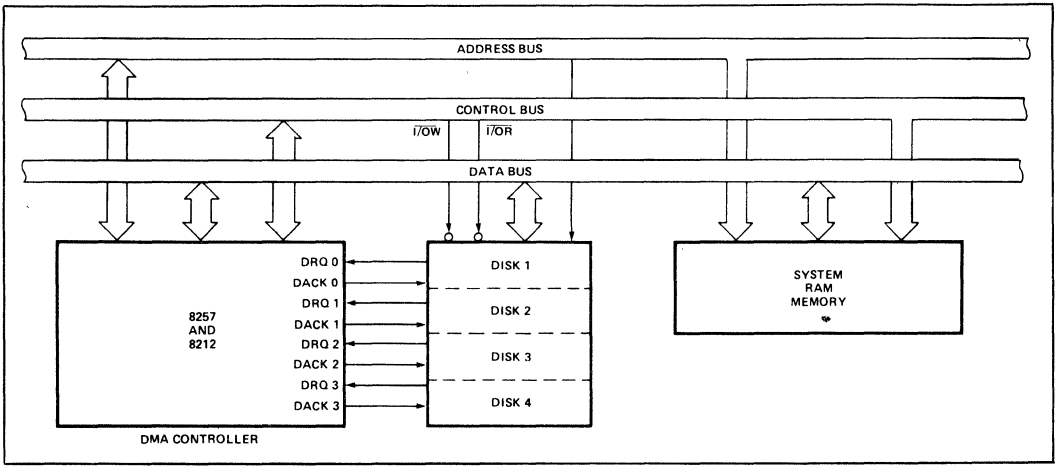


Figure 11. Floppy Disk Controller (4 Drives)

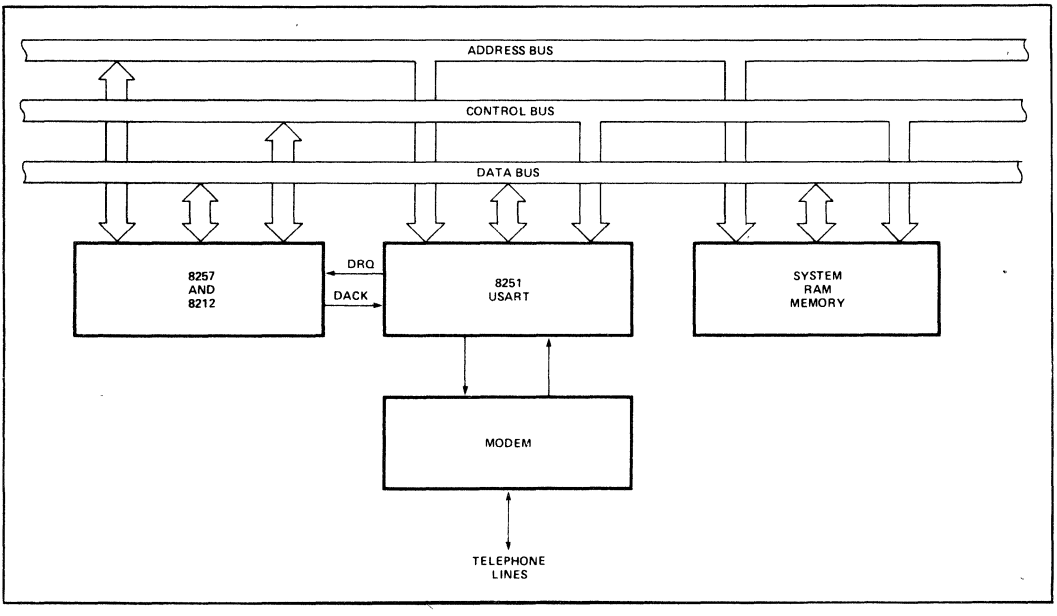
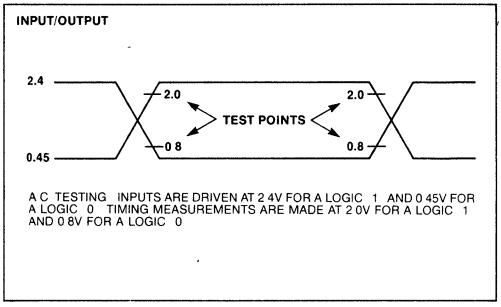


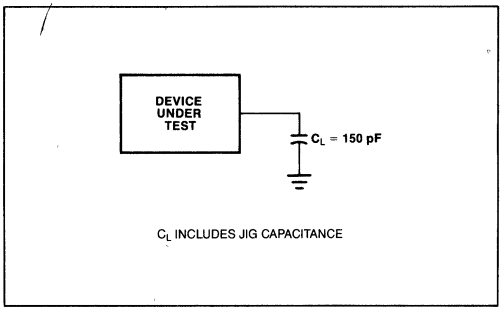
Figure 12. High-Speed Communication Controller

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING 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

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,

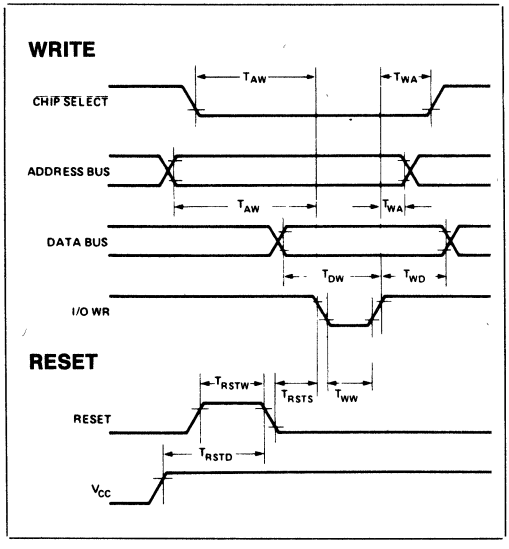
$$T_{A(MIN)} + T_{B(MAX)} \leq 150 \text{ ns}$$

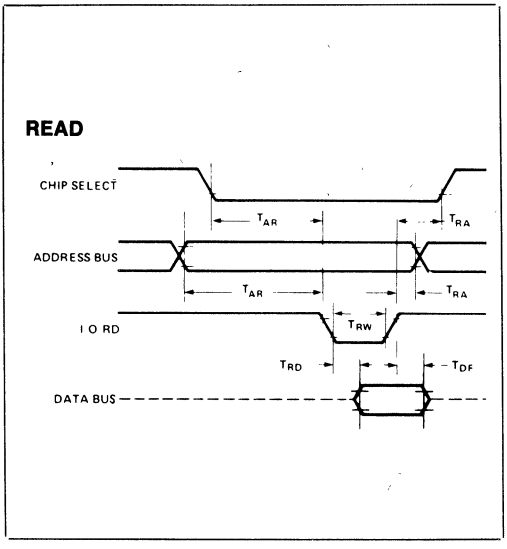
and only minimum specifications exist for T_A and T_B. If $T_{A(MIN)}$ is used, and if T_A and T_B are tracking parameters, $T_{B(MAX)}$ can be taken as $T_{B(MIN)}$ + 50 ns.

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

*if T_A and T_B are tracking parameters

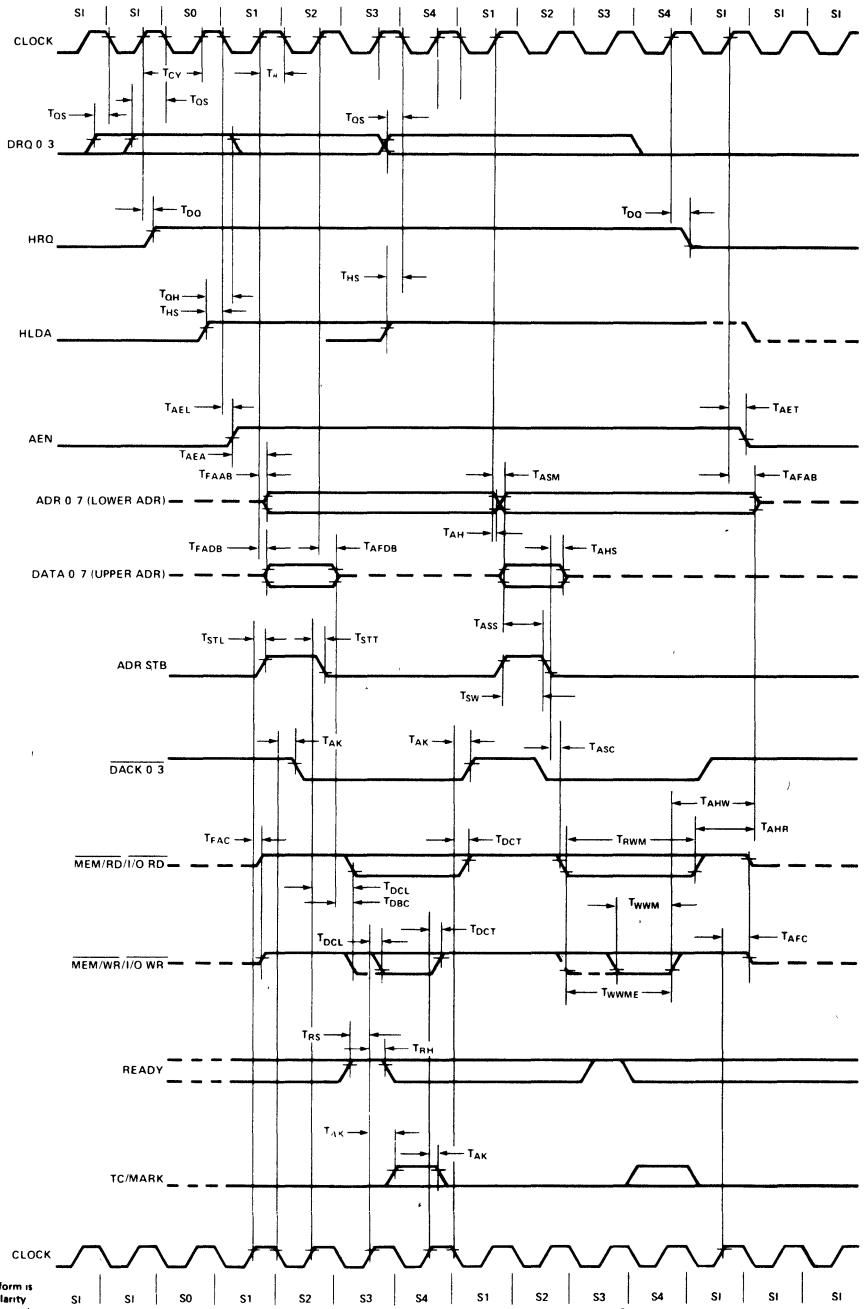
WAVEFORMS—PERIPHERAL MODE





WAVEFORMS—DMA

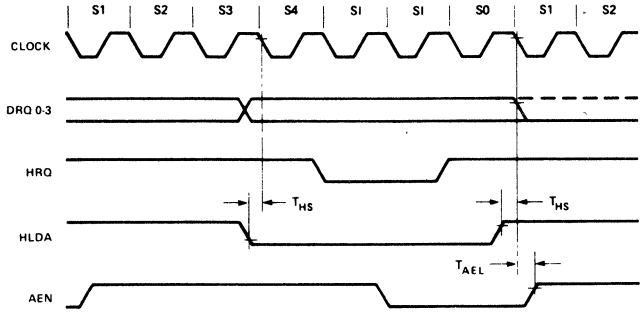
CONSECUTIVE CYCLES AND BURST MODE SEQUENCE



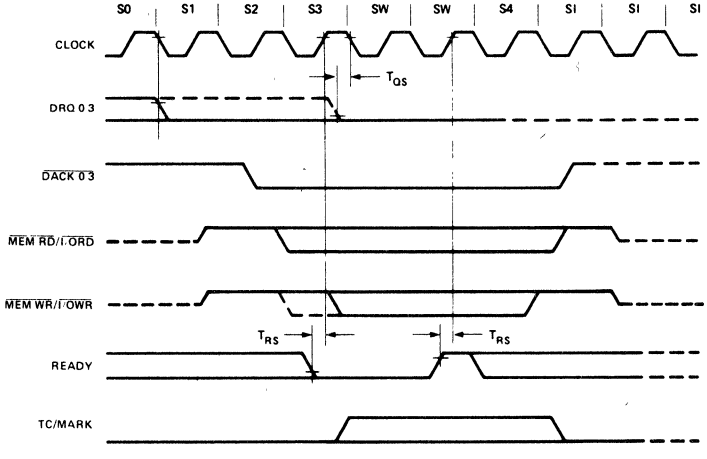
NOTE The clock waveform is duplicated for clarity The 8257 requires only one clock input

WAVEFORMS (Continued)

CONTROL OVERRIDE SEQUENCE



NOT READY SEQUENCE



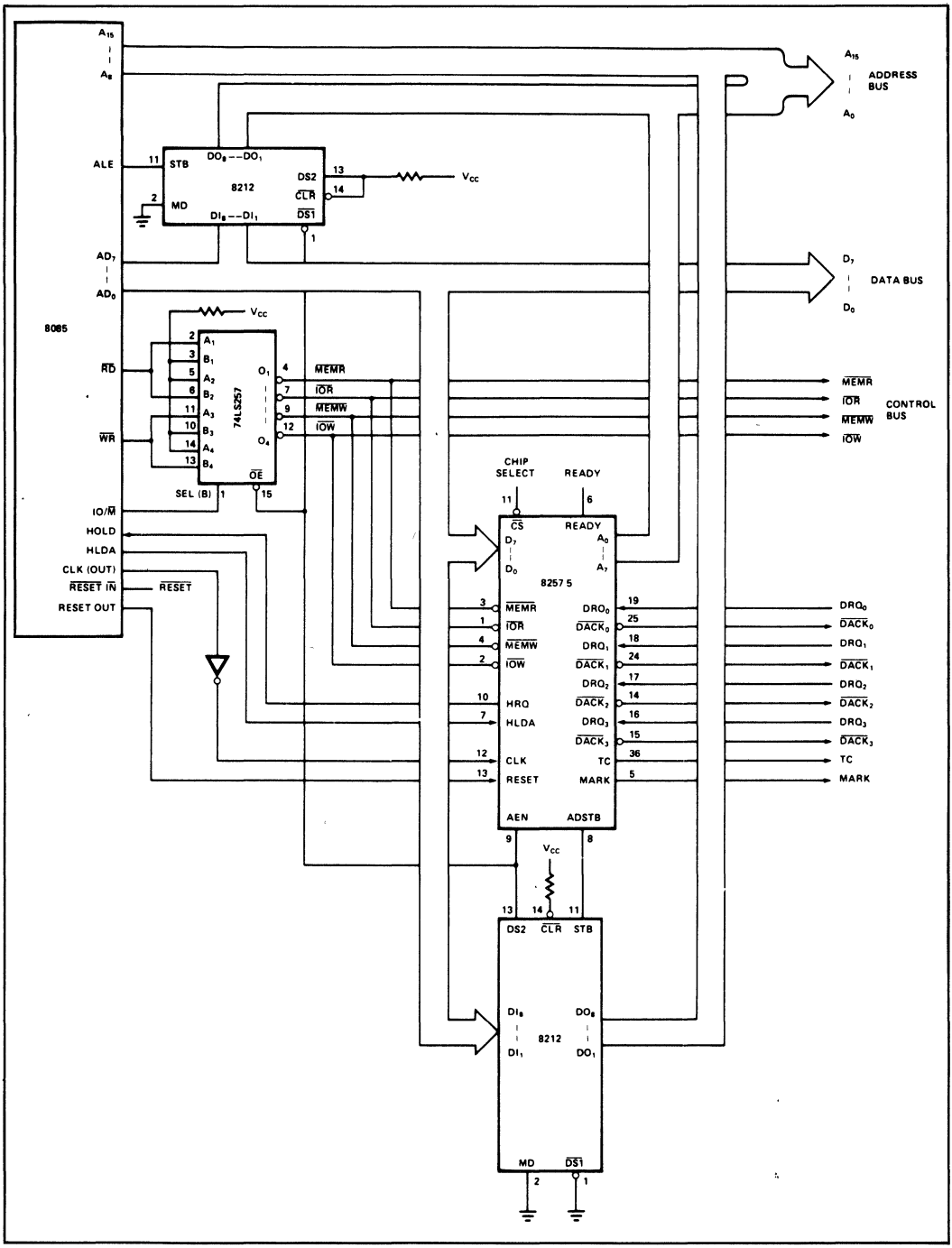


Figure 13. Detailed System Interface Schematic

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_ϕ	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)[7]	100	280	100	280	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.
7. HLDA must remain stable during t_{HS}.

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).

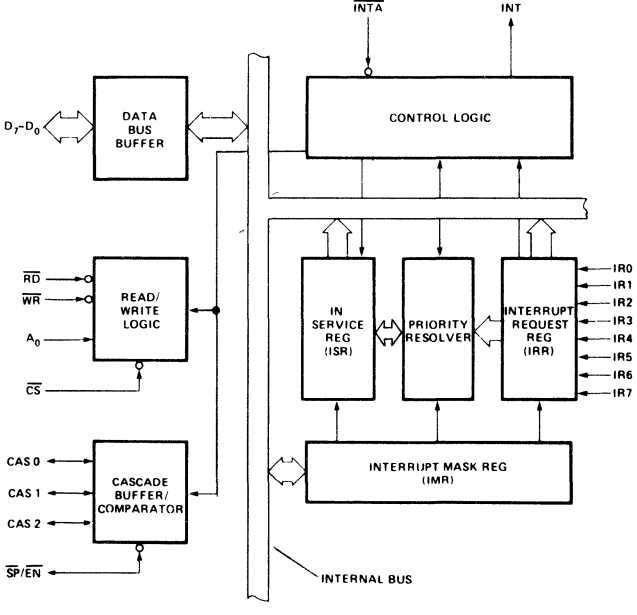


Figure 1. Block Diagram

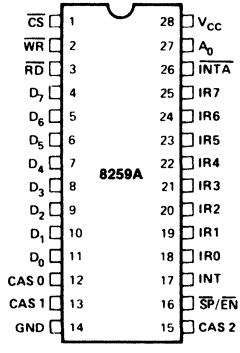


Figure 2. Pin Configuration

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.

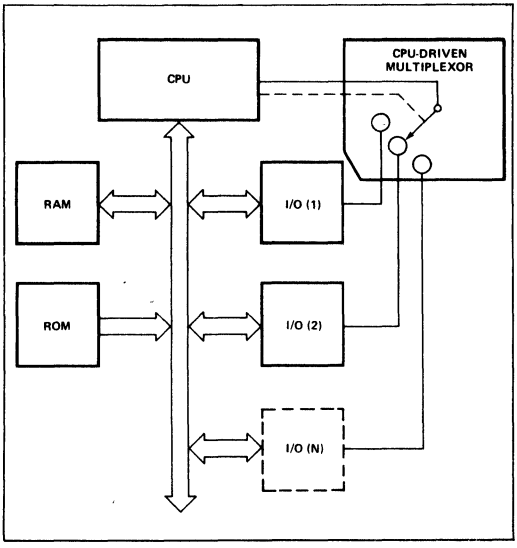


Figure 3a. Polled Method

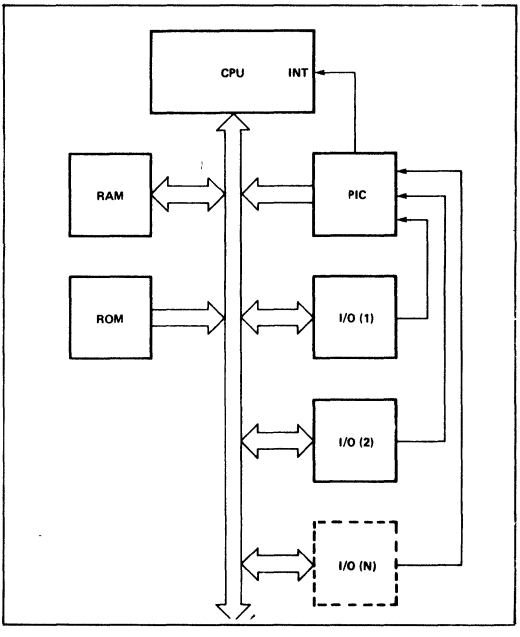


Figure 3b. Interrupt Method

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.

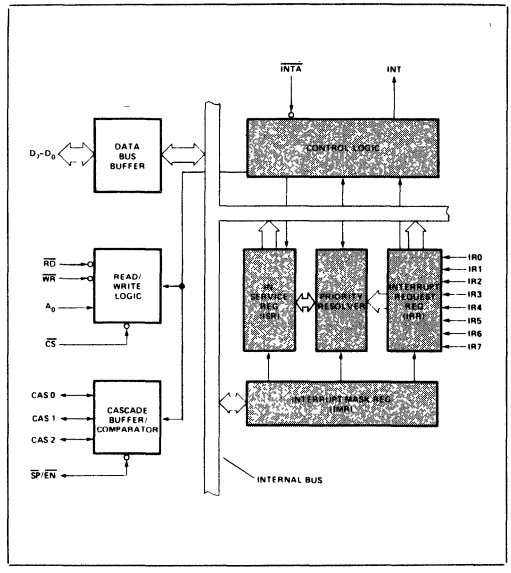


Figure 4a. 8259A Block Diagram

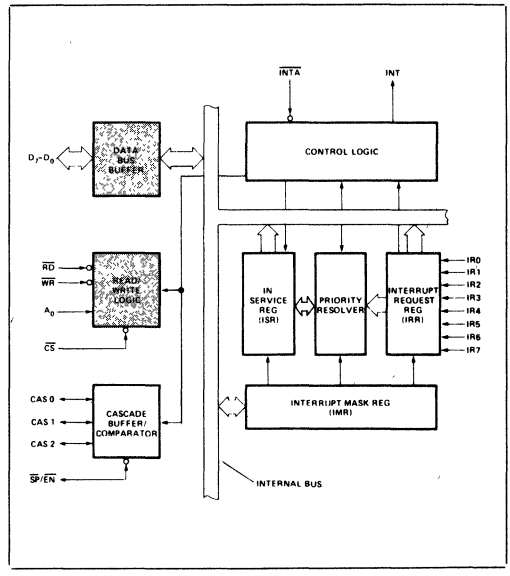


Figure 4b. 8259A Block Diagram

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.

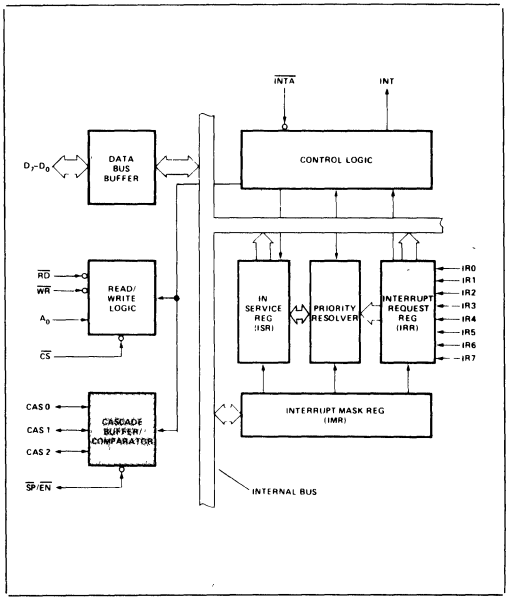


Figure 4c. 8259A Block Diagram

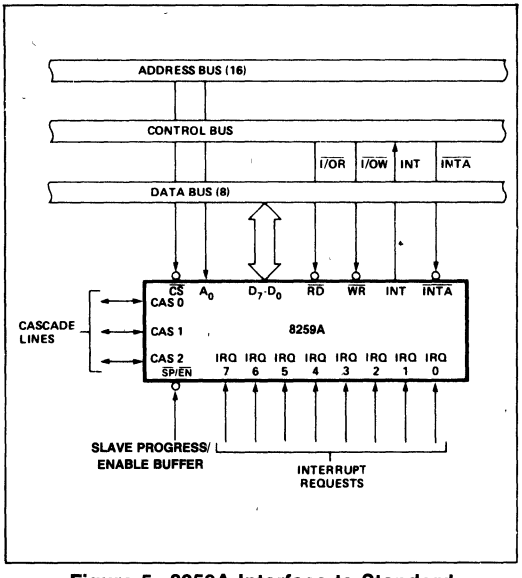


Figure 5. 8259A Interface to Standard System Bus

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.

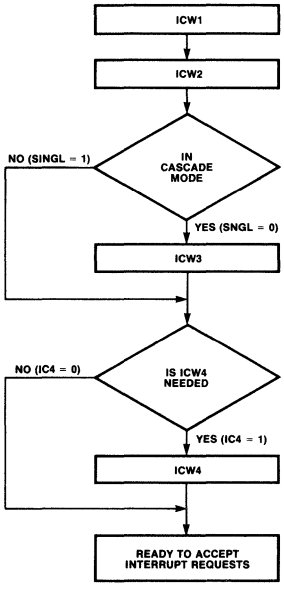
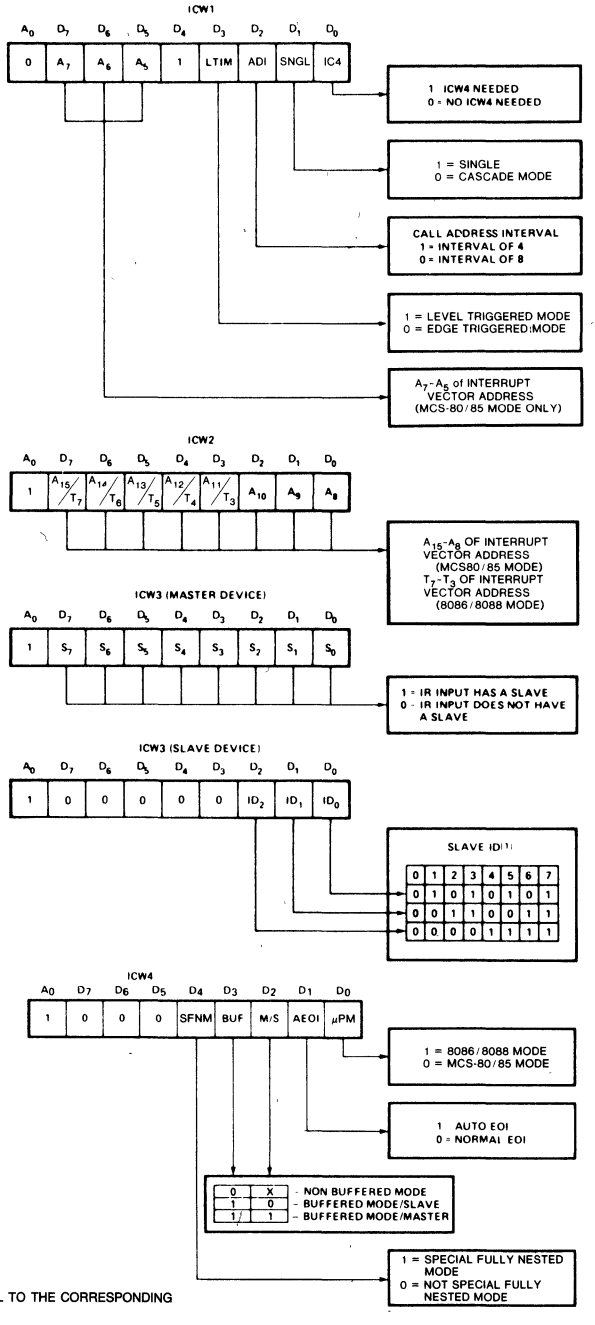


Figure 6. Initialization Sequence



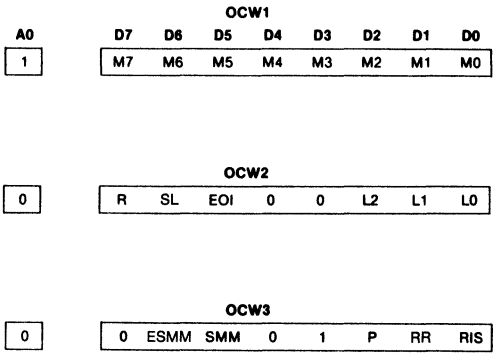
NOTE 1: SLAVE ID IS EQUAL TO THE CORRESPONDING MASTER IR INPUT

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)



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.

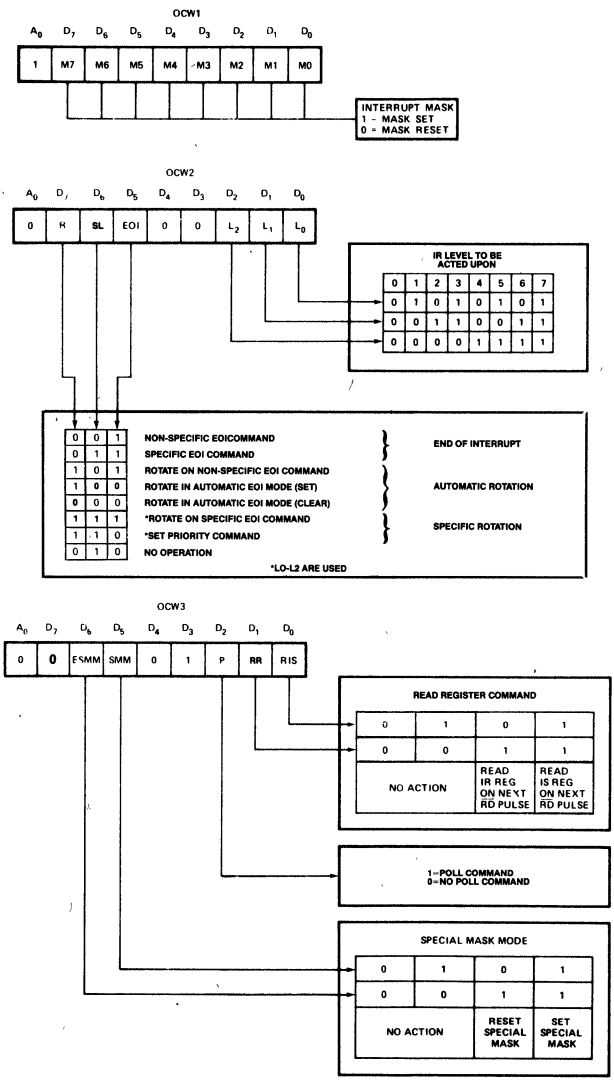


Figure 8. Operation Command Word Format

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)

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	1	0	0	0	0

	Lowest Priority							Highest Priority
Priority Status	7	6	5	4	3	2	1	0

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

	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
"IS" Status	0	1	0	0	0	0	0	0

	Highest Priority							Lowest Priority
Priority Status	2	1	0	7	6	5	4	3

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.

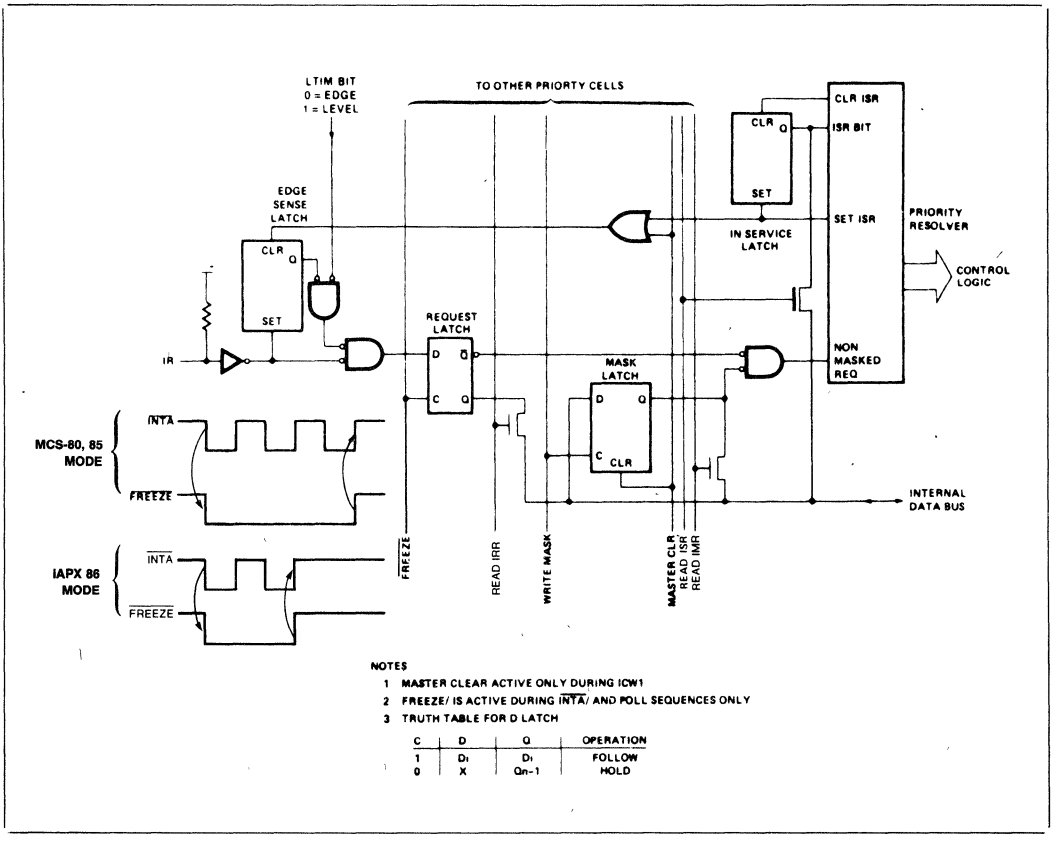


Figure 9. Priority Cell—Simplified Logic Diagram

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.

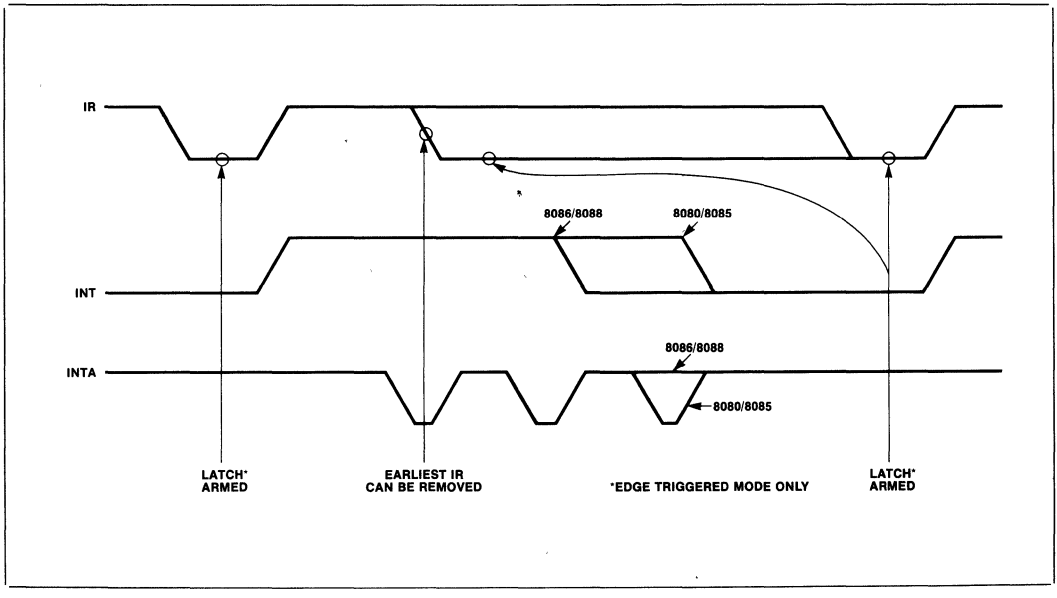


Figure 10. IR Triggering Timing Requirements

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).

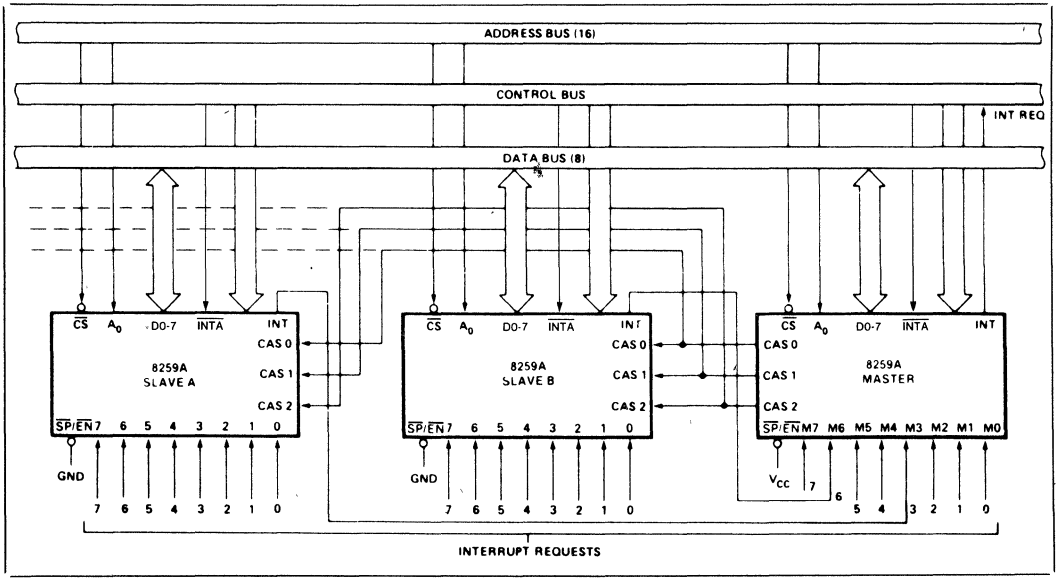


Figure 11. Cascading the 8259A

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 ≤ V_{IN} ≤ V_{CC}
I_{LOL}	Output Leakage Current	−10	+10	μA	0.45V ≤ V_{OUT} ≤ 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.								

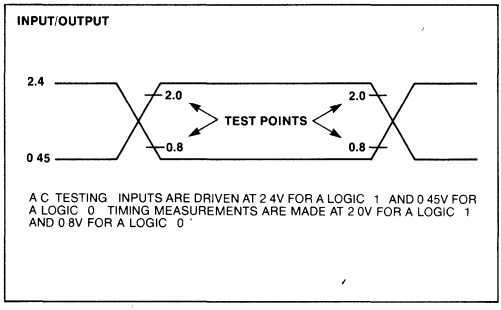
*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 500 ns (i.e. 8085A = 1.6μs, 8085A-2 = 1μs, 8086 = 1μs, 8086-2 = 625 ns)

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

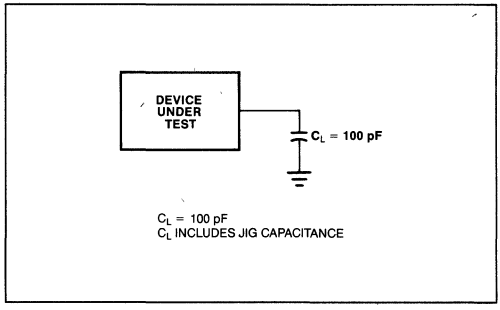
TIMING RESPONSES

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	

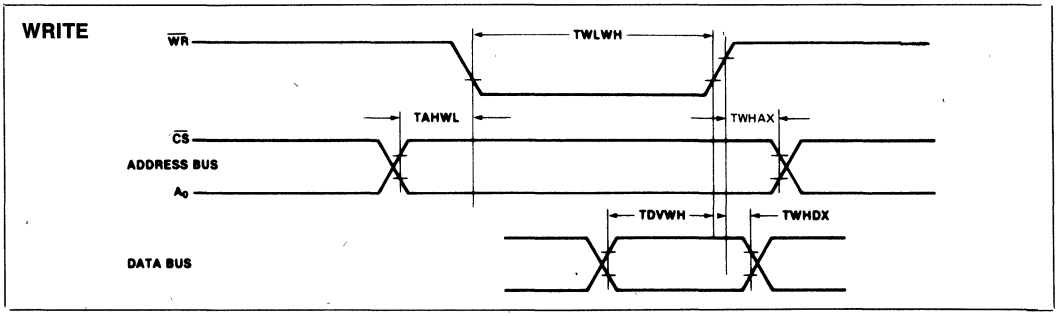
A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING LOAD CIRCUIT

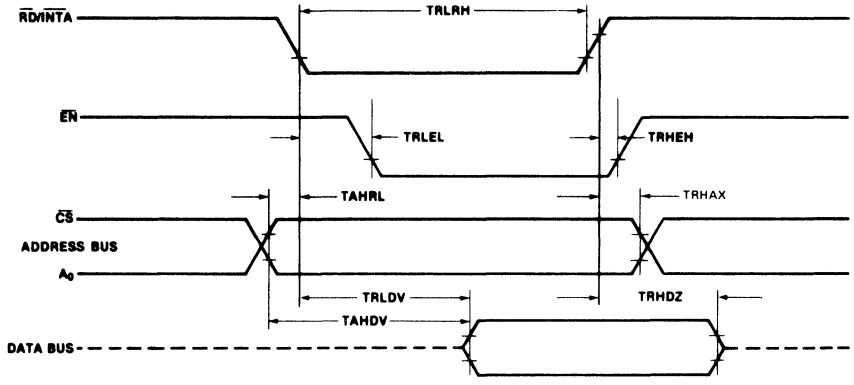


WAVEFORMS

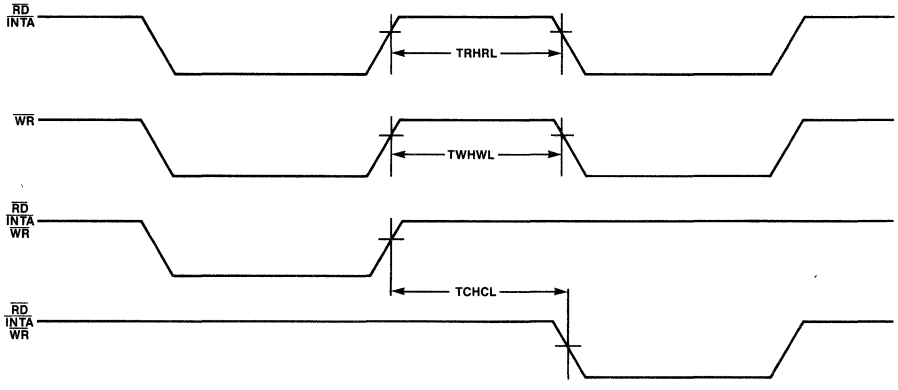


WAVEFORMS (Continued)

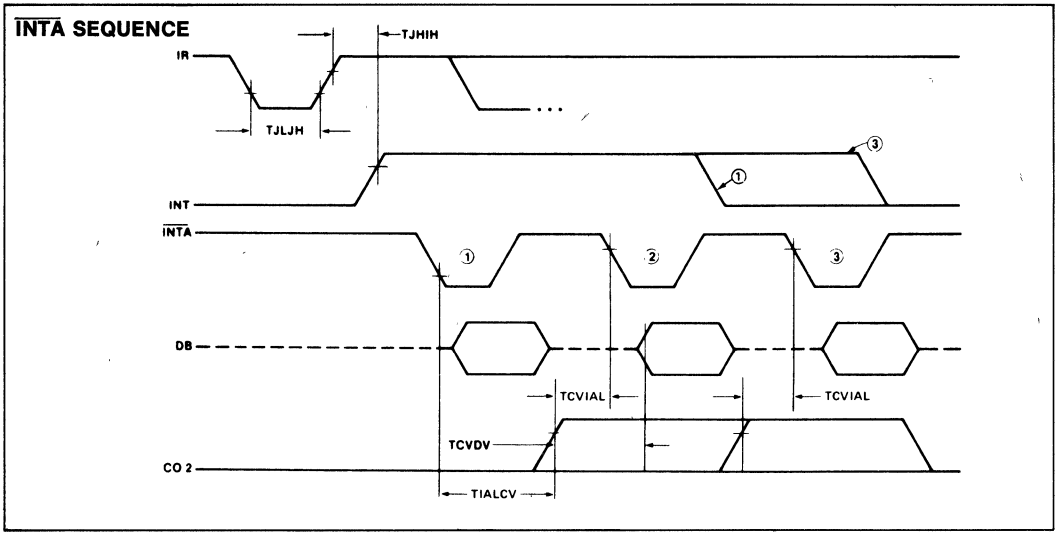
READ/INTA



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.

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 8085AH 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 8085AH 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 8085AH-2 and the 5 MHz iAPX 88 microprocessor.

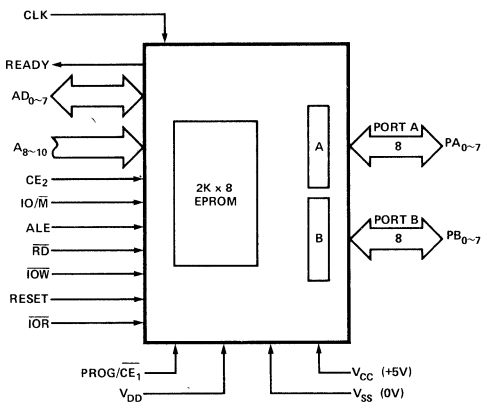


Figure 1. Block Diagram

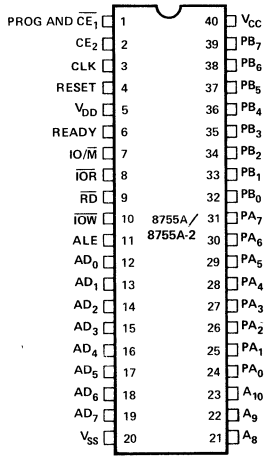


Figure 2. Pin Configuration

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.
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

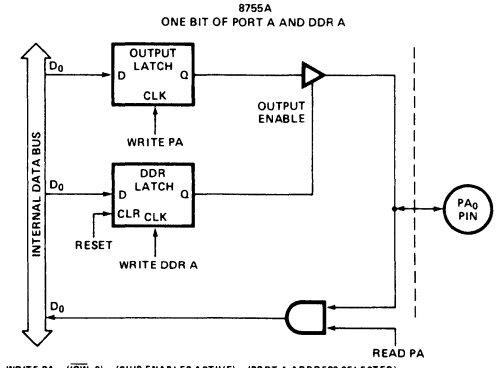
The I/O section of the chip is addressed by the latched value of AD_{0-1}. Two 8-bit Data Direction Registers (DDR) in 8755A determine the input/output status of each pin in the corresponding ports. A "0" in a particular bit position of a DDR signifies that the corresponding I/O port bit is in the input 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 8085AH 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 8085AH 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.

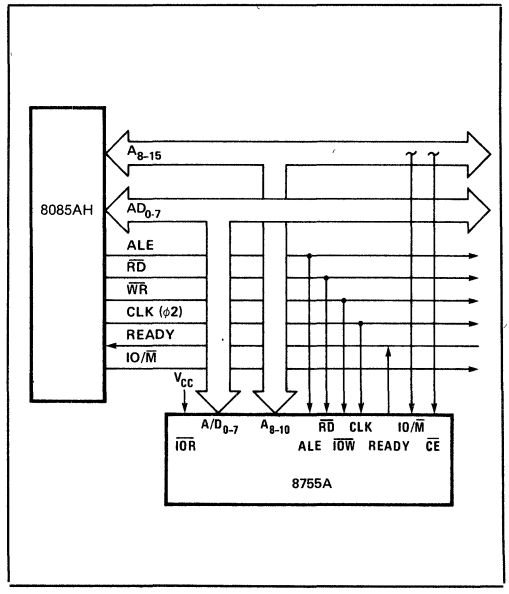


Figure 3. 8755A in 8085AH System (Memory-Mapped I/O)

iAPX 88 FIVE CHIP SYSTEM

Figure 4 shows a five chip system containing:

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

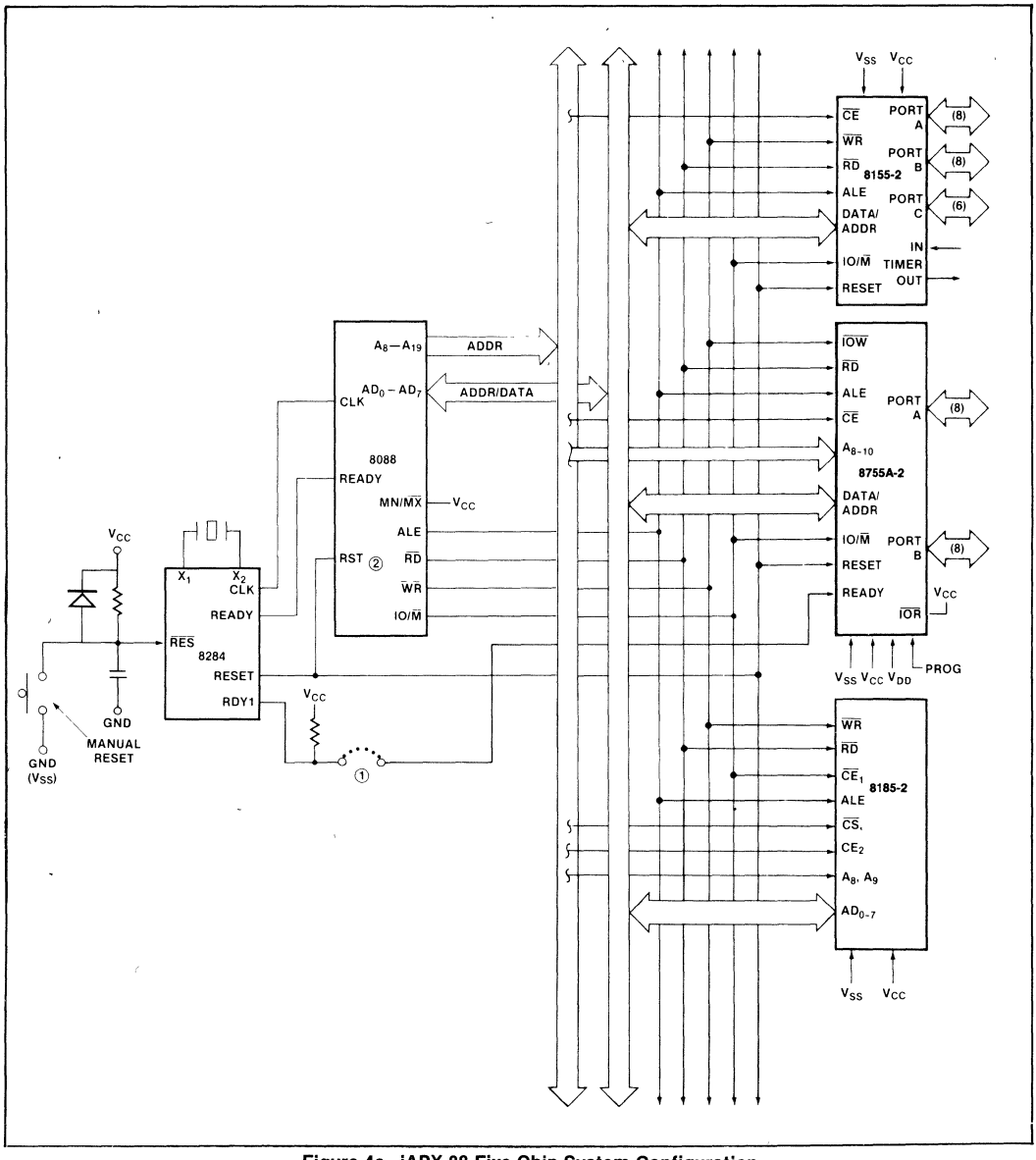
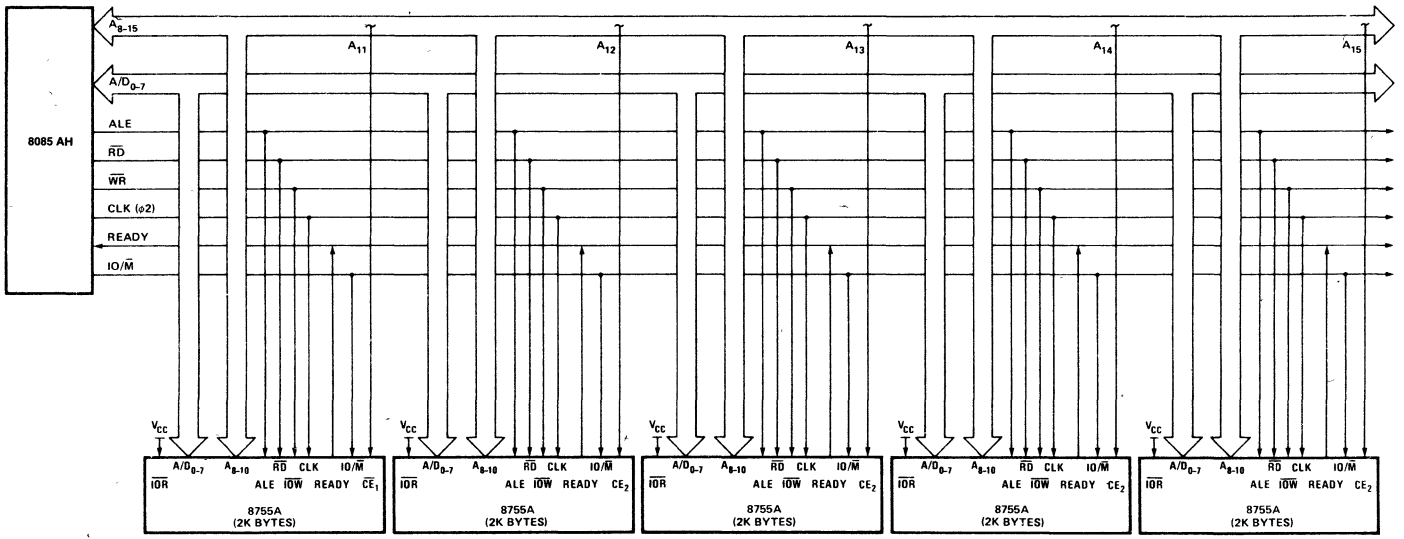


Figure 4a. iAPX 88 Five Chip System Configuration



Note: Use $\overline{CE}_1$ for the first 8755A in the system, and CE_2 for the other 8755A's. Permits up to 5-8755A's in a system without CE decoder.

Figure 4b. 8755A 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°, 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.

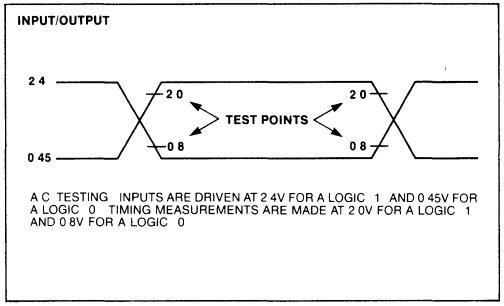
*Or $T_{AD} - (T_{AL} + T_{LC})$, whichever is greater.

**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)

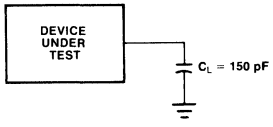
Symbol	Parameter	Min.	Typ.	Max.	Unit
t_{PS}	Data Setup Time	10			ns
t_{PD}	Data Hold Time	0			ns
t_S	Prog Pulse Setup Time	2			μs
t_H	Prog Pulse Hold Time	2			μs
t_{PR}	Prog Pulse Rise Time	0.01	2		μs
t_{PF}	Prog Pulse Fall Time	0.01	2		μs
t_{PRG}	Prog Pulse Width	45	50		msec

A.C. TESTING INPUT, OUTPUT WAVEFORM



A C TESTING 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

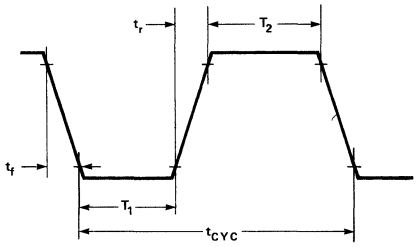
A.C. TESTING LOAD CIRCUIT



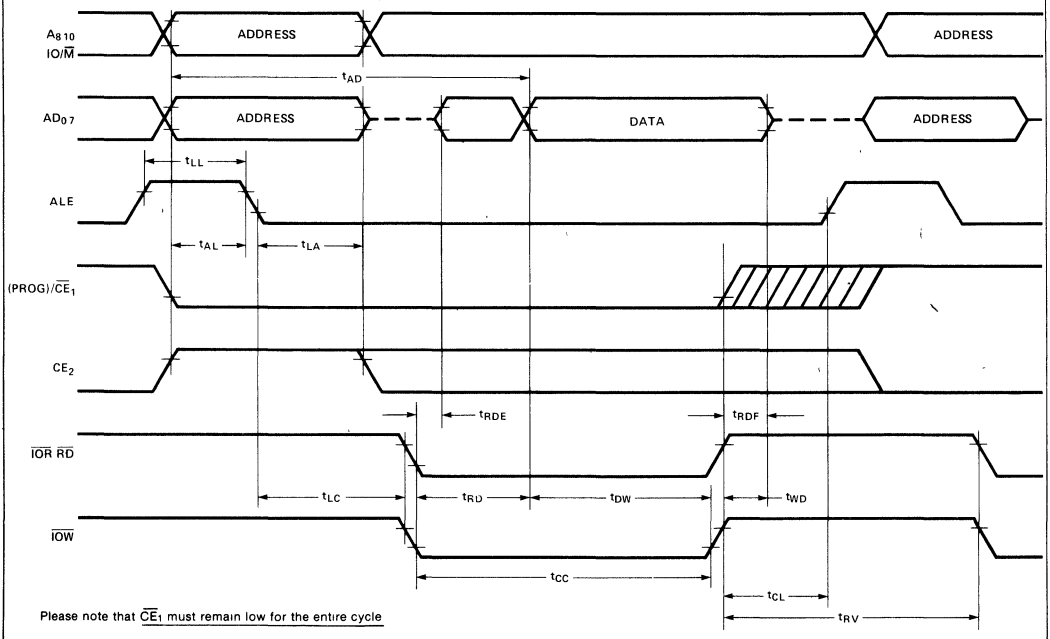
C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

CLOCK SPECIFICATION FOR 8755A



PROM READ, I/O READ AND WRITE

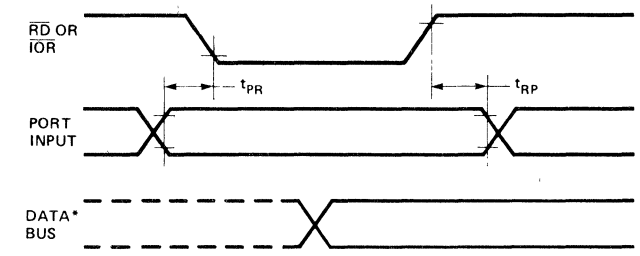


Please note that $\overline{CE_1}$ must remain low for the entire cycle

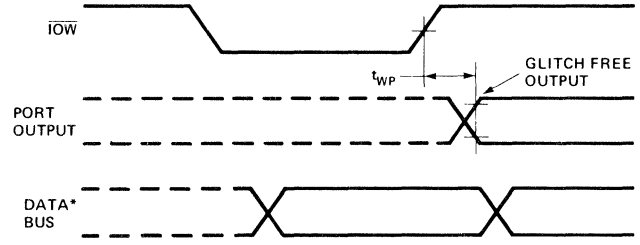
WAVEFORMS (Continued)

I/O PORT

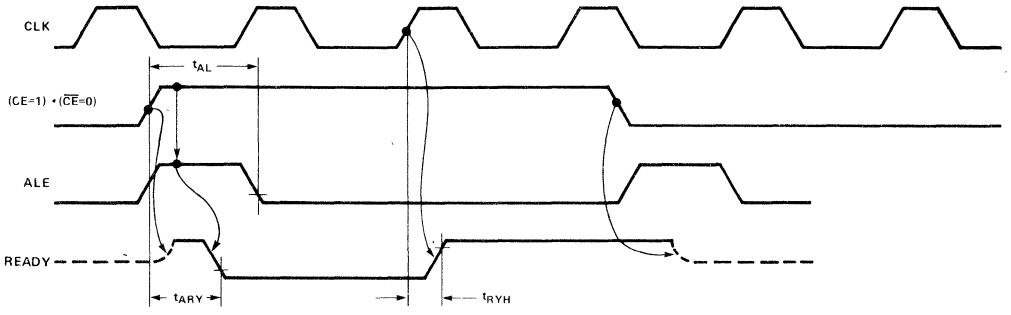
A. INPUT MODE



B. OUTPUT MODE

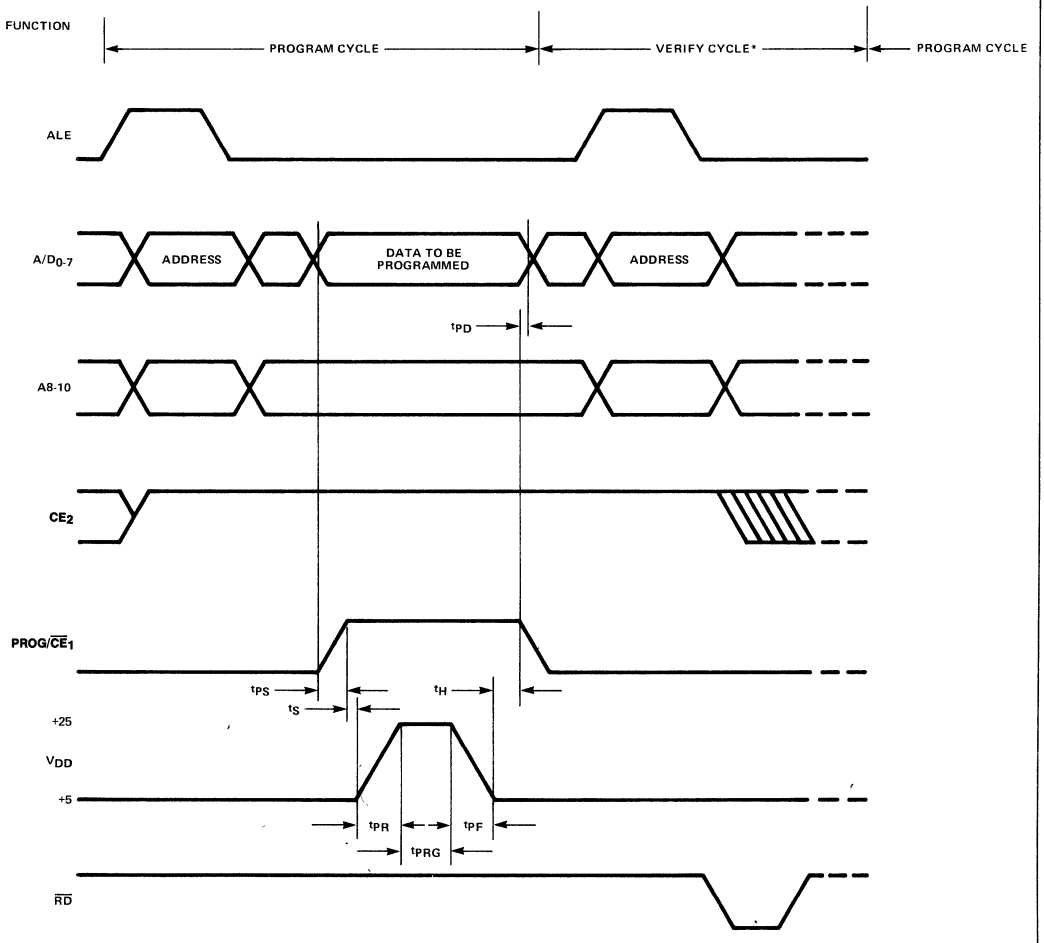


WAIT STATE (READY = 0)



WAVEFORMS (Continued)

8755A PROGRAM MODE



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

intel®

APPLICATION NOTE

AP-59

September 1979

Using the 8259A Programmable Interrupt Controller

Robin Jigour
Microcomputer Applications

121500-001

INTRODUCTION

The Intel 8259A is a Programmable Interrupt Controller (PIC) designed for use in real-time interrupt driven microcomputer systems. The 8259A manages eight levels of interrupts and has built-in features for expansion up to 64 levels with additional 8259A's. Its versatile design allows it to be used within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. Being fully programmable, the 8259A provides a wide variety of modes and commands to tailor 8259A interrupt processing for the specific needs of the user. These modes and commands control a number of interrupt oriented functions such as interrupt priority selection and masking of interrupts. The 8259A programming may be dynamically changed by the software at any time, thus allowing complete interrupt control throughout program execution.

The 8259A is an enhanced, fully compatible revision of its predecessor, the 8259. This means the 8259A can use all hardware and software originally designed for the 8259 without any changes. Furthermore, it provides additional modes that increase its flexibility in MCS-80 and MCS-85 systems and allow it to work in MCS-86 and MCS-88 systems. These modes are:

- MCS-86/88 Mode
- Automatic End of Interrupt Mode
- Level Triggered Mode
- Special Fully Nested Mode
- Buffered Mode

Each of these are covered in depth further in this application note.

This application note was written to explain completely how to use the 8259A within MCS-80, MCS-85, MCS-86, and MCS-88 microcomputer systems. It is divided into five sections. The first section, "Concepts", explains the concepts of interrupts and presents an overview of how the 8259A works with each microcomputer system mentioned above. The second section, "Functional Block Diagram", describes the internal functions of the 8259A in block diagram form and provides a detailed functional description of each device pin. "Operation of the 8259A", the third section, explains in depth the operation and use of each of the 8259A modes and commands. For clarity of explanation, this section doesn't make reference to the actual programming of the 8259A. Instead, all programming is covered in the fourth section, "Programming the 8259A". This section explains how to program the 8259A with the modes and commands mentioned in the previous section. These two sections are referenced in Appendix A. The fifth and final section "Application Examples", shows the 8259A in three typical applications. These applications are fully explained with reference to both hardware and software.

The reader should note that some of the terminology used throughout this application note may differ slightly from existing data sheets. This is done to better clarify and explain the operation and programming of the 8259A.

1. CONCEPTS

In microcomputer systems there is usually a need for the processor to communicate with various Input/Output (I/O) devices such as keyboards, displays, sensors, and other peripherals. From the system viewpoint, the processor should spend as little time as possible servicing the peripherals since the time required for these I/O chores directly affects the amount of time available for other tasks. In other words, the system should be designed so that I/O servicing has little or no effect on the total system throughput. There are two basic methods of handling the I/O chores in a system: status polling and interrupt servicing.

The status poll method of I/O servicing essentially involves having the processor "ask" each peripheral if it needs servicing by testing the peripheral's status line. If the peripheral requires service, the processor branches to the appropriate service routine; if not, the processor continues with the main program. Clearly, there are several problems in implementing such an approach. First, how often a peripheral is polled is an important constraint. Some idea of the "frequency-of-service" required by each peripheral must be known and any software written for the system must accommodate this time dependence by "scheduling" when a device is polled. Second, there will obviously be times when a device is polled that is not ready for service, wasting the processor time that it took to do the poll. And other times, a ready device would have to wait until the processor "makes its rounds" before it could be serviced, slowing down the peripheral.

Other problems arise when certain peripherals are more important than others. The only way to implement the "priority" of devices is to poll the high priority devices more frequently than lower priority ones. It may even be necessary to poll the high priority devices while in a low priority device service routine. It is easy to see that the polled approach can be inefficient both time-wise and software-wise. Overall, the polled method of I/O servicing can have a detrimental effect on system throughput, thus limiting the tasks that can be performed by the processor.

A more desirable approach in most systems would allow the processor to be executing its main program and only stop to service the I/O when told to do so by the I/O itself. This is called the interrupt service method. In effect, the device would asynchronously signal the processor when it required service. The processor would finish its current instruction and then vector to the service routine for the device requesting service. Once the service routine is complete, the processor would resume exactly where it left off. Using the interrupt service method, no processor time is spent testing devices, scheduling is not needed, and priority schemes are readily implemented. It is easy to see that, using the interrupt service approach, system throughput would increase, allowing more tasks to be handled by the processor.

However, to implement the interrupt service method between processor and peripherals, additional hardware is usually required. This is because, after interrupting the processor, the device must supply information for vectoring program execution. Depending on the processor used, this can be accomplished by the device taking control of the data bus and "jamming" an instruction(s) onto it. The instruction(s) then vectors the pro-

gram to the proper service routine. This of course requires additional control logic for each interrupt requesting device. Yet the implementation so far is only in the most basic form. What if certain peripherals are to be of higher priority than others? What if certain interrupts must be disabled while others are to be enabled? The possible variations go on, but they all add up to one theme; to provide greater flexibility using the interrupt service method, hardware requirements increase.

So, we're caught in the middle. The status poll method is a less desirable way of servicing I/O in terms of throughput, but its hardware requirements are minimal. On the other hand, the interrupt service method is most desirable in terms of flexibility and throughput, but additional hardware is required.

The perfect situation would be to have the flexibility and throughput of the interrupt method in an implementation with minimal hardware requirements. The 8259A Programmable Interrupt Controller (PIC) makes this all possible.

The 8259A Programmable Interrupt Controller (PIC) was designed to function as an overall manager of an interrupt driven system. No additional hardware is required. The 8259A alone can handle eight prioritized interrupt levels, controlling the complete interface between peripherals and processor. Additional 8259A's can be "cascaded" to increase the number of interrupt levels processed. A wide variety of modes and commands for programming the 8259A give it enough flexibility for almost any interrupt controlled structure. Thus, the 8259A is the feasible answer to handling I/O servicing in microcomputer systems.

Now, before explaining exactly how to use the 8259A, let's go over interrupt structures of the MCS-80, MCS-85, MCS-86, and MCS-88 systems, and how they interact with the 8259A. Figure 1 shows a block diagram of the 8259A interfacing with a standard system bus. This may prove useful as reference throughout the rest of the "Concepts" section.

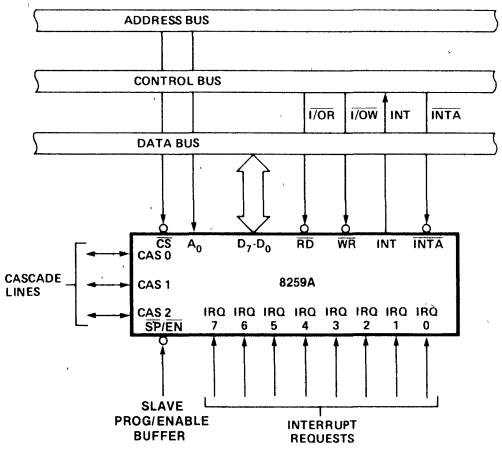


Figure 1. 8259A Interface to Standard System Bus

1.1 MCS-80 —8259A OVERVIEW

In an MCS-80—8259A interrupt configuration, as in Figure 2, a device may cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0-IR7) high. If the 8259A accepts the interrupt request (this depends on its programmed condition), the 8259A's INT (interrupt) pin will go high, driving the 8080A's INT pin high.

The 8080A can receive an interrupt request any time, since its INT input is asynchronous. The 8080A, however, doesn't always have to acknowledge an interrupt request immediately. It can accept or disregard requests under software control using the EI (Enable Interrupt) or DI (Disable Interrupt) instructions. These instructions either set or reset an internal interrupt enable flip-flop. The output of this flip-flop controls the state of the INTE (Interrupt Enabled) pin. Upon reset, the 8080A interrupts are disabled, making INTE low.

At the end of each instruction cycle, the 8080A examines the state of its INT pin. If an interrupt request is present and interrupts are enabled, the 8080A enters an interrupt machine cycle. During the interrupt machine cycle the 8080A resets the internal interrupt enable flip-flop, disabling further interrupts until an EI instruction is executed. Unlike normal machine cycles, the interrupt machine cycle doesn't increment the program counter. This ensures that the 8080A can return to the pre-interrupt program location after the interrupt is completed. The 8080A then issues an $\overline{\text{INTA}}$ (Interrupt Acknowledge) pulse via the 8228 System Controller Bus Driver. This $\overline{\text{INTA}}$ pulse signals the 8259A that the 8080A is honoring the request and is ready to process the interrupt.

The 8259A can now vector program execution to the corresponding service routine. This is done during a sequence of the three $\overline{\text{INTA}}$ pulses from the 8080A via the 8228. Upon receiving the first $\overline{\text{INTA}}$ pulse the 8259A places the opcode for a CALL instruction on the data bus. This causes the contents of the program counter to be pushed onto the stack. In addition, the CALL instruction causes two more $\overline{\text{INTA}}$ pulses to be issued, allowing the 8259A to place onto the data bus the starting address of the corresponding service routine. This address is called the interrupt-vector address. The lower 8 bits (LSB) of the interrupt-vector address are released during the second $\overline{\text{INTA}}$ pulse and the upper 8 bits (MSB) during the third $\overline{\text{INTA}}$ pulse. Once this sequence is completed, program execution then vectors to the service routine at the interrupt-vector address.

If the same registers are used by both the main program and the interrupt service routine, their contents should be saved when entering the service routine. This includes the Program Status Word (PSW) which consists of the accumulator and flags. The best way to do this is to "PUSH" each register used onto the stack. The service routine can then "POP" each register off the stack in the reverse order when it is completed. This prevents any ambiguous operation when returning to the main program.

Once the service routine is completed, the main program may be re-entered by using a normal RET (Return) instruction. This will "POP" the original con-

tents of the program counter back off the stack to resume program execution where it left off. Note, that because interrupts are disabled during the interrupt acknowledge sequence, the EI instruction must be executed either during the service routine or the main program before further interrupts can be processed.

For additional information on the 8080A interrupt structure and operation, refer to the MCS-80 User's Manual.

1.2 MCS-85 —8259A OVERVIEW

An MCS-85—8259A configuration processes interrupts in much the same format as an MCS-80—8259A configuration. When an interrupt occurs, a sequence of three $\overline{INTA}$ pulses causes the 8259A to release onto the data bus a CALL instruction and an interrupt-vector address for the corresponding service routine. Other events that occur during the 8080A interrupt machine cycle, such as disabling interrupts and not incrementing the program counter, also occur in the 8085A interrupt acknowledge machine cycle. Additionally, the instructions for saving registers, enabling or disabling of interrupts, and returning from service routines are literally the same.

The 8085A, however, has a different interrupt hardware scheme as shown in Figure 3. For one, the 8085A supplies its own $\overline{INTA}$ output pin rather than using an addi-

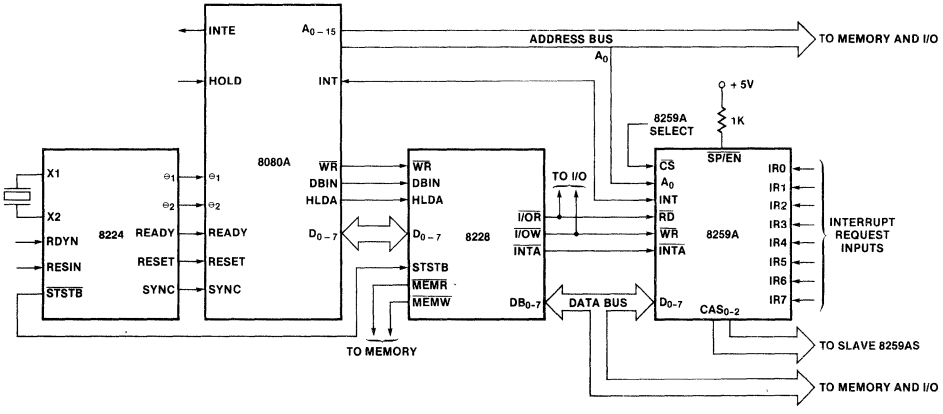


Figure 2. MCS-80 8259A Basic Configuration Example

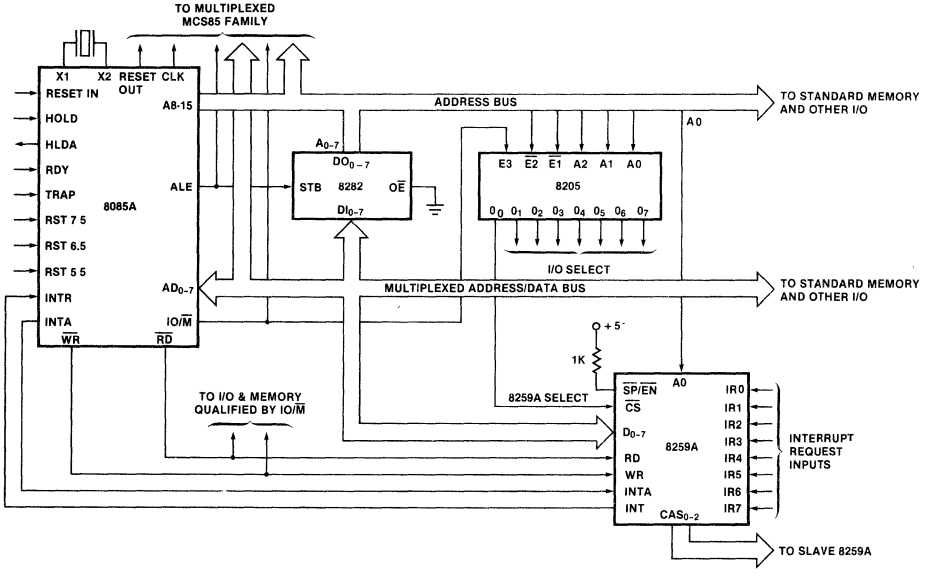


Figure 3. MCS-85 8259A Basic Configuration Example

tional chip, as the 8080A uses the 8228 System Controller Bus Driver. Another hardware difference is the 8085A has five hardware interrupt pins: INTR, RST 7.5, RST 6.5, RST 5.5, and TRAP. The INTR (Interrupt Request) pin is the equivalent to the 8080A's INT pin. The RST (Restart) pins and TRAP pin are all restart interrupts which vector program execution to an individual dedicated address when asserted. The important factor associating these interrupts is their relative priority, as shown below:

TRAP	Highest Priority
RST 7.5	
RST 6.5	
RST 5.5	
INTR	Lowest Priority

The INTR pin has lowest priority among the other 8085A hardware interrupts. Thus, precautions to prevent interrupting 8259A service routines may be necessary. This, of course, depends on how the 8085A interrupts are being used in a particular application. Such precautions can be implemented, however, by masking the RST pins using the SIM instruction. The TRAP pin on the other hand is non-maskable; all interrupt pins but TRAP can be controlled by the EI (Enable Interrupt) and DI (Disable Interrupt) instructions.

For a complete description of the 8085A interrupt structure, refer to the MCS-85 User's Manual.

1.3 MCS-86/88 —8259A OVERVIEW

Operation of an MCS-86/88—8259A configuration has basic similarities of the MCS-80/85—8259A configurations. That is, a device can cause an interrupt by pulling one of the 8259A's interrupt request pins (IR0–IR7) high. If the 8259A honors the request, its INT pin will go high, driving the 8086/8088's INTR pin high. Like the 8080A and 8085A, the INTR pin of the 8086/8088 is asynchronous, thus it can receive an interrupt any time. The 8086/8088 can also accept or disregard requests on INTR under software control using the STI (Set Interrupt) or CLI (Clear Interrupt) instructions. These instructions set or clear the interrupt-enabled flag IF. Upon 8086/8088 reset the IF flag is cleared, disabling external interrupts on INTR. Beside the INTR pin, the 8086/8088 provides an NMI (Non-Maskable Interrupt) pin. The NMI functions similar to the 8085A's TRAP; it can't be disabled or masked. NMI has higher priority than INTR.

Figure 4 shows an MCS-86 MAX Mode system interfacing with an 8259A on the local bus. This MCS-86—8259A configuration is also representative of an MCS-88—8259A configuration except for the data bus which is 16 bits for 8086 and 8 bits for 8088. In the MCS-86 system the 8259A must be on the lower 8 bits of the data bus. Note that the 8259A could also be interfaced on the system bus.

Although there are some basic similarities, the actual processing of interrupts with an 8086/8088 is different than an 8080A or 8085A. When an interrupt request is present and interrupts are enabled, the 8086/8088 enters its interrupt acknowledge machine cycle. The interrupt acknowledge machine cycle pushes the flag registers onto the stack (as in a PUSHF instruction). It then clears the IF flag which disables interrupts. The contents of

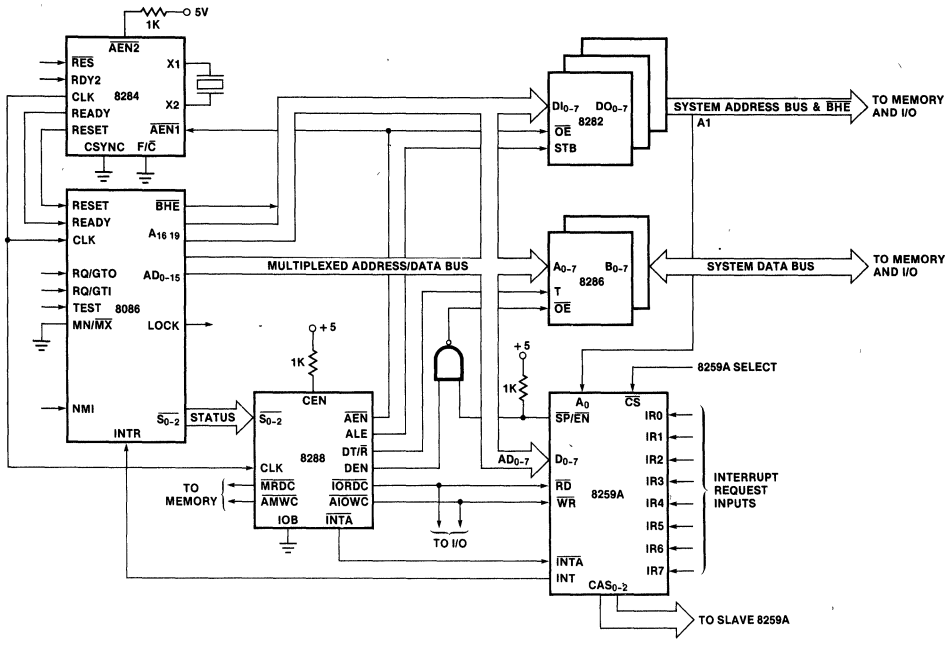


Figure 4. MSC-86 8259A Basic Configuration Example (8086 in Max. Mode)

both the code segment and the instruction pointer are then also pushed onto the stack. Thus, the stack retains the pre-interrupt flag status and pre-interrupt program location which are used to return from the service routine. The 8086/8088 then issues the first of two $\overline{INTA}$ pulses which signal the 8259A that the 8086/8088 has honored its interrupt request. If the 8086/8088 is used in its "MIN Mode" the $\overline{INTA}$ signal is available from the 8086/8088 on its $\overline{INTA}$ pin. If the 8086/8088 is used in the "MAX Mode" the $\overline{INTA}$ signal is available via the 8288 Bus Controller $\overline{INTA}$ pin. Additionally, in the "MAX Mode" the 8086/8088 LOCK pin goes low during the interrupt acknowledge sequence. The LOCK signal can be used to indicate to other system bus masters not to gain control of the system bus during the interrupt acknowledge sequence. A "HOLD" request won't be honored while LOCK is low.

The 8259A is now ready to vector program execution to the corresponding service routine. This is done during the sequence of the two $\overline{INTA}$ pulses issued by the 8086/8088. Unlike operation with the 8080A or 8085A, the 8259A doesn't place a CALL instruction and the starting address of the service routine on the data bus. Instead, the first $\overline{INTA}$ pulse is used only to signal the 8259A of the honored request. The second $\overline{INTA}$ pulse causes the 8259A to place a single interrupt-vector byte onto the data bus. Not used as a direct address, this interrupt-vector byte pertains to one of 256 interrupt "types" supported by the 8086/8088 memory. Program execution is vectored to the corresponding service routine by the contents of a specified interrupt type.

All 256 interrupt types are located in absolute memory locations 0 through 3FFH which make up the 8086/8088's interrupt-vector table. Each type in the interrupt-vector table requires 4 bytes of memory and stores a code segment address and an instruction pointer address. Figure 5 shows a block diagram of the interrupt-vector table. Locations 0 through 3FFH should be reserved for the interrupt-vector table alone. Furthermore, memory locations 00 through 7FH (types 0-31) are reserved for use by Intel Corporation for Intel hardware and software products. To maintain compatibility with present and future Intel products, these locations should not be used.

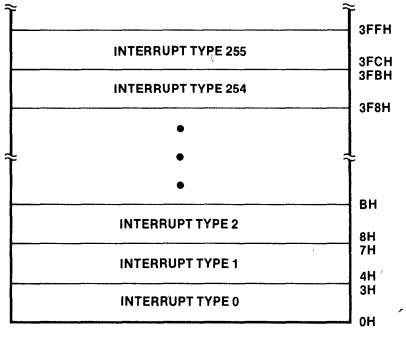


Figure 5. 8086/8088 Interrupt Vector Table

When the 8086/8088 receives an interrupt-vector byte from the 8259A, it multiplies its value by four to acquire the address of the interrupt type. For example, if the interrupt-vector byte specifies type 128 (80H), the vectored address in 8086/8088 memory is 4 x 80H, which equals 200H. Program execution is then vectored to the service routine whose address is specified by the code segment and instruction pointer values within type 128 located at 200H. To show how this is done, let's assume interrupt type 128 is to vector data to 8086/8088 memory location 2FF5FH. Figure 6 shows two possible ways to set values of the code segment and instruction pointer for vectoring to location 2FF5FH. Address generation by the code segment and instruction pointer is accomplished by an offset (they overlap). Of the total 20-bit address capability, the code segment can designate the upper 16 bits, the instruction pointer can designate the lower 16 bits.

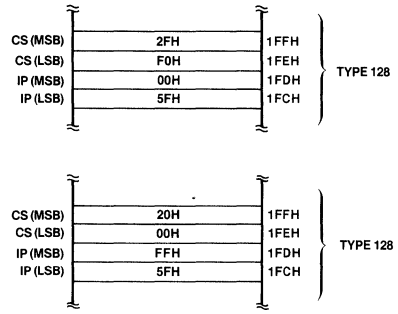


Figure 6. Two Examples of 8086/8088 Interrupt Type 128 Vectoring to Location 2FF5FH

When entering an interrupt service routine, those registers that are mutually used between the main program and service routine should be saved. The best way to do this is to "PUSH" each register used onto the stack immediately. The service routine can then "POP" each register off the stack in the same order when it is completed.

Once the service routine is completed the main program may be re-entered by using a IRET (Interrupt Return) instruction. The IRET instruction will pop the pre-interrupt instruction pointer, code segment and flags off the stack. Thus the main program will resume where it was interrupted with the same flag status regardless of changes in the service routine. Note especially that this includes the state of the IF flag, thus interrupts are re-enabled automatically when returning from the service routine.

Beside external interrupt generation from the INTR pin, the 8086/8088 is also able to invoke interrupts by software. Three interrupt instructions are provided: INT, INT (Type 3), and INTO. INT is a two byte instruction, the second byte selects the interrupt type. INT (Type 3) is a one byte instruction which selects interrupt Type 3. INTO is a conditional one byte interrupt instruction which selects interrupt Type 4 if the OF flag (trap on overflow) is set. All the software interrupts vector program execution as the hardware interrupts do.

For further information on 8086/8088 interrupt operation and internal interrupt structure refer to the MCS-86 User's Manual and the 8086 System Design application note.

2. 8259A FUNCTIONAL BLOCK DIAGRAM

A block diagram of the 8259A is shown in Figure 7. As can be seen from this figure, the 8259A consists of eight major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the cascade buffer/comparator, the data bus buffer, and logic blocks for control and read/write. We'll first go over the blocks directly related to interrupt handling, the IRR, ISR, IMR, PR, and the control logic. The remaining functional blocks are then discussed.

2.1 INTERRUPT REGISTERS AND CONTROL LOGIC

Basically, interrupt requests are handled by three "cascaded" registers: the Interrupt Request Register (IRR) is use to store all the interrupt levels requesting service; the In-Service Register (ISR) stores all the levels which are being serviced; and the Interrupt Mask Register (IMR) stores the bits of the interrupt lines to be masked. The Priority Resolver (PR) looks at the IRR, ISR and IMR, and determines whether an INT should be issued by the the control logic to the processor.

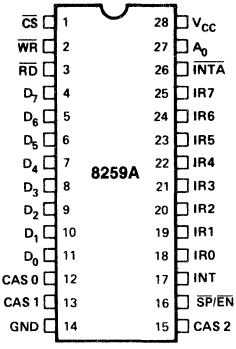
Figure 8 shows conceptually how the Interrupt Request (IR) input handles an interrupt request and how the various interrupt registers interact. The figure represents one of eight "daisy-chained" priority cells, one for each IR input.

The best way to explain the operation of the priority cell is to go through the sequence of internal events that happen when an interrupt request occurs. However, first, notice that the input circuitry of the priority cell allows for both level sensitive and edge sensitive IR inputs. Deciding which method to use is dependent on the particular application and will be discussed in more detail later.

When the IR input is in an inactive state (LOW), the edge sense latch is set. If edge sensitive triggering is selected, the "Q" output of the edge sense latch will arm the input gate to the request latch. This input gate will be disarmed after the IR input goes active (HIGH) and the interrupt request has been acknowledged. This disables the input from generating any further interrupts until it has returned low to re-arm the edge sense latch. If level sensitive triggering is selected, the "Q" output of the edge sense latch is rendered useless. This means the level of the IR input is in complete control of interrupt generation; the input won't be disarmed once acknowledged.

When an interrupt occurs on the IR input, it propagates through the request latch and to the PR (assuming the input isn't masked). The PR looks at the incoming requests and the currently in-service interrupts to ascertain whether an interrupt should be issued to the processor. Let's assume that the request is the only one incoming and no requests are presently in service. The PR then causes the control logic to pull the INT line to the processor high.

PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A_0	COMMAND SELECT ADDRESS
$\overline{CS}$	CHIP SELECT
CAS1-CAS0	CASCADE LINES
$\overline{SP}/\overline{EN}$	SLAVE PROGRAM/ENABLE BUFFER
INT	INTERRUPT OUTPUT
$\overline{INTA}$	INTERRUPT ACKNOWLEDGE INPUT
IR0-IR7	INTERRUPT REQUEST INPUTS

BLOCK DIAGRAM

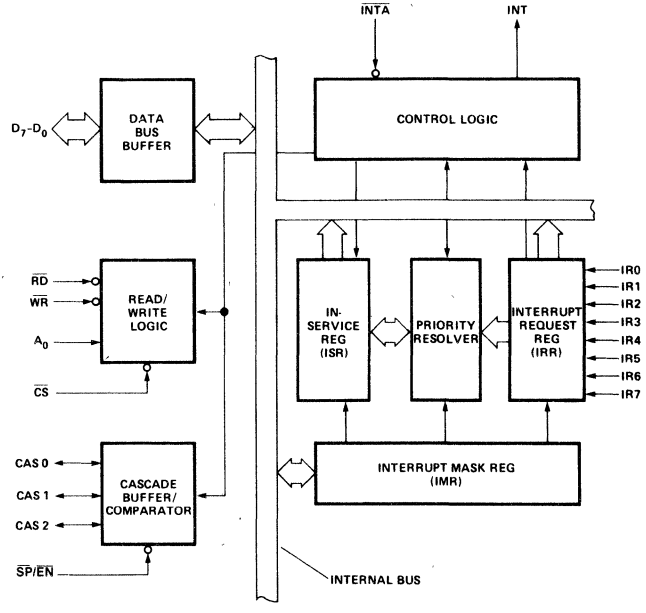


Figure 7. 8259A Block Diagram and Pin Configuration

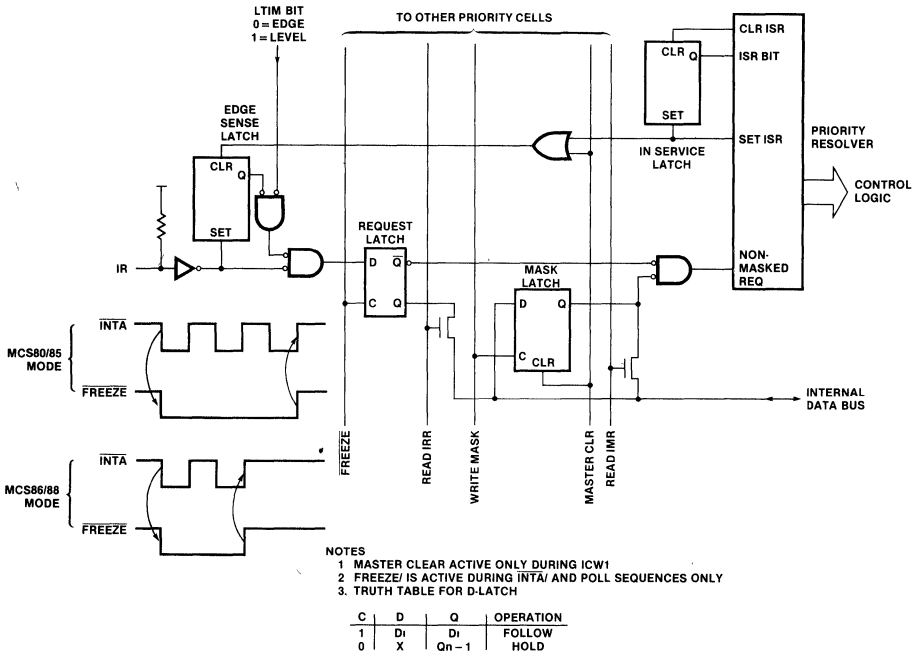


Figure 8. Priority Cell

When the processor honors the INT pulse, it sends a sequence of INTA pulses to the 8259A (three for 8080A/8085A, two for 8086/8088). During this sequence the state of the request latch is frozen (note the INTA-freeze request timing diagram). Priority is again resolved by the PR to determine the appropriate interrupt vectoring which is conveyed to the processor via the data bus.

Immediately after the interrupt acknowledge sequence, the PR sets the corresponding bit in the ISR which simultaneously clears the edge sense latch. if edge sensitive triggering is used, clearing the edge sense latch also disarms the request latch. This inhibits the possibility of a still active IR input from propagating through the priority cell. The IR input must return to an inactive state, setting the edge sense latch, before another interrupt request can be recognized. If level sensitive triggering is used, however, clearing the edge sense latch has no affect on the request latch. The state of the request latch is entirely dependent upon the IR input level. Another interrupt will be generated immediately if the IR level is left active after its ISR bit has been reset. An ISR bit gets reset with an End-of-Interrupt (EOI) command issued in the service routine. End-of-interrupts will be covered in more detail later.

2.2 OTHER FUNCTIONAL BLOCKS

Data Bus Buffer

This three-state, bidirectional 8-bit buffer is used to interface the 8259A to the processor system data bus (via DB0-DB7). Control words, status information, and interrupt-vector data are transferred through the data bus buffer.

Read/Write Control Logic

The function of this block is to control the programming of the 8259A by accepting OUTput commands from the processor. It also controls the releasing of status onto the data bus by accepting INput commands from the processor. The initialization and operation command word registers which store the various control formats are located in this block. The $\overline{RD}$, $\overline{WR}$, $\overline{A0}$, and $\overline{CS}$ pins are used to control access to this block by the processor.

Cascade Buffer/Comparator

As mentioned earlier, multiple 8259A's can be combined to expand the number of interrupt levels. A master-slave relationship of cascaded 8259A's is used for the expansion. The $\overline{SP}/\overline{EN}$ and the CAS0-2 pins are used for operation of this block. The cascading of 8259A's is covered in depth in the "Operation of the 8259A" section of this application note.

2.3 PIN FUNCTIONS

Name	Pin #	I/O	Function
V_{CC}	28	I	+5V supply
GND	14	I	Ground

Name	Pin #	I/O	Function
$\overline{CS}$	1	I	*Chip Select:* A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. $\overline{INTA}$ functions are independent of $\overline{CS}$.
$\overline{WR}$	2	I	*Write:* A low on this pin when $\overline{CS}$ is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	*Read:* A low on this pin when $\overline{CS}$ is low enables the 8259A to release status onto the data bus for the CPU.
D7-D0	4-11	I/O	*Bidirectional Data Bus:* Control, status and interrupt-vector information is transferred via this bus.
CAS0-CAS2	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 ($\overline{EN}$). When not in the buffered mode it is used as an input to designate a master ($\overline{SP} = 1$) or slave ($\overline{SP} = 0$).
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.
IR0-IR7	18-25	I	*Interrupt Requests:* Asynchronous inputs. 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).
$\overline{INTA}$	26	I	*Interrupt Acknowledge:* This pin is used to enable 8259A interrupt-vector data onto the data bus. This is done by a sequence of interrupt acknowledge pulses issued by the CPU.
A0	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 between 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 8086/8088).

3. OPERATION OF THE 8259A

Interrupt operation of the 8259A falls under five main categories: vectoring, priorities, triggering, status, and cascading. Each of these categories use various modes and commands. This section will explain the operation of these modes and commands. For clarity of explanation, however, the actual programming of the 8259A isn't covered in this section but in "Programming the 8259A". Appendix A is provided as a cross reference between these two sections.

3.1 INTERRUPT VECTORING

Each IR input of the 8259A has an individual interrupt-vector address in memory associated with it. Designation of each address depends upon the initial programming of the 8259A. As stated earlier, the interrupt sequence and addressing of an MCS-80 and MCS-85 system differs from that of an MCS-86 and MCS-88 system. Thus, the 8259A must be initially programmed in either a MCS-80/85 or MCS-86/88 mode of operation to insure the correct interrupt vectoring.

MCS-80/85 Mode

When programmed in the MCS-80/85 mode, the 8259A should only be used within an 8080A or an 8085A system. In this mode the 8080A/8085A will handle interrupts in the format described in the "MCS-80—8259A or MCS-85—8259A Overviews."

Upon interrupt request in the MCS-80/85 mode, the 8259A will output to the data bus the opcode for a CALL instruction and the address of the desired routine. This is in response to a sequence of three $\overline{INTA}$ pulses issued by the 8080A/8085A after the 8259A has raised INT high.

The first INTA pulse to the 8259A enables the CALL opcode "CD_H" onto the data bus. It also resolves IR priorities and effects operation in the cascade mode, which will be covered later. Contents of the first interrupt-vector byte are shown in Figure 9A.

During the second and third $\overline{INTA}$ pulses, the 8259A conveys a 16-bit interrupt-vector address to the 8080A/8085A. The interrupt-vector addresses for all eight levels are selected when initially programming the 8259A. However, only one address is needed for programming. Interrupt-vector addresses of IR0-IR7 are automatically set at equally spaced intervals based on the one programmed address. Address intervals are user definable to 4 or 8 bytes apart. If the service routine for a device is short it may be possible to fit the entire routine within an 8-byte interval. Usually, though, the service routines require more than 8 bytes. So, a 4-byte interval is used to store a Jump (JMP) instruction which directs the 8080A/8085A to the appropriate routine. The 8-byte interval maintains compatibility with current 8080A/8085A Restart (RST) instruction software, while the 4-byte interval is best for a compact jump table. If the 4-byte interval is selected, then the 8259A will automatically insert bits A0-A4. This leaves A5-A15 to be programmed by the user. If the 8-byte interval is selected, the 8259A will automatically insert bits A0-A5. This leaves only A6-A15 to be programmed by the user.

The LSB of the interrupt-vector address is placed on the data bus during the second $\overline{INTA}$ pulse. Figure 9B shows the contents of the second interrupt-vector byte for both 4 and 8-byte intervals.

The MSB of the interrupt-vector address is placed on the data bus during the third $\overline{INTA}$ pulse. Contents of the third interrupt-vector byte is shown in Figure 9C.

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

A. FIRST INTERRUPT VECTOR BYTE, MCS80/85 MODE

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

B. SECOND INTERRUPT VECTOR BYTE, MCS80/85 MODE

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

C. THIRD INTERRUPT VECTOR BYTE, MCS80/85 MODE

Figure 9. 9A-C. Interrupt-Vector Bytes for 8259A, MCS 80/85 Mode

MCS-86/88 Mode

When programmed in the MCS-86/88 mode, the 8259A should only be used within an MCS-86 or MCS-88 system. In this mode, the 8086/8088 will handle interrupts in the format described earlier in the "8259A—8086/8088 Overview".

Upon interrupt in the MCS-86/88 mode, the 8259A will output a single interrupt-vector byte to the data bus. This is in response to only two $\overline{\text{INTA}}$ pulses issued by the 8086/8088 after the 8259A has raised INT high.

The first $\overline{\text{INTA}}$ pulse is used only for set-up purposes internal to the 8259A. As in the MCS-80/85 mode, this set-up includes priority resolution and cascade mode operations which will be covered later. Unlike the MCS-80/85 mode, no CALL opcode is placed on the data bus.

The second $\overline{\text{INTA}}$ pulse is used to enable the single interrupt-vector byte onto the data bus. The 8086/8088 uses this interrupt-vector byte to select one of 256 interrupt "types" in 8086/8088 memory. Interrupt type selection for all eight IR levels is made when initially programming the 8259A. However, reference to only one interrupt type is needed for programming. The upper 5 bits of the interrupt vector byte are user definable. The lower 3 bits are automatically inserted by the 8259A depending upon the IR level.

Contents of the interrupt-vector byte for 8086/8088 type selection is put on the data bus during the second $\overline{\text{INTA}}$ pulse and is shown in Figure 10.

IR	D7	D6	D5	D4	D3	D2	D1	D0
7	T7	T6	T5	T4	T3	1	1	1
6	T7	T6	T5	T4	T3	1	1	0
5	T7	T6	T5	T4	T3	1	0	1
4	T7	T6	T5	T4	T3	1	0	0
3	T7	T6	T5	T4	T3	0	1	1
2	T7	T6	T5	T4	T3	0	1	0
1	T7	T6	T5	T4	T3	0	0	1
0	T7	T6	T5	T4	T3	0	0	0

Figure 10. Interrupt Vector Byte, MCS 86/88 Mode

3.2 INTERRUPT PRIORITIES

A variety of modes and commands are available for controlling interrupt priorities of the 8259A. All of them are programmable, that is, they may be changed dynamically under software control. With these modes and commands, many possibilities are conceivable, giving the user enough versatility for almost any interrupt controlled application.

Fully Nested Mode

The fully nested mode of operation is a general purpose priority mode. This mode supports a multilevel-interrupt structure in which priority order of all eight IR inputs are arranged from highest to lowest.

Unless otherwise programmed, the fully nested mode is entered by default upon initialization. At this time, IR0 is assigned the highest priority through IR7 the lowest. The fully nested mode, however, is not confined to this IR structure alone. Once past initialization, other IR inputs can be assigned highest priority also, keeping the multilevel-interrupt structure of the fully nested mode. Figure 11A-C shows some variations of the priority structures in the fully nested mode.

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	7	6	5	4	3	2	1	0

A

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	4	3	2	1	0	7	6	5

B

IR LEVELS	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
PRIORITY	1	0	7	6	5	4	3	2

C

Figure 11. A-C. Some Variations of Priority Structure in the Fully Nested Mode

Further explanation of the fully nested mode, in this section, is linked with information of general 8259A interrupt operations. This is done to ease explanation to the user in both areas.

In general, when an interrupt is acknowledged, the highest priority request is determined from the IRR (Interrupt Request Register). The interrupt vector is then placed on the data bus. In addition, the corresponding bit in the ISR (In-Service Register) is set to designate the routine in service. This ISR bit remains set until an EOI (End-Of-Interrupt) command is issued to the 8259A. EOI's will be explained in greater detail shortly.

In the fully nested mode, while an ISR bit is set, all further requests of the same or lower priority are inhibited from generating an interrupt to the microprocessor. A higher priority request, though, can generate an interrupt, thus vectoring program execution to its service routine. Interrupts are only acknowledged, however, if the microprocessor has previously executed an "Enable Interrupts" instruction. This is because the interrupt request pin on the microprocessor gets disabled automatically after acknowledgement of any interrupt. The assembly language instructions used to enable interrupts are "EI" for 8080A/8085A and "STI" for 8086/8088. Interrupts can be disabled by using the instruction "DI" for 8080A/ 8085A and "CLI" for 8086/8088. When a routine is completed a "return" instruction is executed, "RET" for 8080A/8085A and "IRET" for 8086/8088.

Figure 12 illustrates the correct usage of interrupt related instructions and the interaction of interrupt levels in the fully nested mode.

Assuming the IR priority assignment for the example in Figure 12 is IR0 the highest through IR7 the lowest, the sequence is as follows. During the main program, IR3 makes a request. Since interrupts are enabled, the microprocessor is vectored to the IR3 service routine. During the IR3 routine, IR1 asserts a request. Since IR1 has higher priority than IR3, an interrupt is generated. However, it is not acknowledged because the microprocessor disabled interrupts in response to the IR3 interrupt. The IR1 interrupt is not acknowledged until the "Enable Interrupts" instruction is executed. Thus the IR3 routine has a "protected" section of code over which no interrupts (except non-maskable) are allowed. The IR1 routine has no such "protected" section since an "Enable Interrupts" instruction is the first one in its service routine. Note that in this example the IR1 request must stay high until it is acknowledged. This is covered in more depth in the "Interrupt Triggering" section.

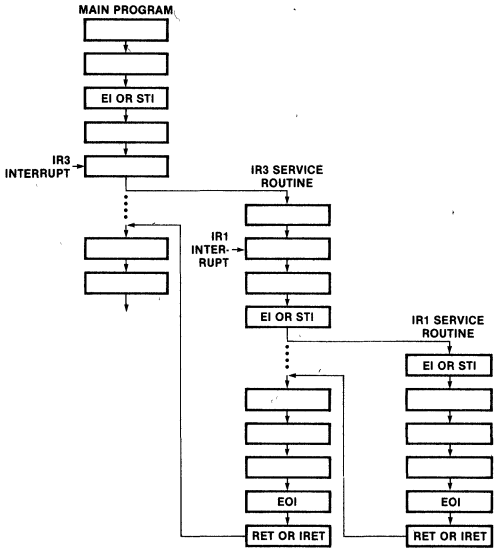


Figure 12. Fully Nested Mode Example (MCS 80/85 or MCS 86/88)

What is happening to the ISR register? While in the main program, no ISR bits are set since there aren't any interrupts in service. When the IR3 interrupt is acknowledged, the ISR3 bit is set. When the IR1 interrupt is acknowledged, both the ISR1 and the ISR3 bits are set, indicating that neither routine is complete. At this time, only IR0 could generate an interrupt since it is the only input with a higher priority than those previously in service. To terminate the IR1 routine, the routine must inform the 8259A that it is complete by resetting its ISR bit. It does this by executing an EOI command. A "return" instruction then transfers execution back to the IR3 routine. This allows IR0-IR2 to interrupt the IR3 routine again, since ISR3 is the highest ISR bit set. No further interrupts occur in the example so the EOI command resets ISR3 and the "return" instruction causes the main program to resume at its pre-interrupt location, ending the example.

A single 8259A is essentially always in the fully nested mode unless certain programming conditions disturb it. The following programming conditions can cause the 8259A to go out of the high to low priority structure of the fully nested mode.

- The automatic EOI mode
- The special mask mode
- A slave with a master not in the special fully nested mode

These modes will be covered in more detail later, however, they are mentioned now so the user can be aware of them. As long as these program conditions aren't inacted, the fully nested mode remains undisturbed.

End of Interrupt

Upon completion of an interrupt service routine the 8259A needs to be notified so its ISR can be updated. This is done to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available for the user. These are: the non-specific EOI command, the specific EOI command, and the automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

Non-Specific EOI Command

A non-specific EOI command sent from the microprocessor lets the 8259A know when a service routine has been completed, without specification of its exact interrupt level. The 8259A automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the non-specific EOI the 8259A must be in a mode of operation in which it can predetermine in-service routine levels. For this reason the non-specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level. When the 8259A receives a non-specific EOI command, it simply resets the highest priority ISR bit, thus confirming to the 8259A that the highest priority routine of the routines in service is finished.

The main advantage of using the non-specific EOI command is that IR level specification isn't necessary as in the "Specific EOI Command", covered shortly. However, special consideration should be taken when deciding to use the non-specific EOI. Here are two program conditions in which it is best not used:

- Using the set priority command within an interrupt service routine.
- Using a special mask mode.

These conditions are covered in more detail in their own sections, but are listed here for the users reference.

Specific EOI Command

A specific EOI command sent from the microprocessor lets the 8259A know when a service routine of a particular interrupt level is completed. Unlike a non-specific EOI command, which automatically resets the highest priority ISR bit, a specific EOI command specifies an exact ISR bit to be reset. One of the eight IR levels of the 8259A can be specified in the command.

The reason the specific EOI command is needed, is to reset the ISR bit of a completed service routine whenever the 8259A isn't able to automatically determine it. An example of this type of situation might be if the priorities of the interrupt levels were changed during an interrupt routine ("Specific Rotation"). In this case, if any other routines were in service at the same time, a non-specific EOI might reset the wrong ISR bit. Thus the specific EOI command is the best bet in this case, or for that matter, any time in which confusion of interrupt priorities may exist. The specific EOI command can be used in all conditions of 8259A operation, including those that prohibit non-specific EOI command usage.

Automatic EOI Mode

When programmed in the automatic EOI mode, the microprocessor no longer needs to issue a command to notify the 8259A it has completed an interrupt routine. The 8259A accomplishes this by performing a non-specific EOI automatically at the trailing edge of the last $\overline{\text{INTA}}$ pulse (third pulse in MCS-80/85, second in MCS-86).

The obvious advantage of the automatic EOI mode over the other EOI command is no command has to be issued. In general, this simplifies programming and lowers code requirements within interrupt routines.

However, special consideration should be taken when deciding to use the automatic EOI mode because it disturbs the fully nested mode. In the automatic EOI mode the ISR bit of a routine in service is reset right after it's acknowledged, thus leaving no designation in the ISR that a sevice routine is being executed. If any interrupt request occurs during this time (and interrupts are enabled) it will get serviced regardless of its priority, low or high. The problem of "over nesting" may also happen in this situation. "Over nesting" is when an IR input keeps interrupting its own routine, resulting in unnecessary stack pushes which could fill the stack in a worst case condition. This is not usually a desired form of operation!

So what good is the automatic EOI mode with problems like those just covered? Well, again, like the other EOIs, selection is dependent upon the application. If interrupts are controlled at a predetermined rate, so as not to cause the problems mentioned above, the automatic EOI mode works perfect just the way it is. However, if interrupts happen sporadically at an indeterminate rate, the automatic EOI mode should only be used under the following guideline:

- When using the automatic EOI mode with an indeterminate interrupt rate, the microprocessor should keep its interrupt request input disabled during execution of service routines.

By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the fully nested structure in regards to the IRR; however, a routine in-service can't be interrupted.

Automatic Rotation — Equal Priority

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority, such as communications channels. The concept is that once a peripheral is serviced, all other equal priority peripherals should be given a chance to be serviced before the original peripheral is serviced again. This is accomplished by automatically assigning a peripheral the lowest priority after being serviced Thus, in worst case, the device would have to wait until all other devices are serviced before being serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the non-specific EOI, "rotate on non-specific EOI command". The other is used with the automatic EOI mode, "rotate in automatic EOI mode".

Rotate on Non-Specific EOI Command

When the rotate on non-specific EOI command is issued, the highest ISR bit is reset as in a normal non-specific EOI command. After it's reset though, the corresponding IR level is assigned lowest priority. Other IR priorities rotate to conform to the fully nested mode based on the newly assigned low priority

Figures 13A and B show how the rotate on non-specific EOI command effects the interrupt priorities. Let's assume the IR priorities were assigned with IR0 the highest and IR7 the lowest, as in 13A. IR6 and IR4 are already in service but neither is completed. Being the higher priority routine, IR4 is necessarily the routine being executed. During the IR4 routine a rotate on non-specific EOI command is executed. When this happens, bit 4 in the ISR is reset. IR4 then becomes the lowest priority and IR5 becomes the highest as in 13B.

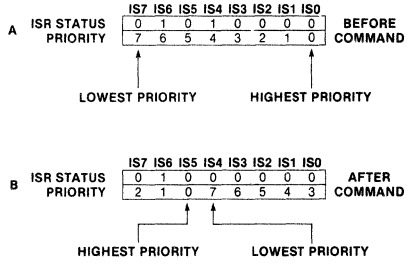


Figure 13. A-B. Rotate on Non-specific EOI Command Example

Rotate in Automatic EOI Mode

The rotate in automatic EOI mode works much like the rotate on non-specific EOI command. The main difference is that priority rotation is done automatically after

the last $\overline{INTA}$ pulse of an interrupt request. To enter or exit this mode a rotate-in-automatic-EOI set command and rotate-in-automatic-EOI clear command is provided. After that, no commands are needed as with the normal automatic EOI mode. However, it must be remembered, when using any form of the automatic EOI mode, special consideration should be taken. Thus, the guideline for the automatic EOI mode also stands for the rotate in automatic EOI mode.

Specific Rotation — Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to automatic rotation which automatically sets priorities, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive lowest or highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user, the "set priority command" and the "rotate on specific EOI command."

Set Priority Command

The set priority command allows the programmer to assign an IR level the lowest priority. All other interrupt levels will conform to the fully nested mode based on the newly assigned low priority.

An example of how the set priority command works is shown in Figures 14A and 14B. These figures show the status of the ISR and the relative priorities of the interrupt levels before and after the set priority command. Two interrupt routines are shown to be in service in Figure 14A. Since IR2 is the highest priority, it is necessarily the routine being executed. During the IR2 routine, priorities are altered so that IR5 is the highest. This is done simply by issuing the set priority command to the 8259A. In this case, the command specifies IR4 as being the lowest priority. The result of this set priority command is shown in Figure 14B. Even though IR7 now has higher priority than IR2, it won't be acknowledged until the IR2 routine is finished (via EOI). This is because priorities are only resolved upon an interrupt request or an interrupt acknowledge sequence. If a higher priority request occurs during the IR2 routine, then priorities are resolved and the highest will be acknowledged.

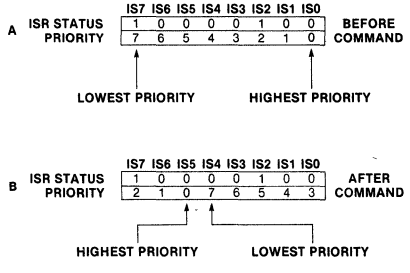


Figure 14. A-B. Set Priority Command Example

When completing a service routine in which the set priority command is used, the correct EOI must be issued. The non-specific EOI command shouldn't be used in the same routine as a set priority command. This is because the non-specific EOI command resets the highest ISR bit, which, when using the set priority command, is not always the most recent routine in service. The automatic EOI mode, on the other hand, can be used with the set priority command. This is because it automatically performs a non-specific EOI before the set priority command can be issued. The specific EOI command is the best bet in most cases when using the set priority command within a routine. By resetting the specific ISR bit of a routine being completed, confusion is eliminated.

Rotate on Specific EOI Command

The rotate on specific EOI command is literally a combination of the set priority command and the specific EOI command. Like the set priority command, a specified IR level is assigned lowest priority. Like the specific EOI command, a specified level will be reset in the ISR. Thus the rotate on specific EOI command accomplishes both tasks in only one command.

If it is not necessary to change IR priorities prior to the end of an interrupt routine, then this command is advantageous. For an EOI command must be executed anyway (unless in the automatic EOI mode), so why not do both at the same time?

Interrupt Masking

Disabling or enabling interrupts can be done by other means than just controlling the microprocessor's interrupt request pin. The 8259A has an IMR (Interrupt Mask Register) which enhances interrupt control capabilities. Rather than all interrupts being disabled or enabled at the same time, the IMR allows individual IR masking. The IMR is an 8-bit register, bits 0-7 directly correspond to IR0-IR7. Any IR input can be masked by writing to the IMR and setting the appropriate bit. Likewise, any IR input can be enabled by clearing the correct IMR bit.

There are various uses for masking off individual IR inputs. One example is when a portion of a main routine wishes only to be interrupted by specific interrupts. Another might be disabling higher priority interrupts for a portion of a lower priority service routine. The possibilities are many.

When an interrupt occurs while its IMR bit is set, it isn't necessarily forgotten. For, as stated earlier, the IMR acts only on the output of the IRR. Even with an IR input masked it is still possible to set the IRR. Thus, when resetting an IMR, if its IRR bit is set it will then generate an interrupt. This is providing, of course, that other priority factors are taken into consideration and the IR request remains active. If the IR request is removed before the IMR is reset, no interrupt will be acknowledged.

Special Mask Mode

In various cases, it may be desirable to enable interrupts of a lower priority than the routine in service. Or, in other words, allow lower priority devices to generate interrupts. However, in the fully nested mode, all IR levels of

priority below the routine in service are inhibited. So what can be done to enable them?

Well, one method could be using an EOI command before the actual completion of a routine in service. But beware, doing this may cause an "over nesting" problem, similar to in the automatic EOI mode. In addition, resetting an ISR bit is irreversible by software control, so lower priority IR levels could only be later disabled by setting the IMR.

A much better solution is the special mask mode. Working in conjunction with the IMR, the special mask mode enables interrupts from all levels except the level in service. This is done by masking the level that is in service and then issuing the special mask mode command. Once the special mask mode is set, it remains in effect until reset.

Figure 15 shows how to enable lower priority interrupts by using the Special Mask Mode (SMM). Assume that IR0 has highest priority when the main program is interrupted by IR4. In the IR4 service routine an enable interrupt instruction is executed. This only allows higher priority interrupt requests to interrupt IR4 in the normal fully nested mode. Further in the IR4 routine, bit 4 of the IMR is masked and the special mask mode is entered. Priority operation is no longer in the fully nested mode. All interrupt levels are enabled except for IR4. To leave the special mask mode, the sequence is executed in reverse.

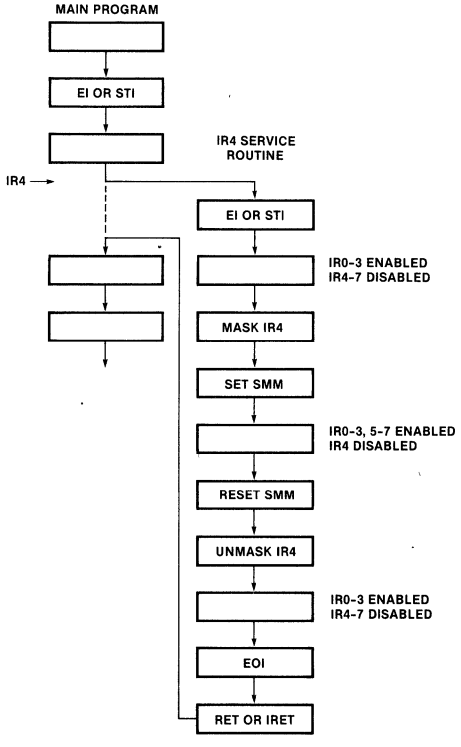


Figure 15. Special Mask Mode Example (MCS 80/85 or MCS 86/88)

Precautions must be taken when exiting an interrupt service routine which has used the special mask mode. A non-specific EOI command can't be used when in the special mask mode. This is because a non-specific won't clear an ISR bit of an interrupt which is masked when in the special mask mode. In fact, the bit will appear invisible. If the special mask mode is cleared before an EOI command is issued a non-specific EOI command can be used. This could be the case in the example shown in Figure 15, but, to avoid any confusion it's best to use the specific EOI whenever using the special mask mode.

It must be remembered that the special mask mode applies to all masked levels when set. Take, for instance, IR1 interrupting IR4 in the previous example. If this happened while in the special mask mode, and the IR1 routine masked itself, all interrupts would be enabled except IR1 and IR4 which are masked.

3.3 INTERRUPT TRIGGERING

There are two classical ways of sensing an active interrupt request: a level sensitive input or an edge sensitive input. The 8259A gives the user the capability for either method with the edge triggered mode and the level triggered mode. Selection of one of these interrupt triggering methods is done during the programmed initialization of the 8259A.

Level Triggered Mode

When in the level triggered mode the 8259A will recognize any active (high) level on an IR input as an interrupt request. If the IR input remains active after an EOI command has been issued (resetting its ISR bit), another interrupt will be generated. This is providing of course, the processor INT pin is enabled. Unless repetitious interrupt generation is desired, the IR input must be brought to an inactive state before an EOI command is issued in its service routine. However, it must not go inactive so soon that it disobeys the necessary timing requirements shown in Figure 16. Note that the request on the IR input must remain until after the falling edge of the first $\overline{\text{INTA}}$ pulse. If on any IR input, the request goes inactive before the first $\overline{\text{INTA}}$ pulse, the 8259A will respond as if IR7 was active. In any design in which there's a possibility of this happening, the IR7 default feature can be used as a safeguard. This can be accomplished by using the IR7 routine as a "clean-up routine" which might recheck the 8259A status or merely return program execution to its pre-interrupt location.

Depending upon the particular design and application, the level triggered mode has a number of uses. For one, it provides for repetitious interrupt generation. This is useful in cases when a service routine needs to be continually executed until the interrupt request goes inactive. Another possible advantage of the level triggered mode is it allows for "wire-OR'ed" interrupt requests. That is, a number of interrupt requests using the same IR input. This can't be done in the edge triggered mode, for if a device makes an interrupt request while the IR input is high (from another request), its transition will be "shadowed". Thus the 8259A won't recognize further interrupt requests because its IR input is already high. Note that when a "wire-OR'ed" scheme is used, the ac-

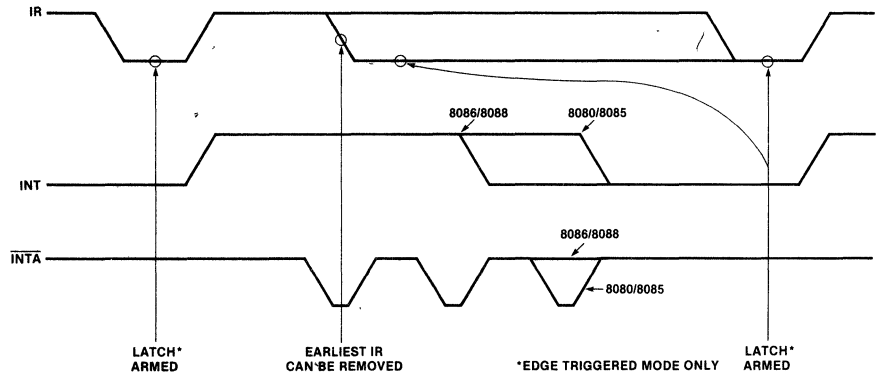


Figure 16. IR Triggering Timing Requirements

tual requesting device has to be determined by the software in the service routine.

Caution should be taken when using the automatic EOI mode and the level triggered mode together. Since in the automatic EOI mode an EOI is automatically performed at the end of the interrupt acknowledge sequence, if the processor enables interrupts while an IR input is still high, an interrupt will occur immediately. To avoid this situation interrupts should be kept disabled until the end of the service routine or until the IR input returns low.

Edge Triggered Mode

When in the edge triggered mode, the 8259A will only recognize interrupts if generated by an inactive (low) to active (high) transition on an IR input. The edge triggered mode incorporates an edge lockout method of operation. This means that after the rising edge of an interrupt request and the acknowledgement of the request, the positive level of the IR input won't generate further interrupts on this level. The user needn't worry about quickly removing the request after acknowledgement in fear of generating further interrupts as might be the case in the level triggered mode. Before another interrupt can be generated the IR input must return to the inactive state.

Referring back to Figure 16, the timing requirements for interrupt triggering is shown. Like the level triggered mode, in the edge triggered mode the request on the IR input must remain active until after the falling edge of the first $\overline{\text{INTA}}$ pulse for that particular interrupt. Unlike the level triggered mode, though, after the interrupt request is acknowledged its IRR latch is disarmed. Only after the IR input goes inactive will the IRR latch again become armed, making it ready to receive another interrupt request (in the level triggered mode, the IRR latch is always armed). Because of the way the edge triggered mode functions, it is best to use a positive level with a negative pulse to trigger the IR requests. With this type of input, the trailing edge of the pulse causes the interrupt and the maintained positive level meets the necessary timing requirements (remaining high until after the interrupt acknowledge occurs). Note that the IR7 default feature mentioned in the "level triggered mode" section also works for the edge triggered mode.

Depending upon the particular design and application, the edge triggered mode has various uses. Because of its edge lockout operation, it is best used in those applications where repetitious interrupt generation isn't desired. It is also very useful in systems where the interrupt request is a pulse (this should be in the form of a negative pulse to the 8259A). Another possible advantage is that it can be used with the automatic EOI mode without the cautions in the level triggered mode. Overall, in most cases, the edge triggered mode simplifies operation for the user, since the duration of the interrupt request at a positive level is not usually a factor.

3.4 INTERRUPT STATUS

By means of software control, the user can interrogate the status of the 8259A. This allows the reading of the internal interrupt registers, which may prove useful for interrupt control during service routines. It also provides for a modified status poll method of device monitoring, by using the poll command. This makes the status of the internal IR inputs available to the user via software control. The poll command offers an alternative to the interrupt vector method, especially for those cases when more than 64 interrupts are needed.

Reading Interrupt Registers

The contents of each 8-bit interrupt register, IRR, ISR, and IMR, can be read to update the user's program on the present status of the 8259A. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations. Before delving into the actual process of reading the registers, let's briefly review their general descriptions:

IRR (Interrupt Request Register)	Specifies all interrupt levels requesting service.
ISR (In-Service Register)	Specifies all interrupt levels which are being serviced.
IMR (Interrupt Mask Register)	Specifies all interrupt levels that are masked.

To read the contents of the IRR or ISR, the user must first issue the appropriate read register command (read IRR or read ISR) to the 8259A. Then by applying a $\overline{RD}$ pulse to the 8259A (an INput instruction), the contents of the desired register can be acquired. There is no need to issue a read register command every time the IRR or ISR is to be read. Once a read register command is received by the 8259A, it "remembers" which register has been selected. Thus, all that is necessary to read the contents of the same register more than once is the $\overline{RD}$ pulse and the correct addressing (A0 = 0, explained in "Programming the 8259A"). Upon initialization, the selection of registers defaults to the IRR. Some caution should be taken when using the read register command in a system that supports several levels of interrupts. If the higher priority routine causes an interrupt between the read register command and the actual input of the register contents, there's no guarantee that the same register will be selected when it returns. Thus it is best in such cases to disable interrupts during the operation.

Reading the contents of the IMR is different than reading the IRR or ISR. A read register command is not necessary when reading the IMR. This is because the IMR can be addressed directly for both reading and writing. Thus all that the 8259A requires for reading the IMR is a $\overline{RD}$ pulse and the correct addressing (A0 = 1, explained in "Programming the 8259A").

Poll Command

As mentioned towards the beginning of this application note, there are two methods of servicing peripherals: status polling and interrupt servicing. For most applications the interrupt service method is best. This is because it requires the least amount of CPU time, thus increasing system throughput. However, for certain applications, the status poll method may be desirable.

For this reason, the 8259A supports polling operations with the poll command. As opposed to the conventional method of polling, the poll command offers improved device servicing and increased throughput. Rather than having the processor poll each peripheral in order to find the actual device requiring service, the processor polls the 8259A. This allows the use of all the previously mentioned priority modes and commands. Additionally, both polled and interrupt methods can be used within the same program.

To use the poll command the processor must first have its interrupt request pin disabled. Once the poll command is issued, the 8259A will treat the next ($\overline{CS}$ qualified) $\overline{RD}$ pulse issued to it (an INput instruction) as an interrupt acknowledge. It will then set the appropriate bit in the ISR, if there was an interrupt request, and enable a special word onto the data bus. This word shows whether an interrupt request has occurred and the highest priority level requesting service. Figure 17 shows the contents of the "poll word" which is read by the processor. Bits W0-W2 convey the binary code of the highest priority level requesting service. Bit I designates whether or not an interrupt request is present. If an interrupt request is present, bit I will equal 1. If there isn't an interrupt request at all, bit I will equal 0 and bits W0-W2 will be set to ones. Service to the requesting device is achieved by software decoding the poll word and branching to the appropriate service routine. Each time the 8259A is to be polled, the poll command must be written before reading the poll word.

The poll command is useful in various situations. For instance, it's a good alternative when memory is very limited, because an interrupt-vector table isn't needed. Another use for the poll command is when more than 64 interrupt levels are needed (64 is the limit when cascading 8259's). The only limit of interrupts using the poll command is the number of 8259's that can be addressed in a particular system. Still another application of the poll command might be when the INT or $\overline{INTA}$ signals are not available. This might be the case in a large system where a processor on one card needs to use an 8259A on a different card. In this instance, the poll command is the only way to monitor the interrupt devices and still take advantage of the 8259A's prioritizing features. For those cases when the 8259A is using the poll command only and not the interrupt method, each 8259A must receive an initialization sequence (interrupt vector). This must be done even though the interrupt vector features of the 8259A are not used. In this case, the interrupt vector specified in the initialization sequence could be a "fake".

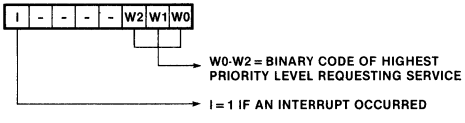


Figure 17. Poll Word

3.5 INTERRUPT CASCADING

As mentioned earlier, more than one 8259A can be used to expand the priority interrupt scheme to up to 64 levels without additional hardware. This method for expanded interrupt capability is called "cascading". The 8259A supports cascading operations with the cascade mode. Additionally, the special fully nested mode and the buffered mode are available for increased flexibility when cascading 8259A's in certain applications.

Cascade Mode

When programmed in the cascade mode, basic operation consists of one 8259A acting as a master to the others which are serving as slaves. Figure 18 shows a system containing a master and two slaves, providing a total of 22 interrupt levels.

A specific hardware set-up is required to establish operation in the cascade mode. With Figure 18 as a reference, note that the master is designated by a high on the $\overline{SP}/\overline{EN}$ pin, while the $\overline{SP}/\overline{EN}$ pins of the slaves are grounded (this can also be done by software, see buffered mode). Additionally, the INT output pin of each slave is connected to an IR input pin of the master. The CAS0-2 pins for all 8259A's are paralleled. These pins act as outputs when the 8259A is a master and as inputs for the slaves. Serving as a private 8259A bus, they control which slave has control of the system bus for interrupt vectoring operation with the processor. All other pins are connected as in normal operation (each 8259A receives an INTA pulse).

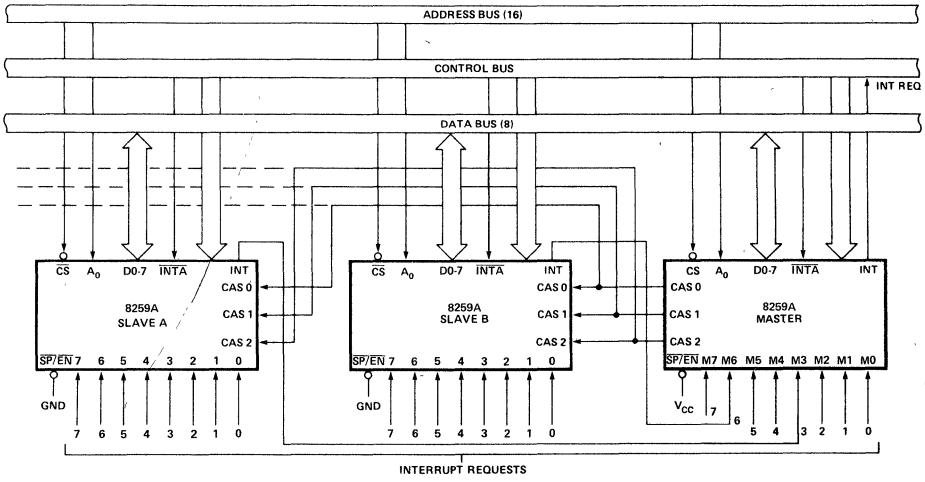


Figure 18. Cascaded 8259A'S 22 Interrupt Levels

Besides hardware set-up requirements, all 8259A's must be software programmed to work in the cascade mode. Programming the cascade mode is done during the initialization of each 8259A. The 8259A that is selected as master must receive specification during its initialization as to which of its IR inputs are connected to a slave's INT pin. Each slave 8259A, on the other hand, must be designated during its initialization with an ID (0 through 7) corresponding to which of the master's IR inputs its INT pin is connected to. This is all necessary so the CAS0-2 pins of the masters will be able to address each individual slave. Note that as in normal operation, each 8259A must also be initialized to give its IR inputs a unique interrupt vector. More detail on the necessary programming of the cascade mode is explained in "Programming the 8259A".

Now, with background information on both hardware and software for the cascade mode, let's go over the sequence of events that occur during a valid interrupt request from a slave. Suppose a slave IR input has received an interrupt request. Assuming this request is higher priority than other requests and in-service levels on the slave, the slave's INT pin is driven high. This signals the master of the request by causing an interrupt request on a designated IR pin of the master. Again, assuming that this request to the master is higher priority than other master requests and in-service levels (possibly from other slaves), the master's INT pin is pulled high, interrupting the processor.

The interrupt acknowledge sequence appears to the processor the same as the non-cascading interrupt acknowledge sequence; however, it's different among the 8259A's. The first $\overline{INTA}$ pulse is used by all the 8259A's for internal set-up purposes and, if in the 8080/8085 mode, the master will place the CALL opcode on the data bus. The first $\overline{INTA}$ pulse also signals the master to place the requesting slave's ID code on the CAS lines. This turns control over to the slave for the rest of the interrupt acknowledge sequence, placing the appropriate pre-programmed interrupt vector on the data bus, completing the interrupt request.

During the interrupt acknowledge sequence, the corresponding ISR bit of both the master and the slave get set. This means two EOI commands must be issued (if not in the automatic EOI mode), one for the master and one for the slave.

Special consideration should be taken when mixed interrupt requests are assigned to a master 8259A; that is, when some of the master's IR inputs are used for slave interrupt requests and some are used for individual interrupt requests. In this type of structure, the master's IR0 must not be used for a slave. This is because when an IR input that isn't initialized as a slave receives an interrupt request, the CAS0-2 lines won't be activated, thus staying in the default condition addressing for IR0 (slave IR0). If a slave is connected to the master's IR0 when a non-slave interrupt occurs on another master IR input, erroneous conditions may result. Thus IR0 should be the last choice when assigning slaves to IR inputs.

Special Fully Nested Mode

Depending on the application, changes in the nested structure of the cascade mode may be desired. This is because the nested structure of a slave 8259A differs from that of the normal fully nested mode. In the cascade mode, if a slave receives a higher priority interrupt request than one which is in service (through the same slave), it won't be recognized by the master. This is because the master's ISR bit is set, ignoring all requests of equal or lower priority. Thus, in this case, the higher priority slave interrupt won't be serviced until after the master's ISR bit is reset by an EOI command. This is most likely after the completion of the lower priority routine.

If the user wishes to have a truly fully nested structure within a slave 8259A, the special fully nested mode should be used. The special fully nested mode is pro-

grammed in the master only. This is done during the master's initialization. In this mode the master will ignore only those interrupt requests of lower priority than the set ISR bit and will respond to all requests of equal or higher priority. Thus if a slave receives a higher priority request than one in service, it will be recognized. To insure proper interrupt operation when using the special fully nested mode, the software must determine if any other slave interrupts are still in service before issuing an EOI command to the master. This is done by resetting the appropriate slave ISR bit with an EOI and then reading its ISR. If the ISR contains all zeros, there aren't any other interrupts from the slave in service and an EOI command can be sent to the master. If the ISR isn't all zeros, an EOI command shouldn't be sent to the master. Clearing the master's ISR bit with an EOI command while there are still slave interrupts in service would allow lower priority interrupts to be recognized at the master. An example of this process is shown in the second application in the "Applications Examples" section.

Buffered Mode

The buffered mode is useful in large systems where buffering is required on the data bus. Although not limited to only 8259A cascading, it's most pertinent in this use. In the buffered mode, whenever the 8259A's data bus output is enabled, its $\overline{SP}/\overline{EN}$ pin will go low. This signal can be used to enable data transfer through a buffer transceiver in the required direction.

Figure 19 shows a conceptual diagram of three 8259A's in cascade, each slave is controlling an individual 8286 8-bit bidirectional bus driver by means of the buffered mode. Note the pull-up on the $\overline{SP}/\overline{EN}$. It is used to enable data transfer to the 8259A for its initial programming. When data transfer is to go from the 8259A to the processor, $\overline{SP}/\overline{EN}$ will go low; otherwise, it will be high.

A question should arise, however, from the fact that the $\overline{SP}/\overline{EN}$ pin is used to designate a master from a slave; how can it be used for both master-slave selection and buffer control? The answer to this is the provision for software programmable master-slave selection when in the buffer mode. The buffered mode is selected during each 8259A's initialization. At the same time, the user can assign each individual 8259A as a master or slave (see "Programming the 8259A").

4. PROGRAMMING THE 8259A

Programming the 8259A is accomplished by using two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). All the modes and commands explained in the previous section, "Operation of the 8259A", are programmable using the ICWs and OCWs (see Appendix A for cross reference). The ICWs are issued from the processor in a sequential format and are used to set-up the 8259A in an initial state of operation. The OCWs are issued as needed to vary and control 8259A operation.

Both ICWs and OCWs are sent by the processor to the 8259A via the data bus (8259A $\overline{CS}=0$, $\overline{WR}=0$). The 8259A distinguishes between the different ICWs and OCWs by the state of its A0 pin (controlled by processor addressing), the sequence they're issued in (ICWs only), and some dedicated bits among the ICWs and OCWs. Those bits which are dedicated are indicated so by fixed values (0 or 1) in the corresponding ICW or OCW programming formats which are covered shortly. Note, when issuing either ICWs or OCWs, the interrupt request pin of the processor should be disabled.

4.1 INITIALIZATION COMMAND WORDS (ICWs)

Before normal operation can begin, each 8259A in a system must be initialized by a sequence of two to four programming bytes called ICWs (Initialization Command Words). The ICWs are used to set-up the necessary conditions and modes for proper 8259A operation.

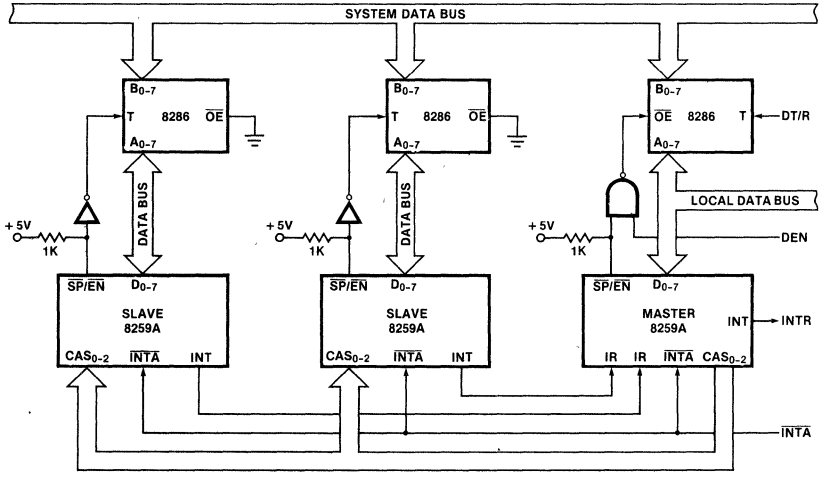


Figure 19. Cascade-Buffered Mode Example

Figure 20 shows the initialization flow of the 8259A. Both ICW1 and ICW2 must be issued for any form of 8259A operation. However, ICW3 and ICW4 are used only if designated so in ICW1. Determining the necessity and use of each ICW is covered shortly in individual groupings. Note that, once intialized, if any programming changes within the ICWs are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Certain internal set-up conditions occur automatically within the 8259A after the first ICW has been issued. These are:

A. Sequencer logic is set to accept the remaining ICWs as designated in ICW1.

B. The ISR (In-Service Register) and IMR (Interrupt Mask Register) are both cleared.

C. The special mask mode is reset.

D. The rotate in automatic EOI mode flip-flop is cleared.

E. The IRR (Interrupt Request Register) is selected for the read register command.

F. If the IC4 bit equals 0 in ICW1, all functions in ICW4 are cleared; 8080/8085 mode is selected by default.

G. The fully nested mode is entered with an initial priority assignment of IR0 highest through IR7 lowest.

H. The edge sense latch of each IR priority cell is cleared, thus requiring a low to high transition to generate an interrupt (edge triggered mode effected only).

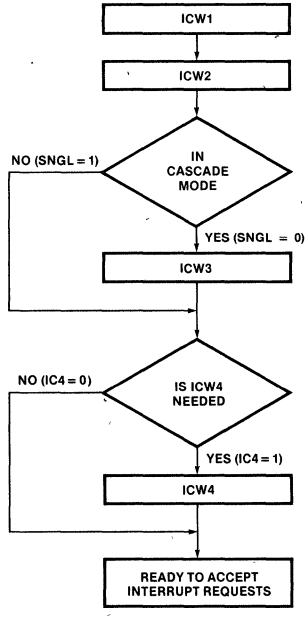
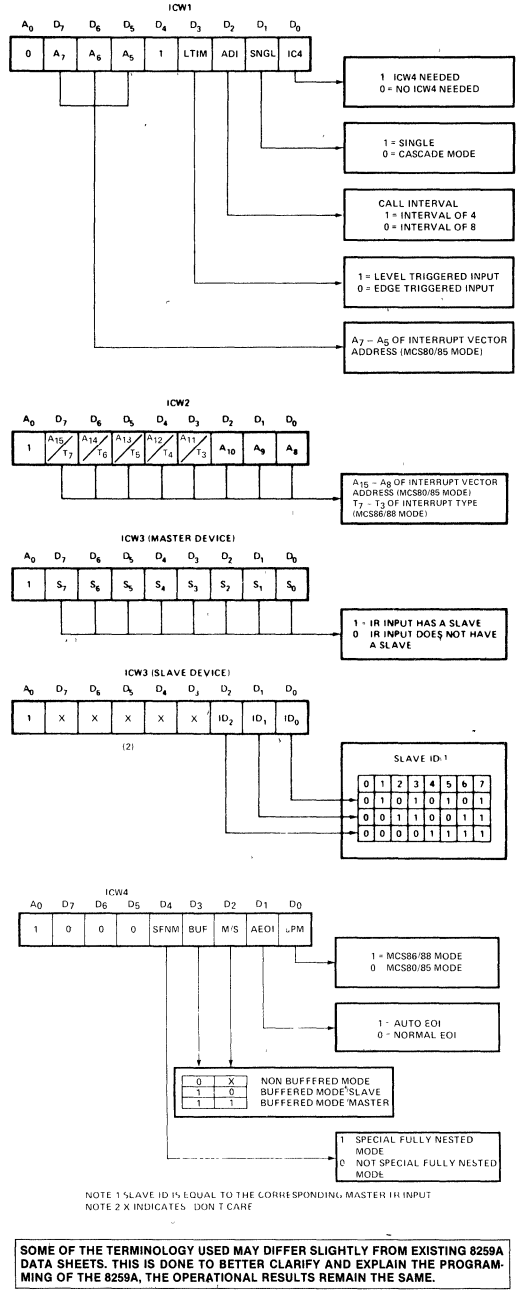


Figure 20. Initialization Flow

The ICW programming format, Figure 21, shows bit designation and a short definition of each ICW. With the ICW format as reference, the functions of each ICW will now be explained individually.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 21. Initialization Command Words (ICWS) Programming Format

ICW1 and ICW2

Issuing ICW1 and ICW2 is the minimum amount of programming needed for any type of 8259A operation. The majority of bits within these two ICWs are used to designate the interrupt vector starting address. The remaining bits serve various purposes. Description of the ICW1 and ICW2 bits is as follows:

IC4: The IC4 bit is used to designate to the 8259A whether or not ICW4 will be issued. If any of the ICW4 operations are to be used, ICW4 must equal 1. If they aren't used, then ICW4 needn't be issued and IC4 can equal 0. Note that if IC4 = 0, the 8259A will assume operation in the MCS-80/85 mode.

SNGL: The SNGL bit is used to designate whether or not the 8259A is to be used alone or in the cascade mode. If the cascade mode is desired, SNGL must equal 0. In doing this, the 8259A will accept ICW3 for further cascade mode programming. If the 8259A is to be used as the single 8259A within a system, the SNGL bit must equal 1; ICW3 won't be accepted.

ADI: The ADI bit is used to specify the address interval for the MCS-80/85 mode. If a 4-byte address interval is to be used, ADI must equal 1. For an 8-byte address interval, ADI must equal 0. The state of ADI is ignored when the 8259A is in the MCS-86/88 mode.

LTIM: The LTIM bit is used to select between the two IR input triggering modes. If LTIM = 1, the level triggered mode is selected. If LTIM = 0, the edge triggered mode is selected.

A5-A15: The A5-A15 bits are used to select the interrupt vector address when in the MCS-80/85 mode. There are two programming formats that can be used to do this. Which one is implemented depends upon the selected address interval (ADI). If ADI is set for the 4-byte interval, then the 8259A will automatically insert A0-A4 (A0, A1 = 0 and A2, A3, A4 = IR0-7). Thus A5-A15 must be user selected by programming the A5-A15 bits with the desired address. If ADI is set for the 8-byte interval, then A0-A5 are automatically inserted (A0, A1, A2 = 0 and A3, A4, A5 = IR0-7). This leaves A6-A15 to be selected by programming the A6-A15 bits with the desired address. The state of bit 5 is ignored in the latter format.

T3-T7: The T3-T7 bits are used to select the interrupt type when the MCS-86/88 mode is used. The programming of T3-T7 selects the upper 5 bits. The lower 3 bits are automatically inserted, corresponding to the IR level causing the interrupt. The state of bits A5-A10 will be ignored when in the MCS-86/88 mode. Establishing the actual memory address of the interrupt is shown in Figure 22.

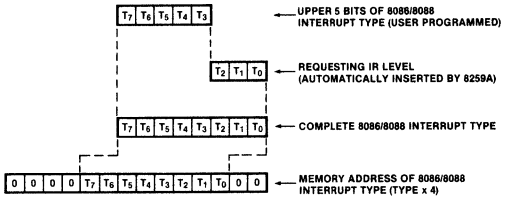


Figure 22. Establishing Memory Address of 8086/8088 Interrupt Type

ICW3

The 8259A will only accept ICW3 if programmed in the cascade mode (ICW1, SNGL = 0). ICW3 is used for specific programming within the cascade mode. Bit definition of ICW3 differs depending on whether the 8259A is a master or a slave. Definition of the ICW3 bits is as follows:

S0-7 (Master): If the 8259A is a master (either when the $\overline{SP}/\overline{EN}$ pin is tied high or in the buffered mode when M/S = 1 in ICW4), ICW3 bit definition is S0-7, corresponding to "slave 0-7". These bits are used to establish which IR inputs have slaves connected to them. A 1 designates a slave, a 0 no slave. For example, if a slave was connected to IR3, the S3 bit should be set to a 1. (S0) should be last choice for slave designation.

ID0-ID2 (Slave): If the 8259A is a slave (either when the $\overline{SP}/\overline{EN}$ pin is low or in the buffered mode when M/S = 0 in ICW4), ICW3 bit definition is used to establish its individual identity. The ID code of a particular slave must correspond to the number of the masters IR input it is connected to. For example, if a slave was connected to IR6 of the master, the slaves ID0-2 bits should be set to ID0 = 0, ID1 = 1, and ID2 = 1.

ICW4

The 8259A will only accept ICW4 if it was selected in ICW1 (bit IC4 = 1). Various modes are offered by using ICW4. Bit definition of ICW4 is as follows:

μPM: The μPM bit allows for selection of either the MCS-80/85 or MCS-86/88 mode. If set as a 1 the MCS-86/88 mode is selected, if a 0, the MCS-80/85 mode is selected.

AEOI: The AEOI bit is used to select the automatic end of interrupt mode. If AEOI = 1, the automatic end of interrupt mode is selected. If AEOI = 0, it isn't selected; thus an EOI command must be used during a service routine.

M/S: The M/S bit is used in conjunction with the buffered mode. If in the buffered mode, M/S defines whether the 8259A is a master or a slave. When M/S is set to a 1, the 8259A operates as the master; when M/S is 0, it operates as a slave. If not programmed in the buffered mode, the state of the M/S bit is ignored.

BUF: The BUF bit is used to designate operation in the buffered mode, thus controlling the use of the $\overline{SP}/\overline{EN}$ pin. If BUF is set to a 1, the buffered mode is programmed and $\overline{SP}/\overline{EN}$ is used as a transceiver enable output. If BUF is 0, the buffered mode isn't programmed and $\overline{SP}/\overline{EN}$ is used for master/slave selection. Note if ICW4 isn't programmed, $\overline{SP}/\overline{EN}$ is used for master/slave selection.

SFNM: The SFNM bit designates selection of the special fully nested mode which is used in conjunction with the cascade mode. Only the master should be programmed in the special fully nested mode to assure a truly fully nested structure among the slave IR inputs. If SFNM is set to a 1, the special fully nested mode is selected; if SFNM is 0, it is not selected.

4.2 OPERATIONAL COMMAND WORD (OCWs)

Once initialized by the ICWs, the 8259A will most likely be operating in the fully nested mode. At this point, operation can be further controlled or modified by the use of OCWs (Operation Command Words). Three OCWs are available for programming various modes and commands. Unlike the ICWs, the OCWs needn't be in any type of sequential order. Rather, they are issued by the processor as needed within a program.

Figure 23, the OCW programming format, shows the bit designation and short definition of each OCW. With the OCW format as reference, the functions of each OCW will be explained individually.

OCW1

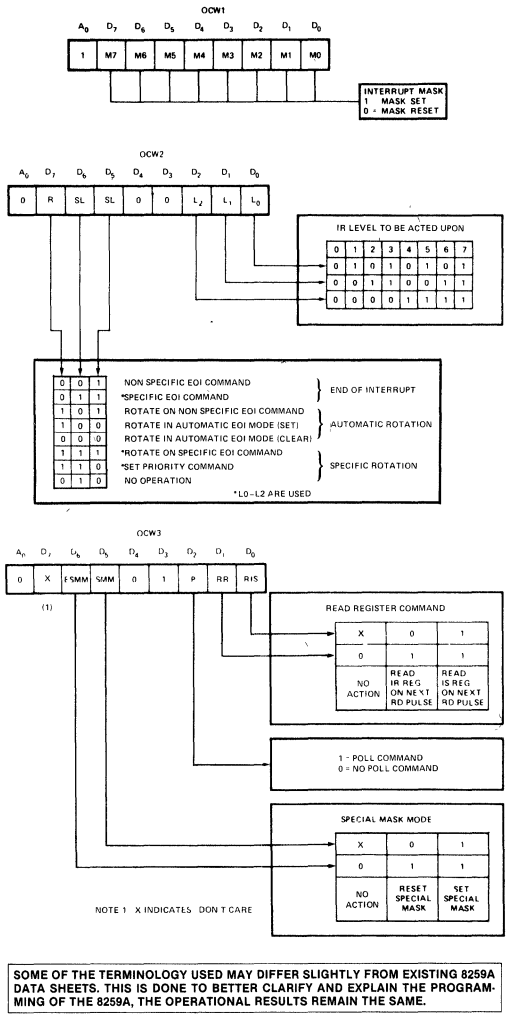
OCW1 is used solely for 8259A masking operations. It provides a direct link to the IMR (Interrupt Mask Register). The processor can write to or read from the IMR via OCW1. The OCW1 bit definition is as follows:

M0-M7: The M0-M7 bits are used to control the masking of IR inputs. If an M bit is set to a 1, it will mask the corresponding IR input. A 0 clears the mask, thus enabling the IR input. These bits convey the same meaning when being read by the processor for status update.

OCW2

OCW2 is used for end of interrupt, automatic rotation, and specific rotation operations. Associated commands and modes of these operations (with the exception of AEOI initialization), are selected using the bits of OCW2 in a combined fashion. Selection of a command or mode should be made with the corresponding table for OCW2 in the OCW programming format (Figure 20), rather than on a bit by bit basis. However, for completeness of explanation, bit definition of OCW2 is as follows:

L0-L2: The L0-L2 bits are used to designate an interrupt level (0-7) to be acted upon for the operation selected by the EOI, SL, and R bits of OCW2. The level designated will either be used to reset a specific ISR bit or to set a specific priority. The L0-L2 bits are enabled or disabled by the SL bit.



SOME OF THE TERMINOLOGY USED MAY DIFFER SLIGHTLY FROM EXISTING 8259A DATA SHEETS. THIS IS DONE TO BETTER CLARIFY AND EXPLAIN THE PROGRAMMING OF THE 8259A, THE OPERATIONAL RESULTS REMAIN THE SAME.

Figure 23. Operational Command Words (OCWs) Programming Format

EOI: The EOI bit is used for all end of interrupt commands (not automatic end of interrupt mode). If set to a 1, a form of an end of interrupt command will be executed depending on the state of the SL and R bits. If EOI is 0, an end of interrupt command won't be executed.

SL: The SL bit is used to select a specific level for a given operation. If SL is set to a 1, the L0-L2 bits are enabled. The operation selected by the EOI and R bits will be executed on the specified interrupt level. If SL is 0, the L0-L2 bits are disabled.

R: The R bit is used to control all 8259A rotation operations. If the R bit is set to a 1, a form of priority rotation will be executed depending on the state of SL and EOI bits. If R is 0, rotation won't be executed.

OCW3

OCW3 is used to issue various modes and commands to the 8259A. There are two main categories of operation associated with OCW3: interrupt status and interrupt masking. Bit definition of OCW3 is as follows:

RIS: The RIS bit is used to select the ISR or IRR for the read register command. If RIS is set to 1, ISR is selected. If RIS is 0, IRR is selected. The state of the RIS is only honored if the RR bit is a 1.

RR: The RR bit is used to execute the read register command. If RR is set to a 1, the read register command is issued and the state of RIS determines the register to be read. If RR is 0, the read register command isn't issued.

P: The P bit is used to issue the poll command. If P is set to a 1, the poll command is issued. If it is 0, the poll command isn't issued. The poll command will override a read register command if set simultaneously.

SMM: The SMM bit is used to set the special mask mode. If SMM is set to a 1, the special mask mode is selected. If it is 0, it is not selected. The state of the SMM bit is only honored if it is enabled by the ESMM bit.

ESMM: The ESMM bit is used to enable or disable the effect of the SMM bit. If ESMM is set to a 1, SMM is enabled. If ESMM is 0, SMM is disabled. This bit is useful to prevent interference of mode and command selections in OCW3.

5. APPLICATION EXAMPLES

In this section, the 8259A is shown in three different application examples. The first is an actual design implementation supporting an 8080A microprocessor system, "Power Fail/Auto Start with Battery Back-Up RAM". The second is a conceptual example of incorporating more than 64 interrupt levels in an 8080A or 8085A system, "78 Level Interrupt System". The third application is a conceptual design using an 8086 system, "Timer Controlled Interrupts". Although specific microprocessor systems are used in each example, these applications can be applied to either MCS-80, MCS-85, MCS-86, or MCS-88 systems, providing the necessary hardware and software changes are made. Overall, these applications should serve as a useful guide, illustrating the various procedures in using the 8259A.

5.1 POWER FAIL/AUTO-START WITH BATTERY BACK-UP RAM

The first application illustrates the 8259A used in an 8080A system, supporting a battery back-up scheme for the RAM (Random Access Memory) in a microcomputer system. Such a scheme is important in numerical and process control applications. The entire microcomputer system could be supported by a battery back-up scheme, however, due to the large amount of current usually required and the fact that most machinery is not supported by an auxiliary power source, only the state of calculations and variables usually need to be saved. In the event of a loss of power, if these items are not already stored in RAM, they can be transferred there and saved using a simple battery back-up system.

The vehicle used in this application is the Intel® SBC-80/20 Single Board Computer. An 8259A is used in the SBC-80/20 along with control lines helpful in implementing the power-down and automatic restart sequence used in a battery back-up system. The SBC-80/20 also contains user-selectable jumpers which allow the on-board RAM to be powered by a supply separate from the supply used for the non-RAM components. Also, the output of an undedicated latch is available to be connected to the IR inputs of the 8259A (the latch is cleared via an output port). In addition, an undedicated, buffered input line is provided, along with an input to the RAM decoder that will protect memory when asserted.

The additional circuitry to be described was constructed on an SBC-905 prototyping board. An SBC-635 power supply was used to power the non-RAM section of the SBC-80/20 while an external DC supply was used to simulate the back-up battery supplying power to the RAM. The SBC-635 was used since it provides an open collector ACLO output which indicates that the AC input line voltage is below 103/206 VAC (RMS).

The following is an example of a power-down and restart sequence that introduces the various power fail signals.

1. An AC power failure occurs and the ACLO goes high (ACLO is pulled up by the battery supply). This indicates that DC power will be reliable for at most 7.5 ms. The power fail circuitry generates a Power Fail Interrupt ($\overline{PFI}$) signal. This signal sets the $\overline{PFI}$ latch, which is connected to the IR0 input of the 8259A, and sets the Power Fail Sense (PFS) latch. The state of this latch will indicate to the processor, upon reset, whether it is coming up from a power failure (warm start) or if it is coming up initially (cold start).
2. The processor is interrupted by the 8259A when the PFI latch is set. This pushes the pre-power-down program counter onto the stack and calls the service routine for the IR0 input. The IR0 service routine saves the processor status and any other needed variables. The routine should end with a HALT instruction to minimize bus transitions.
3. After a predetermined length of time (5 ms in this example) the power fail circuitry generates a Memory Protect ($\overline{MPRO}$) signal. All processing for the power failure (including the interrupt response delays) must be completed within this 5 ms window. The $\overline{MPRO}$ signal ensures that spurious transitions on the system control bus caused by power going down do not alter the contents of the RAM.
4. DC power goes down.
5. AC power returns. The power-on reset circuitry on the SBC-80/20 generates a system RESET.
6. The processor reads the state of the $\overline{PFS}$ line to determine the appropriate start-up sequence. The PFS latch is cleared, the MPRO signal is removed, and the PFI latch driving IR0 is cleared by the Power Fail Sense Reset ($\overline{PFSR}$) signal. The system then continues from the pre-power-down location for a warm start by restoring the processor status and popping the pre-power-down program counter off the stack.

Figure 24 illustrates this timing.

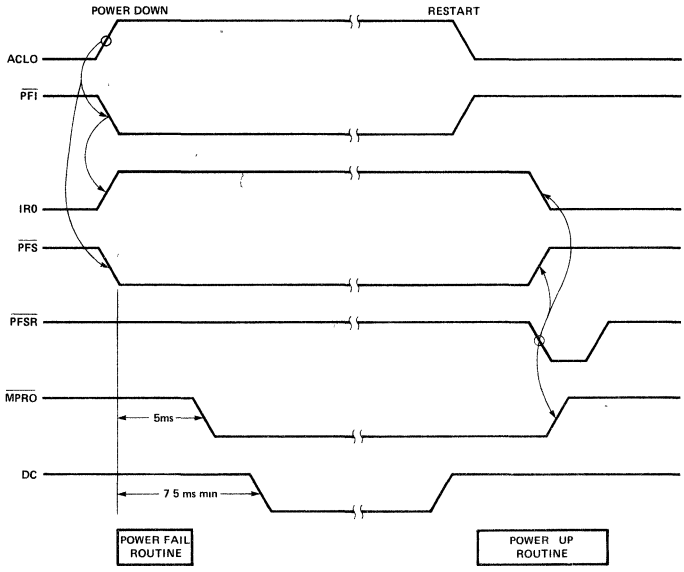


Figure 24. Power Down Restart Timing

Figure 25 shows the block diagram for the system. Notice that the RAM, the RAM decoder, and the power-down circuitry are powered by the battery supply.

The schematic of the power-down circuitry and the SBC-80/20 interface is shown in Figure 26. The design is very straightforward and uses CMOS logic to minimize the battery current requirements. The cold start switch is necessary to ensure that during a cold start, the $\overline{PFS}$ line is indicating "cold start" sense ($\overline{PFS}$ high). Thus, for a cold start, the cold start switch is depressed during power on. After that, no further action is needed. Notice that the $\overline{PFI}$ signal sets the on-board PFI latch. The output of this latch drives the 8259A IR0 input. This latch is cleared during the restart routine by executing an OUTput D4H instruction. The state of the $\overline{PFS}$ line may be read on the least significant data bus line (DB0) by executing an INput D4H instruction. An 8255 port (8255 #1, port C, bit 0) is used to control the $\overline{PFSR}$ line.

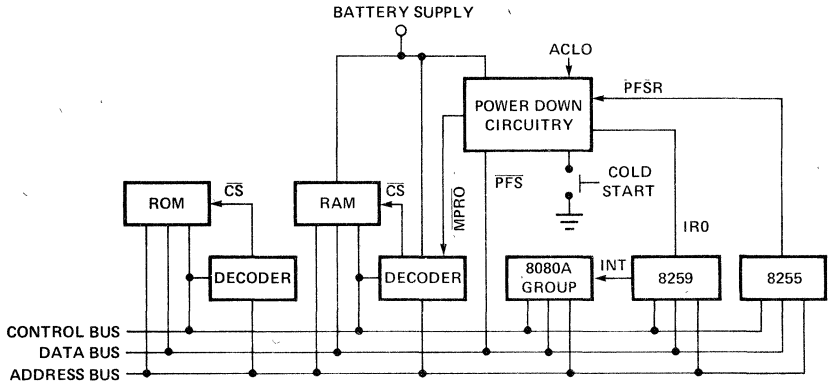


Figure 25. Block Diagram of SBC 80/20 with Power Down Circuit

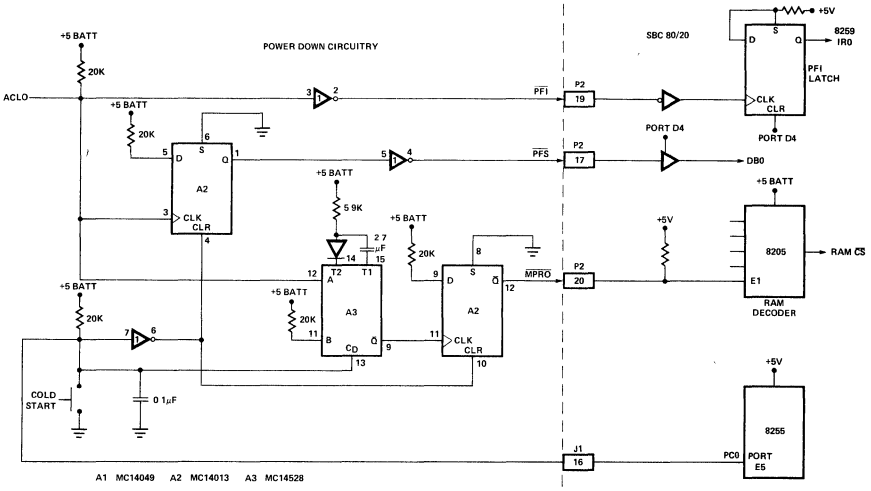


Figure 26. Power Down Circuit - SBC 80/20 Interface

The fully nested mode for the 8259A is used in its initial state to ensure the IR0 always has the highest priority. The remaining IR inputs can be used for any other purpose in the system. The only constraint is that the service routines must enable interrupts as early as possible. Obviously, this is to ensure that the power-down interrupt does not have to wait for service. If a rotating priority scheme is desired, another 8259A could be added as a slave and be programmed to operate in a rotating mode. The master would remain in the initial state of the fully nested mode so that the IR0 still remains the highest priority input.

The software to support the power-down circuitry is shown in Figure 27. The flow for each label will be discussed.

After any system reset, the processor starts execution at location 0000H (START). The $\overline{PFS}$ status is read and execution is transferred to CSTART if $\overline{PFS}$ indicates a cold start (i.e., someone is depressing the cold start switch) or WSTART if a warm start is indicated ($\overline{PFS}$ LOW). CSTART is the start of the user's program. The Stack Pointers (SP) and device initialization were included just to remind the reader that these must occur. The first EI instruction must appear after the 8259A has received its initialization sequence. The 8259A (and other devices) are initialized in the INIT subroutine.

When a power failure occurs, execution is vectored by the 8259A to REGSAV by way of the jump table at JSTART. The pre-power-down program counter is placed on the stack. REGSAV saves the processor registers and flags in the usual manner by pushing them onto the stack. Other items, such as output port status, programmable peripheral states, etc., are pushed onto the stack at this time. The Stack Pointer (SP) could be pushed onto the stack by way of the register pair HL but the top of the stack can exist anywhere in memory and there is no way then of knowing where that is when in the power-up routine. Thus, the SP is saved at a dedicated location in RAM. It isn't really necessary to send an EOI command to the 8259A in REGSAV since power will be removed from the 8259A, but one is included for completeness. The final instruction before actually losing power is a HALT. This minimizes somewhat spurious transitions on the various busses and lets the processor die gracefully.

On reset, when a warm start is detected, execution is transferred to WSTART. WSTART activates $\overline{PFSR}$ by way of the 8255 (all outputs go low then the 8255 is initialized). In the power-down circuitry, $\overline{PFSR}$ clears the PFS latch and removes the $\overline{MPRO}$ signal which then allows access to the RAM. WSTART also clears the PFI latch which arms the 8259A IR0 input. Then the 8259A is re-initialized along with any other devices. The SP is retrieved from RAM and the processor registers and flags are restored by popping them off the stack. Interrupts are then enabled. Now the power-down program counter is on top of the stack, so executing a RETurn instruction transfers the processor to exactly where it left off before the power failure.

Aside from illustrating the usefulness of the 8259A (and the SBC-80/20) in implementing a power failure protected microcomputer system, this application should also point out a way of preserving the processor status when using interrupts.

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    0  ;
                    1
                    2  ;POWER DOWN AND RESTART FOR THE SBC 88/20
                    3  ;
                    4  ;
                    5  ;SYSTEM EQUATES
000A                6  PT59A    EQU     0AH             ;8259 PORT WITH A0=0
000B                7  PT59B    EQU     0BH             ;8259 PORT WITH A0=1
00E7                8  PPI1CT   EQU     0E7H            ;8255 #1 CONTROL PORT
00E6                9  PPI1C    EQU     0E6H            ;8255 #1 PORT C
3800               10  SYSAVE   EQU     3800H           ;SP STORAGE IN RAM
0001               11  JPT      EQU     01H             ;MSB OF 8259 JUMP TABLE
                   12  ;
                   13  ;
                   14  ;STARTING POINT AFTER SYSTEM RESET
                   15
                   16  ;
0000               17          ORG     0H
0000 DBD4          18  START   IN      0D4H            ;READ PFS/ STATUS
0002 1F            19          RAR                     ;PFS/ IN DB0 PUT IN CARRY
0003 DA2001        20          JC      WSTART          ;PFS/=1  THEN COLD START
                   21
                   22  ;
                   23  ;WSTART LOCATION  PFS/=0  THEN WARM START
                   24  ;
                   25  ;
0006 3E80          26  WSTART  MVI     A,80H           ;SET 8255 #1 TO OUTPUT MODE
0008 D3E7          27          OUT     PPI1CT          ;8255 CONTROL PORT. PFCR/ GOES LOW
                   28  ;
                   29  ;OUTPUT COMMAND MAKES PFS/ GO LOW WHICH REMOVES MASTER RESET
                   30  ;CLEARS MTS LATCH
                   31  ;
000A 3C01          32          MVI     A,01H           ;RETURN PFCR/ HIGH
000C D3E6          33          OUT     PPI1C           ;8255 #1 PORT C
000E D3D4          34          OUT     0D4H            ;RESET MFI LATCH
0010 CD1D00        35          CALL    INIT            ;GO INITIALIZE EVERYTHING
0013 2A003A        36          LHLD    SYSAVE          ;RETRIEVE SP FROM RAM
0016 F9            37          SPHL                    ;PUT BACK INTO SP
0017 C1            38          POP     B               ;RESTORE BC
0018 D1            39          POP     D               ;RESTORE DE
0019 E1            40          POP     H               ;RESTORE HL
001A F1            41          POP     PSW             ;RESTORE A PLUS FLAGS
001B FB            42          EI                      ;ENABLE INTERRUPTS
001C C9            43          RET                     ;RPC-POWER-DOWN PC ON TOP OF STACK
                   44                                  ;RETURN TO IT
                   45
                   46  ;
                   47  ;INITIALIZATION ROUTINE  AT LEAST DO 8259 BUT OTHERS CAN BE ADDED
                   48
                   49
001D 3E16          50  INIT    MVI     A,16H           ;F=1,S=1,A7-A5=0  ICW1
001F D3DA          51          OUT     PT59A           ;8259 PORT WITH A0=0
0021 3C01          52          MVI     A,JPT           ;MSB OF JUMP TABLE ICW2
0023 D3DB          53          OUT     PT59B           ;8259 PORT WITH A0=1
                   54  ;

                   55  ;ADD ANY OTHER INITIALIZATIONS HERE
                   56  ;
0025 C9            57          RET                     ;RETURN
                   58  ;-----  ----------  ----  ---  ----------
                   59  ;
                   60  ;POWER DOWN ROUTINE TO SAVE REGISTERS AND STATUS
                   61  ;
                   62  ;
0026 F5            63  REGSAV  PUSH    PSW             ;SAVE A PLUS FLAGS
0027 E5            64          PUSH    H               ;SAVE HL
0028 D5            65          PUSH    D               ;SAVE DE
0029 C5            66          PUSH    B               ;SAVE BC
002A 210000        67          LXI     H,0000H         ;GET SP TO GET SP
002D 39            68          DAD     SP              ;SP NOW IN HL
002E 220038        69          SHLD    SYSAVE          ;SAVE SP IN RAM
                   70
                   71          EOI NOT REALLY NEEDED BUT INCLUDED FOR COMPLETENESS
                   72  ;
0031 3E20          73          MVI     A,20H           ;NON-SPECIFIC EOI
0033 D3DA          74          OUT     PT59A           ;8259 PORT WITH A0=0
0035 76            75          HLT                     ;HALT - GO DOWN GRACEFULLY
                   76  ;
                   77  ;
                   78  ;8259 JUMP TABLE  ONLY IR0 IS USED  OTHERS DIRECTED TO HALT
                   79  ;
                   80
0100               81          ORG     100H
0100 C32600        82  JSTART  JMP     REGSAV          ;IR0
0103 00            83          NOP
0104 C31038        84          JMP     3810H           ;IR1
0107 00            85          NOP
0108 C32038        86          JMP     3820H           ;IR2
010B 00            87          NOP
010C C33038        88          JMP     3830H           ;IR3
010F 00            89          NOP
0110 C34038        90          JMP     3840H           ;IR4
0113 00            91          NOP
0114 C35038        92          JMP     3850H           ;IR5
0117 00            93          NOP
0118 C36038        94          JMP     3860H           ;IR6
011B 00            95          NOP
011C C37038        96          JMP     3870H           ;IR7
011F 00            97          NOP
                   98  ;
                   99  ;
                  100  ;COLD START LOCATION  USER'S PROGRAM ENTERS HERE
                  101  ;
                  102  ;
0120 31003F       103  CSTART  LXI     SP,3F00H        ;INITIALIZE SP
0123 CD1D00       104          CALL    INIT            ;INITIALIZE EVERYTHING ELSE
0126 D3D4         105          OUT     0D4H            ;RESET MFI LATCH
0128 FB           106          EI                      ;ENABLE INTERRUPTS
                  107  ;
                  108  ;USER PROGRAM STARTS HERE
                  109  ;
                  110          END                     ;DONE
```

Figure 27. Power Down and Restart Software

5.2 78 LEVEL INTERRUPT SYSTEM

The second application illustrates an interrupt structure with greater than 64 levels for an 8080A or 8085A system. In the cascade mode, the 8259A supports up to 64 levels with direct vectoring to the service routine. Extending the structure to greater than 64 levels requires polling, using the poll command. A 78 level interrupt structure is used as an illustration; however, the principles apply to systems with up to 512 levels.

To implement the 78 level structure, 3 tiers of 8259A's are used. Nine 8259A's are cascaded in the master-slave scheme, giving 64 levels at tier 2. Two additional 8259A's are connected, by way of the INT outputs, to two of the 64 inputs. The 16 inputs at tier 3, combined with the 62 remaining tier 2 inputs, give 78 total levels. The fully nested structure is preserved over all levels, although direct vectoring is supplied for only the tier 2 inputs. Software is required to vector any tier 3 requests. Figure 28 shows the tiered structure used in this example. Notice that the tier 3 8259A's are connected to the bottom level slave (SA7). The master-slaves are interconnected as shown in "Interrupt Cascading", while the tier 3 8259A's are connected as "masters"; that is, the SP/EN pins are pulled high and the CAS pins are left unconnected. Since these 8259A's are only going to be used with the poll command, no INTA is required, therefore the INTA pins are pulled high.

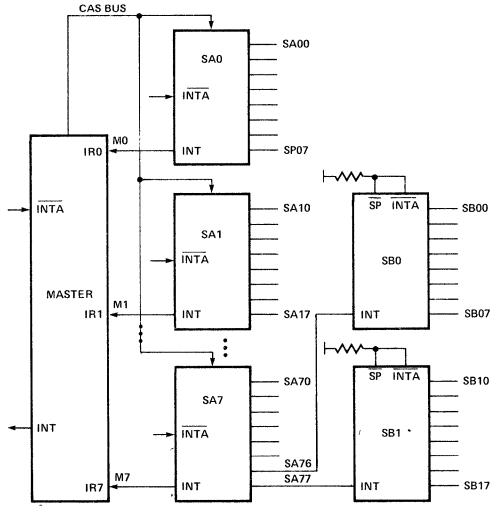


Figure 28. 78 Level Interrupt Structure

The concept used to implement the 78 levels is to directly vector to all tier 2 input service routines. If a tier 2 input contains a tier 3 8259A, the service routine for that input will poll the tier 3 8259A and branch to the tier 3 input service routine based on the poll word read after the poll command. Figure 29 shows how the jump table is organized assuming a starting location of 1000H and contiguous tables for all the tier 2 8259A's. Note that "SA35" denotes the IR5 input of the slave connected to the master IR3 input. Also note that for the normal tier 2 inputs, the jump table vectors the processor directly to the service routine for that input, while for the tier 2 inputs with 8259A's connected to their IR inputs, the processor is vectored to a service routine (i.e., SB0) which will poll to determine the actual tier 3 input requesting service. The polling routine utilizes the jump table starting at 1200H to vector the processor to the correct tier 3 service routine.

Each 8259A must receive an initialization sequence regardless of the mode. Since the tier 1 and 2 8259A's are in cascade and the special fully nested mode is used (covered shortly), all ICWs are required. The tier 3 8259A's don't require ICW3 or ICW4 since only polling will be used on them and they are connected as masters not in the cascade mode. The initialization sequence for each tier is shown in Figure 30. Notice that the master is initialized with a "dummy" jump table starting at 00H since all vectoring is done by the slaves. The tier 3 devices also receive "dummy" tables since only polling is used on tier 3.

As explained in "Interrupt Cascading", to preserve a truly fully nested mode within a slave, the master 8259A should be programmed in the special fully nested mode. This allows the master to acknowledge all interrupts at and above the level in service disregarding only those of lower priority. The special fully nested mode is programmed in the master only, so it only affects the immediate slaves (tier 2 not tier 3). To implement a fully nested structure among tier 3 slaves some special housekeeping software is required in all the tier-2-with-tier-3-slave routines. The software should simply save the state of the tier 2 IMR, mask all the lower tier 2 interrupts, then issue a specific EOI, resetting the ISR of the tier 2 interrupt level. On completion of the routine the IMR is restored.

Figure 31 shows an example flow and program for any tier 2 service routine without a tier 3 8259A. Figure 32 shows an example flow and program for any tier 2 service routine with a tier 3 8259A. Notice the reading of the ISR in both examples; this is done to determine whether or not to issue an EOI command to the master (refer to the section on "Special Fully Nested Mode" for further details).

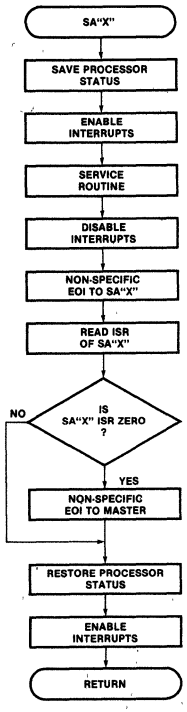
```
LOCATION    8259    CODE                COMMENTS
1000 H      SA0     JMP     SA00        ; SA00 SERVICE ROUTINE
   .
   .
101C H              JMP     SA07        ; SA07 SERVICE ROUTINE
1020 H      SA1     JMP     SA10        ; SA10 SERVICE ROUTINE
   .
   .
103C H              JMP     SA17        ; SA17 SERVICE ROUTINE
   .        .               .
   .        .               .          ; SA20–SA67 SERVICE ROUTINES
   .        .               .
10E0 H      SA7     JMP     SA70        ; SA70 SERVICE ROUTINE
   .
   .
10F8 H              JMP     SB0         ; SB0 POLL ROUTINE
10FC H              JMP     SB1         ; SB1 POLL ROUTINE
1200 H      SB0     JMP     SB00        ; SB00 SERVICE ROUTINE
   .
   .
121C H              JMP     SB07        ; SB07 SERVICE ROUTINE
1220 H      SB1     JMP     SB10        ; SB10 SERVICE ROUTINE
   .
   .
123C H              JMP     SB17        ; SB17 SERVICE ROUTINE
```

Figure 29. Jump Table Organization

```
; INITIALIZATION SEQUENCE FOR 78 LEVEL INTERRUPT STRUCTURE
;
; INITIALIZE MASTER
;
MINT:   MVI     A,15H       ; ICWI, LTM = 0, ADI = 1, S = 0, IC4 = 1
        OUT     MPTA        ; MASTER PORT A0 = 0
        MVI     A,00H       ; ICW2, DUMMY ADDRESS
        OUT     MPTB        ; MASTER PORT A0 = 1
        MVI     A,0FFH      ; ICW3, S7-S0 = 1
        OUT     MPTB        ; MASTER PORT A0 = 1
        MVI     A,I0H       ; ICW4, SFNM = 1
        OUT     MPTB        ; MASTER PORT A0 = 1
;
; INITIALIZE SA SLAVES - X DENOTES SLAVE ID (SEE KEY)
;
SAXINT: MVI     A,α         ; SEE KEY FOR ICW1, LTM = 0, ADI = 1, S = 0, IC4 = 1
        OUT     SAXPTA      ; SA"X" PORT A0 = 0
        MVI     A, 10H      ; ICW2, ADDRESS MSB
        OUT     SAXPTB      ; SA"X" PORT A0 = 1
        MVI     A0XH        ; ICW3, SA ID
        OUT     SAXPTB      ; SA"X" PORT A0 = 1
        MVI     A10H        ; ICW4, SFNM = 1
        OUT     SAXPTB      ; SA"X" PORT A0 = 1
;
; REPEAT ABOVE FOR EACH SA SLAVE
;
; INITIALIZE SB SLAVES - X DENOTES 0 or 1 (DO SB0, REPEAT FOR SB1)
;
SBXINT  MVI     A,I6H       ; ICW1, LTM = 0, ADI = 1, S = 1, IC4 = 0
        OUT     SBXPTA      ; SB"X" PORT A0 = 0
        MVI     A,00H       ; ICW2, DUMMY ADDRESS
        OUT     SBXPTB      ; SB"X" PORT A0 = 1
```

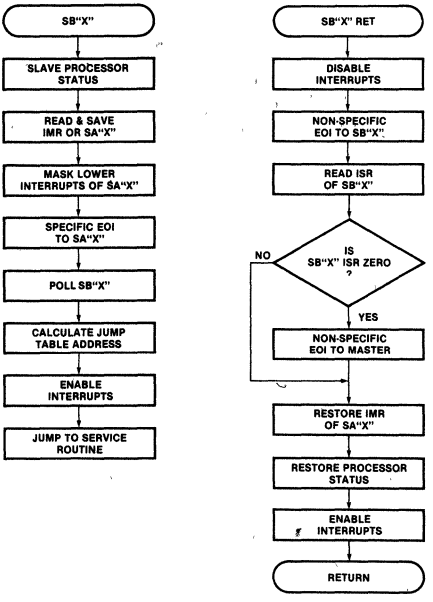
SA INITIALIZATION KEY		
SA"X"	α (ICW1)	JUMP TABLE START (H)
0	15	1000
1	35	1020
2	55	1040
3	75	1060
4	95	1080
5	B5	10A0
6	D5	10C0
7	F5	10E0

Figure 30. Initialization Sequence for 78 Level Interrupt Structure




```
; SA"X" ROUTINE - GENERAL INTERRUPT SERVICE ROUTINE
; FOR TIER 2 INTERRUPTS WITHOUT TIER 3 8259A
;
SAX     PUSH D          ; SAVE DE
        PUSH B          ; SAVE BC
        PUSH H          ; SAVE HL
        PUSH PSW        ; SAVE A, FLAGS
        EI              ; ENABLE INTERRUPTS
;
; SERVICE ROUTINE GOES HERE
;
        DI              ; DISABLE INTERRUPTS
        MVI   20H       ; OCW2, NON-SPECIFIC EOI
        OUT   SAXPTA    ; SA"X" PORT A0 = 0
        MUI   A,0BH     ; OCW3, READ REGISTER, ISR
        OUT   SAXPTA    ; SA"X" PORT A0 = 0
        IN    SAXPTA    ; SA"X" PORT A0 = 0, SA"X" ISR
        ANI   0FFH      ; TEST FOR ZERO
        JZN   SAXRSR    ; IF NOT ZERO, RESTORE STATUS
        MVI   A,08H     ; OCW2, NON SPECIFIC EOI
        OUT   MASPTA    ; MASTER PORT A0 = 0
SAXRSR  POP   PSW       ; RESTORE A, FLAGS
        POP   H         ; RESTORE HL
        POP   B         ; RESTORE BC
        POP   D         ; RESTORE DE
        EI              ; ENABLE INTERRUPTS
        RET             ; RETURN
```

Figure 31. Example Service Routine for Tier 2 Interrupt (SA"X") without Tier 3 8259A (SB"X")




```
; SB"X" ROUTINE - SERVICE ROUTINE FOR TIER 2
; INTERRUPTS WITH TIER 3 8259AS
SBX     PUSH D          ; SAVE DE
        PUSH B          ; SAVE BC
        PUSH H          ; SAVE HL
        PUSH PSW        ; SAVE A, FLAGS
        IN    SAXPTB    ; READ SA"X" IMR
        MOV   D,A       ; SAVE
        MVI   A,XXH     ; MASK SA"X" LOWER IR
        OUT   SAXPTB    ; SA"X" PORT A0 = 1
        MVI   A,6XH     ; OCW2 SPECIFIC EOI SA"X"
        OUT   SAXPTA    ; SA"X" PORT A0 = 1
        LXI   H,1200H   ; JUMP TABLE START
        MVI   B,00H     ; CLEAR B
        MVI   A,0CH     ; OCW3, POLL COMMAND
        OUT   SBXPTA    ; SB"X" PORT A0 = 0
        IN    SBXPTA    ; GET POLL WORD
        ANI   07H       ; LIMIT TO 3 BITS
        ADD   A         ; GET TABLE OFFSET
        ADD   A
        MOV   C,A       ; OFFSET TO C
        DAD   B         ; HL HAS TABLE ADDRESS
        EI              ; ENABLE INTERRUPTS
;
; SB"X"RET ROUTINE - FOR EOI AND MASK RESTORE
; AFTER SB"X" ROUTINE
;
SBXRET  DI              ; DISABLE INTERRUPTS
        MVI   A,20H     ; OCW2, NON SPECIFIC EOI
        OUT   SBXPTA    ; SA"X" PORT A0 = 0
        MVI   A,0BH     ; OCW3, READ REGISTER ISR
        OUT   SAXPTA    ; SA"X" PORT A0 = 0
        IN    SBXPTA    ; SA"X" PORT A0 = 0, ISR
        ANI   0FFH      ; TEST FOR ZERO
        JNZ   SBXRSR    ; IF ≠ 0 RESTORE IMR
        MVI   A,20H     ; OCW2, NON-SPECIFIC EOI
        OUT   MASPTA    ; MASTER PORT A0 = 0
SBXRSR  MOV   A,D       ; RESTORE SA"X" IMR
        OUT   SAXPTB    ; SA"X" PORT A0 = 1
        POP   PSW       ; RESTORE A, FLAGS
        POP   H         ; RESTORE HL
        POP   B         ; RESTORE BC
        POP   D         ; RESTORE BC
        EI              ; RESTORE DE
        RET             ; RETURN
```

Figure 32. Example Service Routine for Tier 2 Interrupt (SA"X") with Tier 3 8259A (SB"X")

5.3 TIMER CONTROLLED INTERRUPTS

In a large number of controller type microprocessor designs, certain timing requirements must be implemented throughout program execution. Such time dependent applications include control of keyboards, displays, CRTs, printers, and various facets of industrial control. These examples, however, are just a few of many designs which require device servicing at specific rates or generation of time delays. Trying to maintain these timing requirements by processor control alone can be costly in throughput and software complexity. So, what can be done to alleviate this problem? The answer, use the 8259A Programmable Interrupt Controller and external timing to interrupt the processor for time dependent device servicing.

This application example uses the 8259A for timer controlled interrupts in an 8086 system. External timing is done by two 8253 Programmable Interval Timers. Figure 33 shows a block diagram of the timer controlled interrupt circuitry which was built on the breadboard area of an SDK-86 (system design kit). Besides the 8259A and the 8253's, the necessary I/O decoding is also shown. The timer controlled interrupt circuitry interfaces with the SDK-86 which serves as the vehicle of operation for this design.

A short overview of how this application operates is as follows. The 8253's are programmed to generate interrupt requests at specific rates to a number of the 8259A IR inputs. The 8259A processes these requests by interrupting the 8086 and vectoring program execution to the appropriate service routine. In this example, the routines use the SDK-86 display panel to display the number of the interrupt level being serviced. These routines are merely for demonstration purposes to show the necessary procedures to establish the user's own routines in a timer controlled interrupt scheme.

Let's go over the operation starting with the actual interrupt timing generation which is done by two 8253 Programmable Interval Timers (8253 #1 and 8253 #2). Each 8253 provides three individual 16-bit counters (counters 0-2) which are software programmable by the processor. Each counter has a clock input (CLK), gate input (GATE), and an output (OUT). The output signal is based on divisions of the clock input signal. Just how or when the output occurs is determined by one of the 8253's six programmable modes, a programmable 16-bit count, and the state of the gate input.

Figure 34 shows the 8253 timing configuration used for generating interrupts to the 8259A. The SDK-86's PCLK (peripheral clock) signal provides a 400 ns period clock to CLK0 of 8253 #1. Counter 0 is used in mode 3 (square wave rate generator), and acts as a prescaler to provide the clock inputs of the other counters with a 10 ms period square wave. This 10 ms clock period made it easy to calculate exact timings for the other counters. Counter 2 of the 8253 #1 is used in mode 2 (rate generator), it is programmed to output a 10 ms pulse for every 200 pulses it receives (every 2 sec). The output of counter 2 causes an interrupt on IR1 of the 8259A. All the 8253 #2 counters are used in mode 5 (hardware triggered strobe) in which the gate input initiates counter operations. In this case the output of 8253 #1 counter 2 controls the gate of each 8253 #2 counter. When one of the 8253 #2 counters receive the 8253 #1 counter 2 output pulse on its gate, it will output a pulse (10 ms in duration) after a certain preprogrammed number of clock pulses have occurred. The programmed number of clock pulses for the 8253 #2 counters is as follows: 50 pulses (0.5 sec) for counter 0, 100 pulses (1 sec) for counter 1, and 150 pulses (1.5 sec) for counter 2. The outputs of these counters cause interrupt requests on IR2 through IR4 of the 8259A. Counter 1 of 8253 #1 is used in mode 0 (interrupt on terminal count). Unlike the other modes used which initialize operation automatically or by gate triggering, mode 0 allows software controlled counter initialization. When counter 1 of 8253 #1 is set during program execution, it will count 25 clocks (250 ms) and then pull its output high, causing an interrupt request on IR0 of the 8259A. Figure 35 shows the timing generated by the 8253's which cause interrupt request on the 8259A IR inputs.

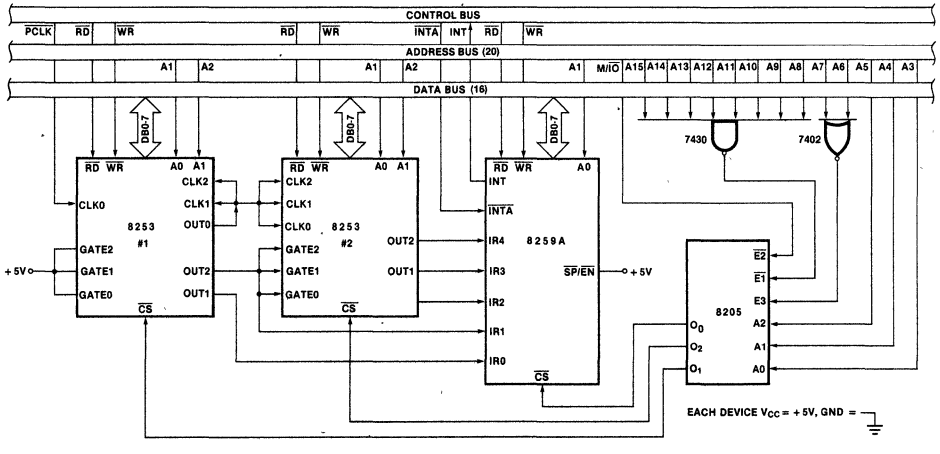


Figure 33. Timer Controlled Interrupt Circuit on SDK 86 Breadboard Area

121500-001

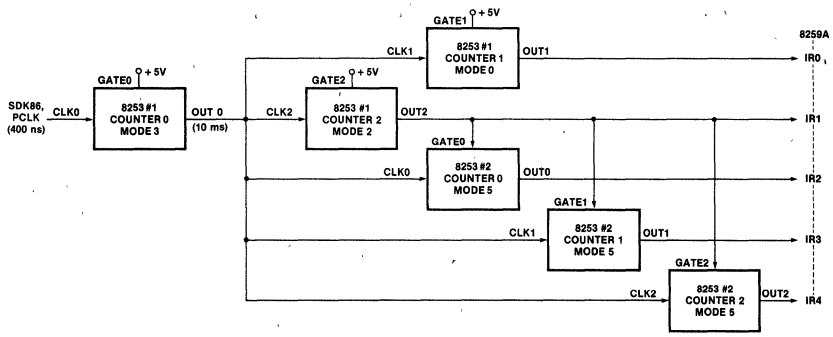


Figure 34. 8253 Timing Configuration for Timer Controlled Interrupts

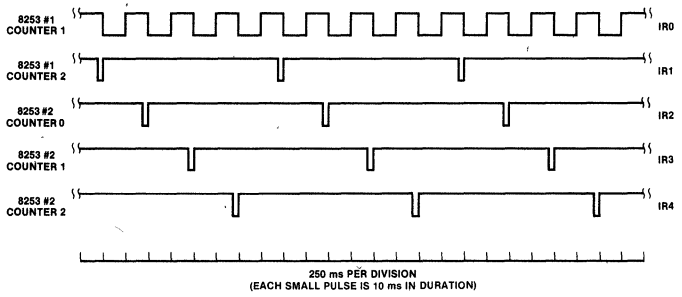


Figure 35. 8259A IR Input Signal From 8253S

There are basically two methods of timing generation that can be used in a timer controlled interrupt structure: dependent timing and independent timing. Dependent timing uses a single timing occurrence as a reference to base other timing occurrences on. On the other hand, independent timing has no mutual reference between occurrences. Industrial controller type applications are more apt to use dependent timing, whereas independent timing is prone to individual device control.

Although this application uses primarily dependent timing, independent timing is also incorporated as an example. The use of dependent timing can be seen back in Figure 34, where timing for IR2 through IR4 uses the IR1 pulse as reference. Each one of the 8253 #2 counters will generate an interrupt request a specific amount of times after the IR1 interrupt request occurs. When using the dependent method, as in this case, the IR2 through IR4 requests must occur before the next IR1 request. Independent timing is used to control the IR0 interrupt request. Note that its timing isn't controlled by any of the other IR requests. In this timer controlled interrupt configuration the dependent timing is initially set to be self running and the independent timing is software initialized. However, both methods can work either way by using the various 8253 modes to generate the same interrupt timing.

The 8259A processes the interrupts generated by the 8253's according to how it is programmed. In this application it is programmed to operate in the edge triggered mode, MCS-86/88 mode, and automatic EOI mode. In the edge triggered mode an interrupt request on an 8259A IR input becomes active on the rising edge. With this in mind, Figure 35 shows that IR0 will generate an interrupt every half second and IR1 through IR4 will each generate an interrupt every 2 seconds spaced apart at half second intervals. Interrupt vectoring in the MCS-86/88 mode is programmed so IR0, when activated, will select interrupt type 72. This means IR1 will select interrupt type 73, IR2 interrupt type 74, and so on through IR4. Since IR5 through IR7 aren't used, they are masked off. This prevents the possibility of any accidental interrupts and rids the necessity to tie the unused IR inputs to a steady level. Figure 36 shows the 8259A IR levels (IR0-IR4) with their corresponding interrupt type in the 8086 interrupt-vector table. Type 77 in the table is selected by a software "INT" instruction during program execution. Each type is programmed with the necessary code segment and instruction pointer values for vectoring to the appropriate service routine. Since the 8259A is programmed in the automatic EOI Mode, it doesn't require an EOI command to designate the completion of the service routine.

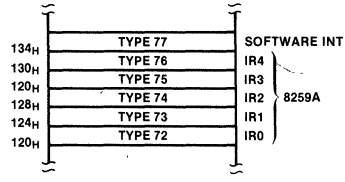


Figure 36. Interrupt "Type" Designation

As mentioned earlier, the interrupt service routines in this application are used merely to demonstrate the timer controlled interrupt scheme, not to implement a particular design. Thus a service routine simply displays the number of its interrupting level on the SDK-86 display panel. The display panel is controlled by the 8279 Keyboard and Display Controller. It is initialized to display "Ir" in its two left-most digits during the entire display sequence. When an interrupt from IR1 through IR4 occurs the corresponding routine will display its IR number via the 8279. During each IR1 through IR4 service routine a software "INT77" instruction is executed. This instruction vectors program execution to the service routine designated by type 77, which sets the 8253 counter controlling IR0 so it will cause an interrupt in 250 ms. When the IR0 interrupt occurs its routine will turn off the digit displayed by the IR1 through IR4 routines. Thus each IR level (IR1-IR4) will be displayed for 250 ms followed by a 250 ms off time caused by IR0. Figure 37 shows the entire display sequence of the timer controlled interrupt application.

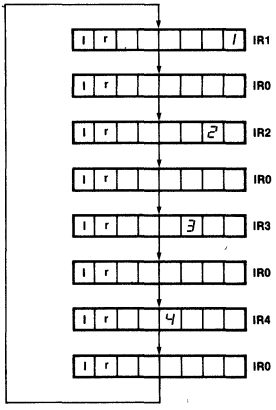


Figure 37. SDK Display Sequence for Timer Controlled Interrupts Program (Each Display Block Shown is 250 msec in Duration)

Now that we've covered the operation, let's move on to the program flow and structure of the timer controlled interrupt program. The program flow is made up of an initialization section and six interrupt service routines. The initialization program flow is shown in Figure 38. It starts by initializing some of the 8086's registers for program operation; this includes the extra segment, data segment, stack segment, and stack pointer. Next, by using the extra segement as reference, interrupt types 72 through 77 are set to vector interrupts to the appropriate routines. This is done by moving the code segment and instruction pointer values of each service routine into the corresponding type location. The 8253 counters are then programmed with the proper mode and count to provide the interrupt timing mentioned earlier. All counters with the exception of the 8253 #1, counter 1 are fully initialized at this point and will start counting. Counter 1 of 8253 #1 starts counting when its counter is loaded during the "INTR77" service routine, which will be covered shortly. Next, the 8259A is issued ICW1, ICW2, ICW4, and OCW1. The ICWs program the 8259A for the edge triggered mode, automatic EOI mode, and the proper interrupt vectoring (IR0, type 72). OCW1 is used to mask off the unused IR inputs (IR5-IR7). The 8279 is then set to display "IR" on its two left-most digits. After that the 8086 enables interrupts and a "dummy" main program is executed to wait for interrupt requests.

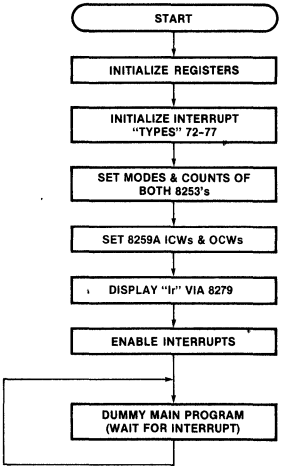


Figure 38. Initialization Program Flow for Timer Controlled Interrupts

There are six different interrupt service routines used in the program. Five of these routines, "INTR72" through "INTR76", are vectored to via the 8259A. Figure 39A-C shows the program flow for all six service routines. Note that "INTR73" through "INTR76" (IR1-IR4) basically use the same flow. These four similar routines display the number of its interrupting IR level on the SDK-86 display panel. The "INTR77" routine is vectored to by software during each of the previously mentioned routines and sets up interrupt timing to cause the "INTR72" (IR0) routine to be executed. The "INTR72" routine turns off the number on the SDK-86 display panel.

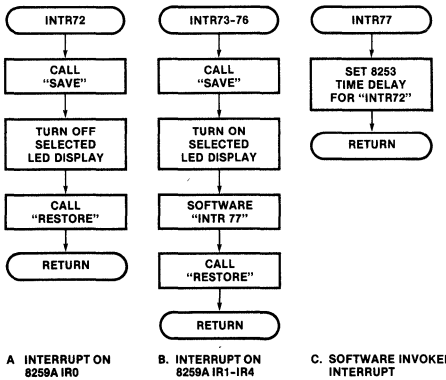


Figure 39. A-C. Interrupts Service Routine Flow for Timer Controlled Interrupts.

To best explain how these service routines work, let's assume an interrupt occurred on IR1 of the 8259A. The associated service routine for IR1 is "INTR73". Entering "INTR73", the first thing done is saving the pre-interrupt program status. This isn't really necessary in this program since a "dummy" main program is being executed; however, it is done as an example to show the operation. Rather than having code for saving the registers in each separate routine, a mutual call routine, "SAVE", is used. This routine will save the register status by pushing it on the stack. The next portion of "INTR73" will display the number of its IR level, "1", in the first digit of the SDK-86 display panel. After that, a software INT instruction is executed to vector program execution to the "INTR77" service routine. The "INTR77" service routine simply sets the 8253 #1 counter 1 to cause an interrupt on IR0 in 250 ms and then returns to "INTR73". Once back in "INTR73", the pre-interrupt status is restored by a call routine, "RESTORE". It does the opposite of "SAVE", returning the register status by popping it off the stack. The "INTR73" routine then returns to the "dummy " main program. The flow for the "INTR74" through "INTR76" routines are the same except for the digit location and the IR level displayed.

After 250 ms have elapsed, counter 1 of 8253 #1 makes an interrupt request on IR0 of the 8259A. This causes the "INTR72" service routine to be executed. Since this routine interrupts the main program, it also uses the "SAVE" routine to save pre-interrupt program status. It then turns off the digit displaying the IR level. In the case of the "INTR73" routine, the "1" is blanked out. The pre-interrupt status is then restored using the "RESTORE" routine and program execution returns to the "dummy" main program.

The complete program for the timer controlled interrupts application is shown in Appendix B. The program was executed in SDK-86 RAM starting at location 0500H (code segment = 0050, instruction pointer = 0).

CONCLUSION

This application note has explained the 8259A in detail and gives three applications illustrating the use of some of the numerous programmable features available. It should be evident from these discussions that the 8259A is an extremely flexible and easily programmable member of the Intel® MCS-80, MCS-85, MCS-86, and MCS-88 families.

APPENDIX A

This table is provided merely for reference information between the "Operation of the 8259A" and "Programming the 8259A" sections of this application note. It shouldn't be used as a programming reference guide (see "Programming the 8259A").

Operational Description	Command Words	Bits
MCS-80/85 Mode	ICW1, ICW4*	IC4, μPM*
Address Interval for MCS-80/85 Mode	ICW1	ADI
Interrupt Vector Address for MCS-80/85 Mode	ICW1, ICW2	A5-A15
MCS-86/88 Mode	ICW1, ICW4	IC4, μPM
Interrupt Vector Byte for MCS-86/88 Mode	ICW2	T3-T7
Fully Nested Mode	OCW-Default	—
Non-Specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SEOI, EOI, LO-L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate In Automatic EOI Mode	OCW2	R, SEOI, EOI
Set Priority Command	OCW2	L0-L2
Rotate on Specific EOI Command	OCW2	R, SEOI, EOI
Interrupt Mask Register	OCW1	M0-M7
Special Mask Mode	OCW3	ESMM-SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	ERIS, RIS
Read Register Command, ISR	OCW3	ERIS, RIS
Read IMR	OCW1	M0-M7
Poll Command	OCW3	P
Cascade Mode	ICW1, ICW3	SNGL, S0-7, ID0-2
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM
Buffered Mode	ICW1, ICW4	IC4, BUF, M/S

*Only needed if ICW4 is used for purposes other than μP mode set

```
MCS-86 ASSEMBLER    TCI59A                                                PAGE    1


ISIS-II MCS-86 ASSEMBLER V1.0 ASSEMBLY OF MODULE TCI59A
OBJECT MODULE PLACED IN :F1:TCI59A.OBJ
ASSEMBLER INVOKED BY: :F1:ASM86 :F1.TCI59A.SRC

LOC  OBJ                  LINE   SOURCE

                             1   ;******************** TIMER CONTROLLED INTERRUPTS ********************
                             2   ;
                             3   ;
                             4   ;
                             5   ;                EXTRA SEGMENT DECLARATIONS
                             6   ;
----                         7   EXTRA SEGMENT
                             8   ;
0120                         9           ORG     120H
0120 0401                   10   TP72IP  DW      INTR72          ;TYPE 72 INSTRUCTION POINTER
0122 ????                   11   TP72CS  DW      ?               ;TYPE 72 CODE SEGMENT
0124 1801                   12   TP73IP  DW      INTR73          ;TYPE 73 INSTRUCTION POINTER
0126 ????                   13   TP73CS  DW      ?               ;TYPE 73 CODE SEGMENT
0128 3001                   14   TP74IP  DW      INTR74          ;TYPE 74 INSTRUCTION POINTER
012A ????                   15   TP74CS  DW      ?               ;TYPE 74 CODE SEGMENT
012C 4801                   16   TP75IP  DW      INTR75          ;TYPE 75 INSTRUCTION POINTER
012E ????                   17   TP75CS  DW      ?               ;TYPE 75 CODE SEGMENT
0130 6001                   18   TP76IP  DW      INTR76          ;TYPE 76 INSTRUCTION POINTER
0132 ????                   19   TP76CS  DW      ?               ;TYPE 76 CODE SEGMENT
0134 7801                   20   TP77IP  DW      INTR77          ;TYPE 77 INSTRUCTION POINTER
0136 ????                   21   TP77CS  DW      ?               ;TYPE 77 CODE SEGMENT
                            22   ;
----                        23   EXTRA ENDS
                            24   ;
                            25   ;                DATA SEGMENT DECLARATIONS
                            26   ;
----                        27   DATA    SEGMENT
                            28   ;
0000 ????                   29   STACK1  DW      ?                       ;VARIABLE TO SAVE CALL ADDRESS
0002 ????                   30   AXTEMP  DW      ?                       ;VARIABLE TO SAVE AX REGISTER
0004 ??                     31   DIGIT   DB      ?                       ;VARIABLE TO SAVE SELECTED DIGIT
                            32   ;
----                        33   DATA    ENDS
                            34   ;
                            35   ;                CODE SEGMENT DECLARATION
                            36   ;
----                        37   CODE    SEGMENT
                            38   ;
                            39   ASSUME  ES:EXTRA,DS:DATA,CS:CODE
                            40   ;
                            41   ;                INITIALIZE REGISTERS
                            42   ;
0000 B80000                 43   START:  MOV     AX,0H                   ;EXTRA SEGMENT AT 0H
0003 8EC0                   44           MOV     ES,AX
0005 B87000                 45           MOV     AX,70H                  ;DATA SEGMENT AT 700H
0008 8ED8                   46           MOV     DS,AX
000A B87800                 47           MOV     AX,78H                  ;STACK SEGMENT AT 780H
000D 8ED0                   48           MOV     SS,AX
000F BC8000                 49           MOV     SP,80H                  ;STACK POINTER AT 80H (STACK=800H)
```

```
MCS-86 ASSEMBLER   TCI59A                                          PAGE   2

LOC  OBJ              LINE   SOURCE

                       50    ;
                       51    ;              LOAD INTERRUPT VECTOR TABLE
                       52    ;
0012 B80401            53    TYPE5.  MOV    AX,OFFSET (INTR72)      ;LOAD TYPE 72
0015 26A32001          54            MOV    TP72IP,AX
0019 268C0E2201        55            MOV    TP72CS,CS
001E B81801            56            MOV    AX,OFFSET (INTR73)      ;LOAD TYPE 73
0021 26A32401          57            MOV    TP73IP,AX
0025 268C0E2601        58            MOV    TP73CS,CS
002A B83001            59            MOV    AX,OFFSET (INTR74)      ;LOAD TYPE 74
002D 26A32801          60            MOV    TP74IP,AX
0031 268C0E2A01        61            MOV    TP74CS,CS
0036 B84801            62            MOV    AX,OFFSET (INTR75)      ;LOAD TYPE 75
0039 26A32C01          63            MOV    TP75IP,AX
003D 268C0E2E01        64            MOV    TP75CS,CS
0042 B86001            65            MOV    AX,OFFSET (INTR76)      ;LOAD TYPE 76
0045 26A33001          66            MOV    TP76IP,AX
0049 268C0E3201        67            MOV    TP76CS,CS
004E B87801            68            MOV    AX,OFFSET (INTR77)      ;LOAD TYPE 77
0051 26A33401          69            MOV    TP77IP,AX
0055 268C0E3601        70            MOV    TP77CS,CS
                       71    ;
                       72    ;              8253 INITIALIZATION
                       73    ;
005A BA0EFF            74    SET531: MOV    DX,0FF0EH               ;8253 #1 CONTROL WORD
005D B036              75            MOV    AL,36H                  ;COUNTER 0, MODE 3, BINARY
005F EE                76            OUT    DX,AL
0060 B071              77            MOV    AL,71H                  ;COUNTER 1, MODE 0, BCD
0062 EE                78            OUT    DX,AL
0063 B0B5              79            MOV    AL,0B5H                 ;COUNTER 2, MODE 2, BCD
0065 EE                80            OUT    DX,AL
0066 BA08FF            81            MOV    DX,0FF08H               ;LOAD COUNTER 0 (10MS)
0069 B0A8              82            MOV    AL,0A8H                 ;LSB
006B EE                83            OUT    DX,AL
006C B061              84            MOV    AL,61H                  ;MSB
006E EE                85            OUT    DX,AL
006F BA0CFF            86            MOV    DX,0FF0CH               ;LOAD COUNTER 2 (2SEC)
0072 B000              87            MOV    AL,00H                  ;LSB
0074 EE                88            OUT    DX,AL
0075 B002              89            MOV    AL,02H                  ;MSB
0077 EE                90            OUT    DX,AL
0078 BA16FF            91    SET532: MOV    DX,0FF16H               ;8253 #2 CONTROL WORD
007B B03B              92            MOV    AL,3BH                  ;COUNTER 0, MODE 5,BCD
007D EE                93            OUT    DX,AL
007E B07B              94            MOV    AL,7BH                  ;COUNTER 1, MODE 5, BCD
0080 EE                95            OUT    DX,AL
0081 B0BB              96            MOV    AL,0BBH                 ;COUNTER 2, MODE 5, BCD
0083 EE                97            OUT    DX,AL
0084 BA10FF            98            MOV    DX,0FF10H               ;LOAD COUNTER 0 (.5SEC)
0087 B050              99            MOV    AL,50H                  ;LSB
0089 EE               100            OUT    DX,AL
008A B000             101            MOV    AL,00H                  ;MSB
008C EE               102            OUT    DX,AL
008D BA12FF           103            MOV    DX,0FF12H               ;LOAD COUNTER 1 (1SEC)
0090 B000             104            MOV    AL,00H                  ;LSB
```

```
MCS-86 ASSEMBLER    TCI59A                                                PAGE  3

LOC  OBJ                LINE   SOURCE

0092 EE                  105           OUT     DX,AL
0093 B001                106           MOV     AL,01H                  ;MSB
0095 EE                  107           OUT     DX,AL
0096 BA14FF              108           MOV     DX,0FF14H               ;LOAD COUNTER 2 (1.5SEC)
0099 B050                109           MOV     AL,50H                  ;LSB
009B EE                  110           OUT     DX,AL
009C B001                111           MOV     AL,01H                  ;MSB
009E EE                  112           OUT     DX,AL
                         113   ;
                         114   ;               8259A INITIALIZATION
                         115   ;
009F BA00FF              116   SET59A: MOV     DX,0FF00H               ;8259A A0=0
00A2 B013                117           MOV     AL,13H                  ;ICW1-LTIM=0,S=1,IC4=1
00A4 EE                  118           OUT     DX,AL
00A5 BA02FF              119           MOV     DX,0FF02H               ;8259A A0=1
00A8 B048                120           MOV     AL, 48H                 ;ICW2-INTERRUPT TYPE 72 (120H)
00AA EE                  121           OUT     DX,AL
00AB B003                122           MOV     AL, 03H                 ;ICW4-SFNM=0,BUF=0,AEOI=1,MPM=1
00AD EE                  123           OUT     DX,AL
00AE B0E0                124           MOV     AL,0E0H                 ;OCW1-MASK IR5,6,7 (NOT USED)
00B0 EE                  125           OUT     DX,AL
                         126   ;
                         127   ;               8279 INITIALIZATION
                         128   ;
00B1 BAEAFF              129   SET79:  MOV     DX,0FFEAH               ;8279 COMMAND WORDS AND STATUS
00B4 B0D0                130           MOV     AL,0D0H                 ;CLEAR DISPLAY
00B6 EE                  131           OUT     DX,AL
00B7 EC                  132   WAIT79: IN      AL,DX                   ;READ STATUS
00B8 D0C0                133           ROL     AL,1                    ;"DU" BIT TO CARRY
00BA 72FB                134           JB      WAIT79                  ;JUMP IF DISPLAY IS UNAVAILABLE
00BC B087                135           MOV     AL, 87H                 ;DIGIT 8
00BE EE                  136           OUT     DX,AL
00BF BAE8FF              137           MOV     DX,0FFE8H               ;8279 DATA WORD
00C2 B006                138           MOV     AL,06H                  ;CHARACTER "1"
00C4 EE                  139           OUT     DX,AL
00C5 BAEAFF              140           MOV     DX,0FFEAH               ;8279 COMMAND WORD
00C8 B086                141           MOV     AL,86H                  ;DIGIT 7
00CA EE                  142           OUT     DX,AL
00CB BAE8FF              143           MOV     DX,0FFE8H               ;8279 DATA WORD
00CE B050                144           MOV     AL,50H                  ;CHARACTER "R"
00D0 EE                  145           OUT     DX,AL
00D1 FB                  146           STI                             ;ENABLE INTERRUPTS
                         147   ;
                         148   ;
                         149   ;               DUMMY PROGRAM
                         150   ;
00D2 EBFE                151   DUMMY:  JMP     DUMMY                   ;WAIT FOR INTERRUPT
                         152   ;
                         153   ;
00D4 A30200              154   SAVE:   MOV     AXTEMP,AX               ;SAVE AX
00D7 58                  155           POP     AX                      ;POP CALL RETURN ADDRESS
00D8 A30000              156           MOV     STACK1,AX               ;SAVE CALL RETURN ADDRESS
00DB A10200              157           MOV     AX,AXTEMP               ;RESTORE AX
00DE 50                  158           PUSH    AX                      ;SAVE PROCESSOR STATUS
00DF 53                  159           PUSH    BX
```

```
MCS-86 ASSEMBLER    TCI59A                                                PAGE    4

LOC  OBJ                LINE   SOURCE

00E0 51                 160            PUSH    CX
00E1 52                 161            PUSH    DX
00E2 55                 162            PUSH    BP
00E3 56                 163            PUSH    SI
00E4 57                 164            PUSH    DI
00E5 1E                 165            PUSH    DS
00E6 06                 166            PUSH    ES
00E7 A10000             167            MOV     AX,STACK1               ;RESTORE CALL RETURN ADDRESS
00EA 50                 168            PUSH    AX                      ;PUSH CALL RETURN ADDRESS
00EB C3                 169            RET
                        170    ;
00EC 58                 171    RESTOR: POP     AX                      ;POP CALL RETURN ADDRESS
00ED A30000             172            MOV     STACK1,AX               ;SAVE CALL RETURN ADDRESS
00F0 07                 173            POP     ES                      ;RESTORE PROCESSOR STATUS
00F1 1F                 174            POP     DS
00F2 5F                 175            POP     DI
00F3 5E                 176            POP     SI
00F4 5D                 177            POP     BP
00F5 5A                 178            POP     DX
00F6 59                 179            POP     CX
00F7 5B                 180            POP     BX
00F8 58                 181            POP     AX
00F9 A30200             182            MOV     AXTEMP,AX               ;SAVE AX
00FC A10000             183            MOV     AX,STACK1               ;RESTORE CALL RETURN ADDRESS
00FF 50                 184            PUSH    AX                      ;PUSH CALL RETURN ADDRESS
0100 A10200             185            MOV     AX,AXTEMP               ;RESTORE AX
0103 C3                 186            RET
                        187    ;
                        188    ;
                        189    ;              INTERRUPT 72, CLEAR DISPLAY, IR0 8259A
                        190    ;
0104 E8CDFF             191    INTR72: CALL    SAVE                    ;ROUTINE TO SAVE PROCESSOR STATUS
0107 BAEAFF             192            MOV     DX,0FFEAH               ;8279 COMMAND WORD
010A A00400             193            MOV     AL,DIGIT                ;SELECTED LED DIGIT
010D EE                 194            OUT     DX,AL
010E BAE8FF             195            MOV     DX,0FFE8H               ;8279 DATA
0111 B000               196            MOV     AL,00H                  ;BLANK OUT DIGIT
0113 EE                 197            OUT     DX,AL
0114 E8D5FF             198            CALL    RESTOR                  ;ROUTINE TO RESTORE PROCESSOR STATUS
0117 CF                 199            IRET                            ;RETURN FROM INTERRUPT
                        200    ;
                        201    ;
                        202    ;              INTERRUPT 73, IR1 8259A
                        203    ;
0118 E8B9FF             204    INTR73: CALL    SAVE                    ;ROUTINE TO SAVE PROCESSOR STATUS
011B BAEAFF             205            MOV     DX,0FFEAH               ;8279 COMMAND WORD
011E B080               206            MOV     AL,80H                  ;LED DISPLAY DIGIT 1
0120 A20400             207            MOV     DIGIT,AL
0123 EE                 208            OUT     DX,AL
0124 BAE8FF             209            MOV     DX,0FFE8H               ;8279 DATA
0127 B006               210            MOV     AL,06H                  ;CHARACTER "1"
0129 EE                 211            OUT     DX,AL
012A CD4D               212            INT     77                      ;TIMER DELAY FOR LED ON TIME
012C E8BDFF             213            CALL    RESTOR                  ;ROUTINE TO RESTORE PROCESSOR STATUS
012F CF                 214            IRET                            ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER   TCI59A                                          PAGE   5

LOC  OBJ              LINE   SOURCE

                       215   ;
                       216   ;
                       217   ;              INTERRUPT 74, IR2 8259A
                       218   ;
0130 E8A1FF            219   INTR74: CALL   SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
0133 BAEAFF            220           MOV    DX,0FFEAH           ;8279 COMMAND WORD
0136 B081              221           MOV    AL,81H              ;LED DISPLAY DIGIT 2
0138 A20400            222           MOV    DIGIT,AL
013B EE                223           OUT    DX,AL
013C BAE8FF            224           MOV    DX,0FFE8H           ;8279 DATA
013F B05B              225           MOV    AL,5BH              ;CHARACTER "2"
0141 EE                226           OUT    DX,AL
0142 CD4D              227           INT    77                  ;TIMER DELAY FOR LED ON TIME
0144 E8A5FF            228           CALL   RESTOR              ;ROUTINE TO RESTORE PROCESSOR STATUS
0147 CF                229           IRET                       ;RETURN FROM INTERRUPT
                       230   ;
                       231   ;
                       232   ;              INTERRUPT 75, IR3 8259A
                       233   ;
0148 E889FF            234   INTR75: CALL   SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
014B BAEAFF            235           MOV    DX,0FFEAH           ;8279 COMMAND WORD
014E B082              236           MOV    AL,82H              ;LED DISPLAY DIGIT 3
0150 A20400            237           MOV    DIGIT,AL
0153 EE                238           OUT    DX,AL
0154 BAE8FF            239           MOV    DX,0FFE8H           ;8279 DATA
0157 B04F              240           MOV    AL,4FH              ;CHARACTER "3"
0159 EE                241           OUT    DX,AL
015A CD4D              242           INT    77                  ;TIMER DELAY FOR LED ON TIME
015C E88DFF            243           CALL   RESTOR              ;ROUTINE TO RESTORE PROCESSOR STATUS
015F CF                244           IRET                       ;RETURN FROM INTERRUPT
                       245   ;
                       246   ;
                       247   ;              INTERRUPT 76, IR4 8259A
                       248   ;
0160 E871FF            249   INTR76: CALL   SAVE                  ;ROUTINE TO SAVE PROCESSOR STATUS
0163 BAEAFF            250           MOV    DX,0FFEAH           ;8279 COMMAND WORD
0166 B083              251           MOV    AL,83H              ;LED DISPLAY DIGIT 4
0168 A20400            252           MOV    DIGIT,AL
016B EE                253           OUT    DX,AL
016C BAE8FF            254           MOV    DX,0FFE8H           ;8279 DATA
016F B066              255           MOV    AL,66H              ;CHARACTER "4"
0171 EE                256           OUT    DX,AL
0172 CD4D              257           INT    77                  ;TIMER DELAY FOR LED ON TIME
0174 E875FF            258           CALL   RESTOR              ;ROUTINE TO RESTORE PROCESSOR STATUS
0177 CF                259           IRET                       ;RETURN FROM INTERRUPT
                       260   ;
                       261   ;
                       262   ;              INTERRUPT 77, TIMER DELAY, SOFTWARE CONTROLLED
                       263   ;
0178 BA0AFF            264   INTR77: MOV    DX,0FF0AH           ;LOAD COUNTER 1 8253 #1 (250 MSEC)
017B B025              265           MOV    AL,25H              ;LSB
017D EE                266           OUT    DX,AL
017E B000              267           MOV    AL,00H              ;MSB
0180 EE                268           OUT    DX,AL
0181 CF                269           IRET                       ;RETURN FROM INTERRUPT
```

```
MCS-86 ASSEMBLER    TC159A                                          PAGE    6


LOC  OBJ                LINE   SOURCE

                        270    ;
                        271    ;
----                    272    CODE    ENDS;
                        273    ;
                        274    ;
  0000                  275            END     START

SYMBOL TABLE LISTING
------ ----- -------


NAME    TYPE     VALUE  ATTRIBUTES

??SEG . SEGMENT         SIZE=0000H PARA PUBLIC
AXTEMP  V WORD   0002H  DATA
CODE    SEGMENT         SIZE=0182H PARA
DATA    SEGMENT         SIZE=0005H PARA
DIGIT . V BYTE   0004H  DATA
DUMMY . L NEAR   00D2H  CODE
EXTRA   SEGMENT         SIZE=0138H PARA
INTR72  L NEAR   0104H  CODE
INTR73  L NEAR   0118H  CODE
INTR74  L NEAR   0130H  CODE
INTR75  L NEAR   0148H  CODE
INTR76. L NEAR   0160H  CODE
INTR77  L NEAR   0178H  CODE
RESTOR  L NEAR   00ECH  CODE
SAVE.   L NEAR   00D4H  CODE
SET531  L NEAR   005HH  CODE
SET532. L NEAR   0078H  CODE
SET59A  L NEAR   009FH  CODE
SET79   L NEAR   00B1H  CODE
STACK1  V WORD   0000H  DATA
START . L NEAR   0000H  CODE
TP72CS  V WORD   0122H  EXTRA
TP72IP  V WORD   0120H  EXTRA
TP73CS. V WORD   0126H  EXTRA
TP73IP. V WORD   0124H  EXTRA
TP74CS  V WORD   012AH  EXTRA
TP74IP. V WORD   0128H  EXTRA
TP75CS  V WORD   012EH  EXTRA
TP75IP. V WORD   012CH  EXTRA
TP76CS. V WORD   0132H  EXTRA
TP76IP  V WORD   0130H  EXTRA
TP77CS. V WORD   0136H  EXTRA
TP77IP  V WORD   0134H  EXTRA
TYPES   L NEAR   0012H  CODE
WAIT79  L NEAR   00B7H  CODE


ASSEMBLY COMPLETE, NO ERRORS FOUND
```

iAPX 86, 88, 186, 188 Microprocessors

3

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.

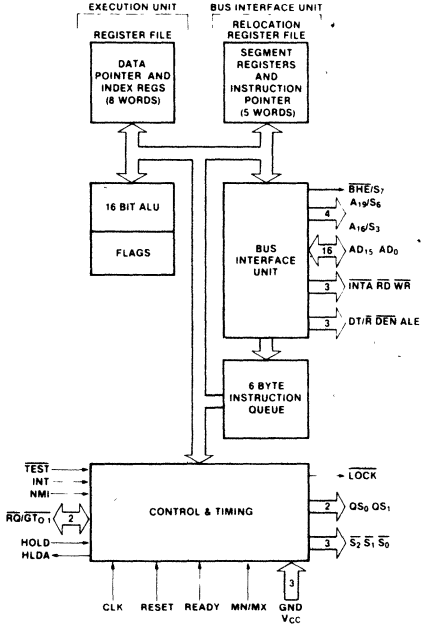
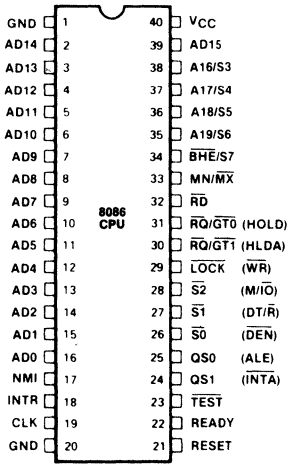


Figure 1. iAPX 86/10 CPU Block Diagram



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.

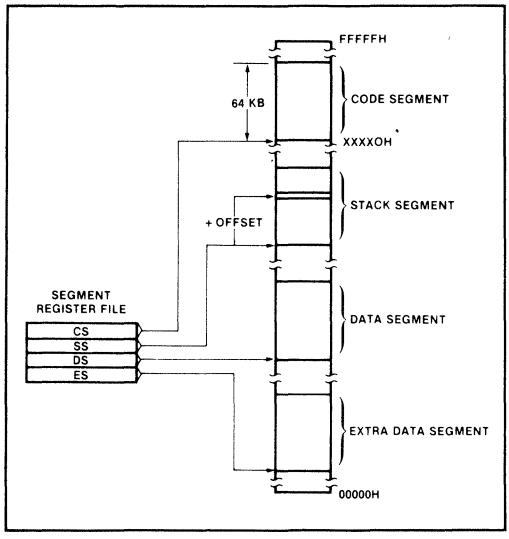


Figure 3a. Memory Organization

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 FFFF0; 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.

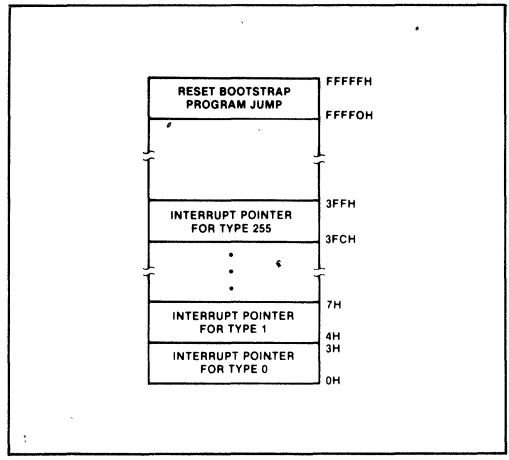


Figure 3b. Reserved Memory Locations

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.

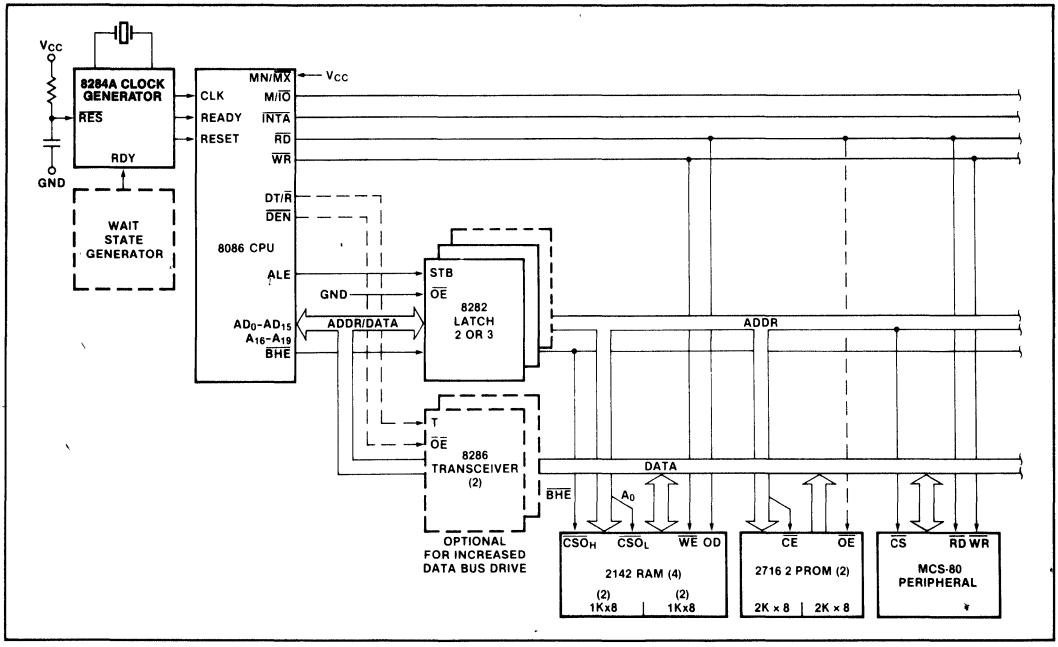


Figure 4a. Minimum Mode iAPX 86/10 Typical Configuration

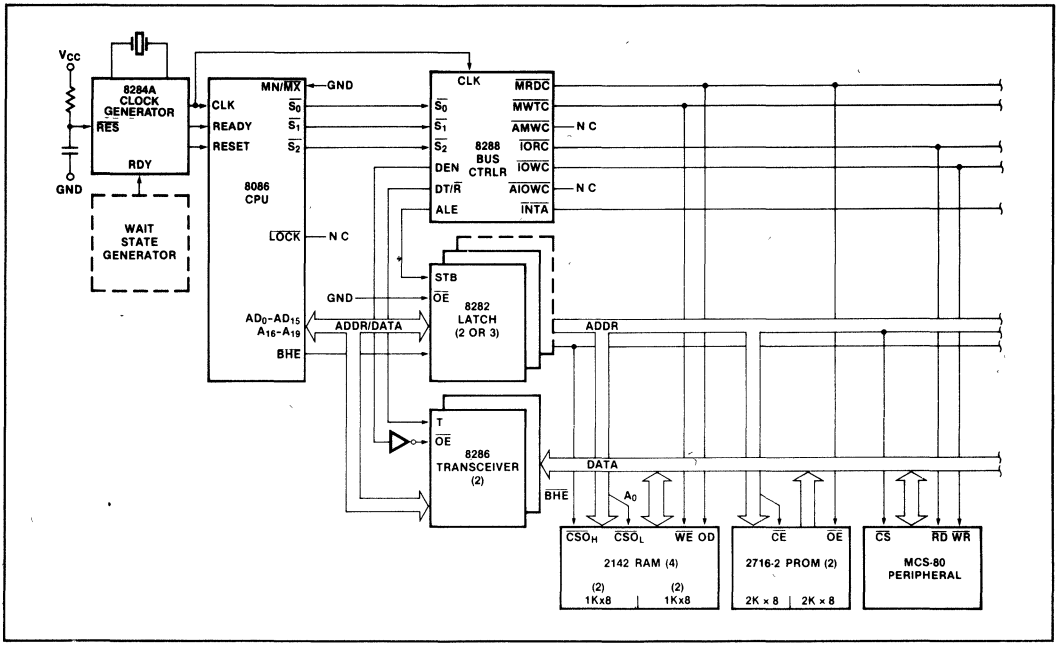


Figure 4b. Maximum Mode iAPX 86/10 Typical Configuration

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.

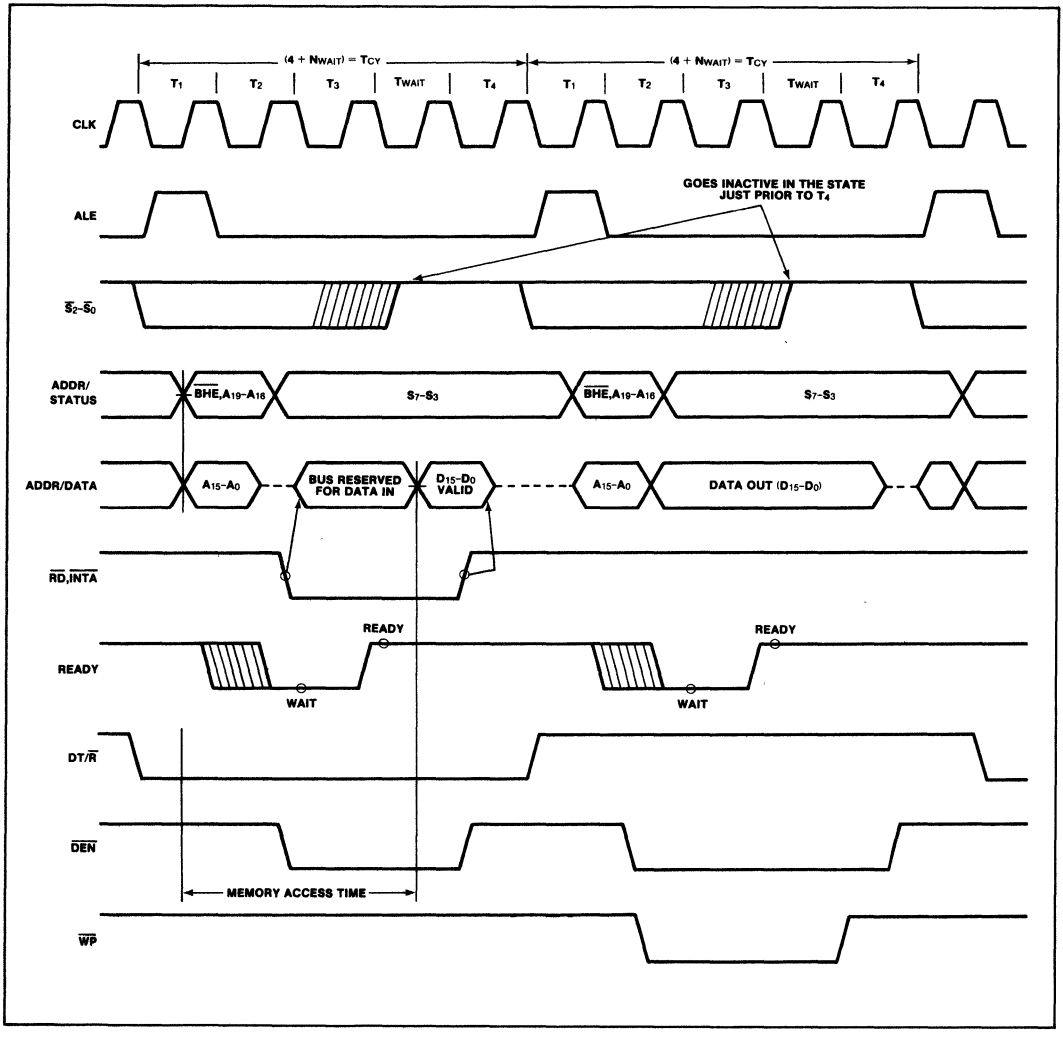


Figure 5. Basic System Timing

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

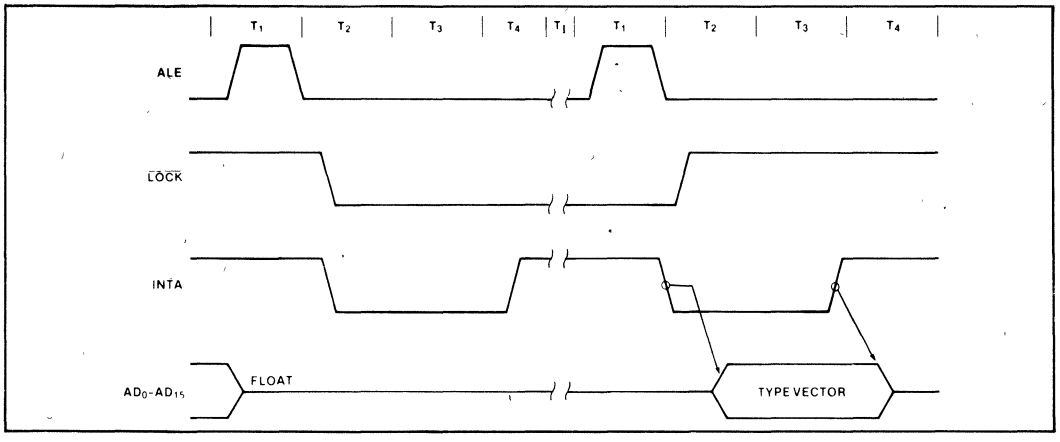


Figure 6. Interrupt Acknowledge Sequence

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.

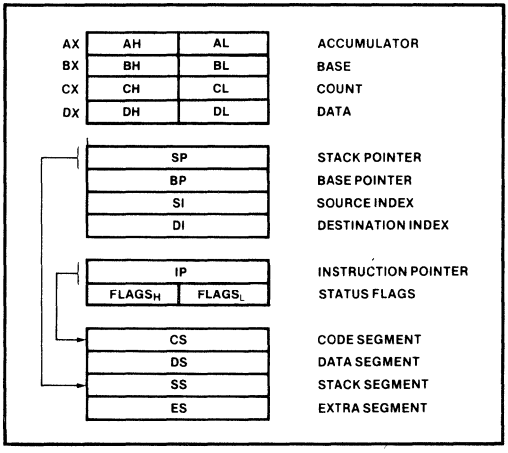


Figure 7. iAPX 86/10 Register Model

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

Note. 1 V_{IL} tested with MN/$\overline{MX}$ Pin = 0V.
2. V_{IH} tested with MN/$\overline{MX}$ Pin = 5V
MN/$\overline{MX}$ Pin is a Strap Pin

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

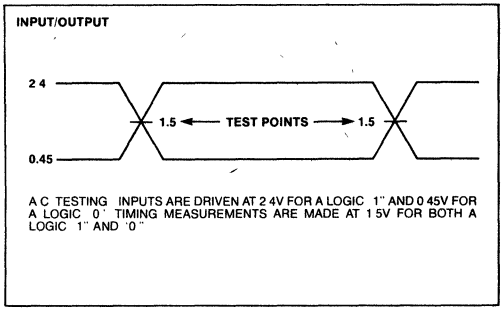
A.C. CHARACTERISTICS (Continued)

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

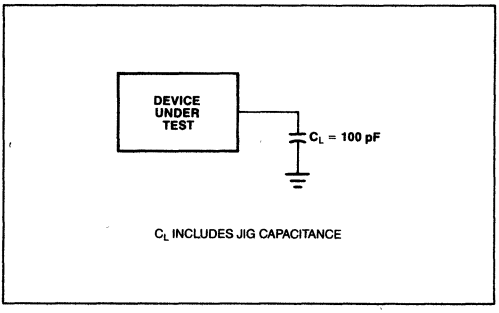
NOTES:
1. Signal at 8284A shown for reference only.
2. Setup requirement for asynchronous signal only to guarantee recognition at next CLK.
3. Applies only to T2 state. (8 ns into T3).

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING 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".

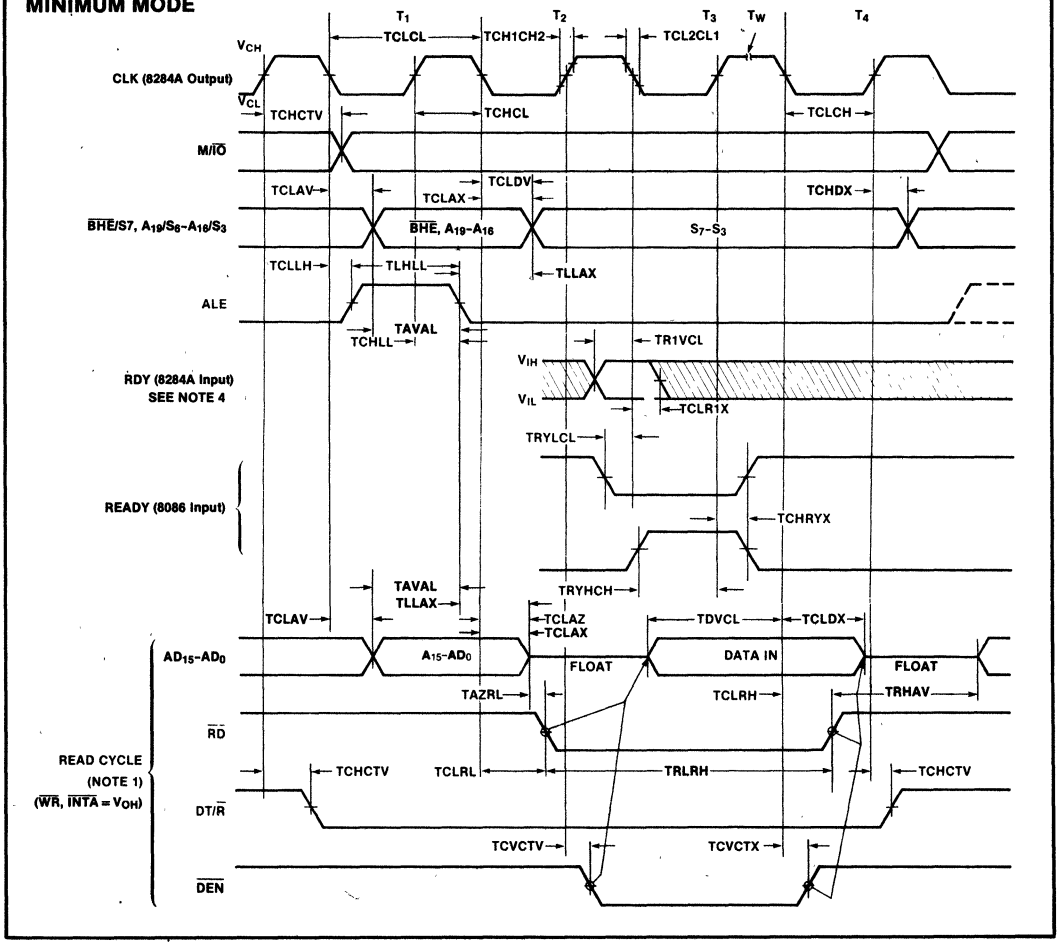
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

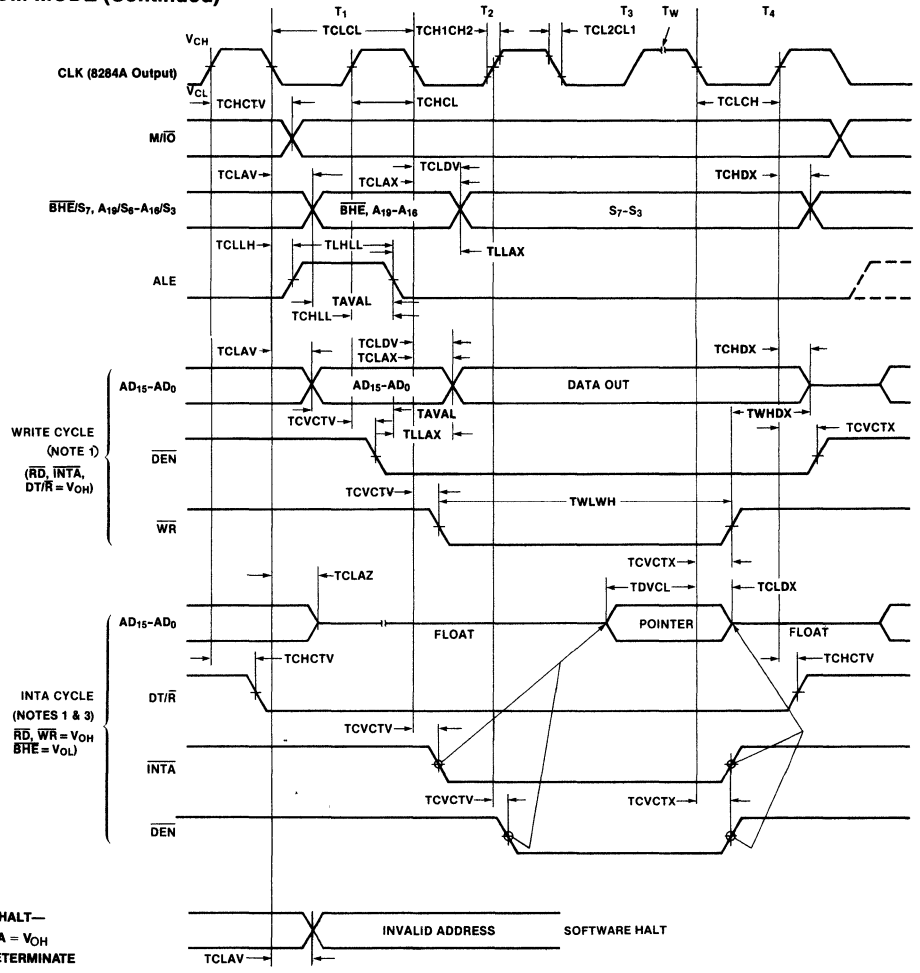
WAVEFORMS

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).

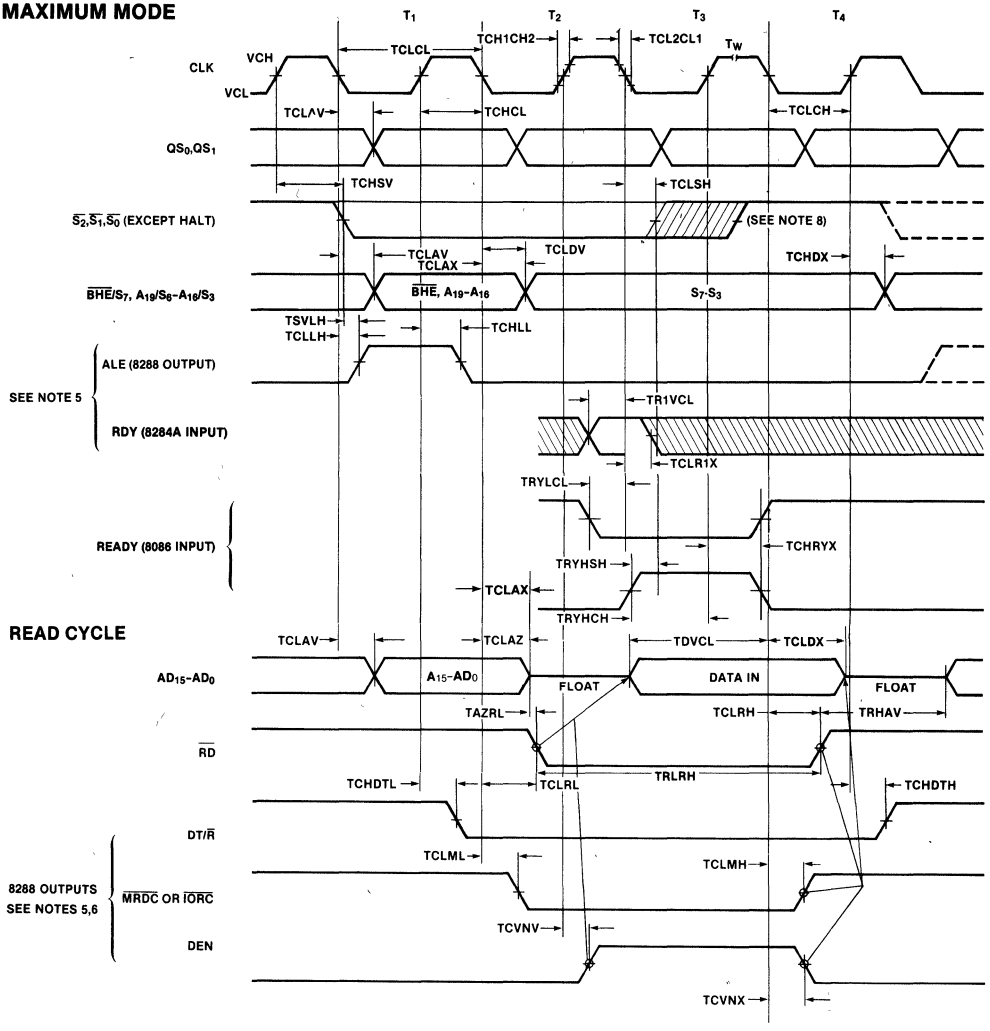
A.C. CHARACTERISTICS (Continued)

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

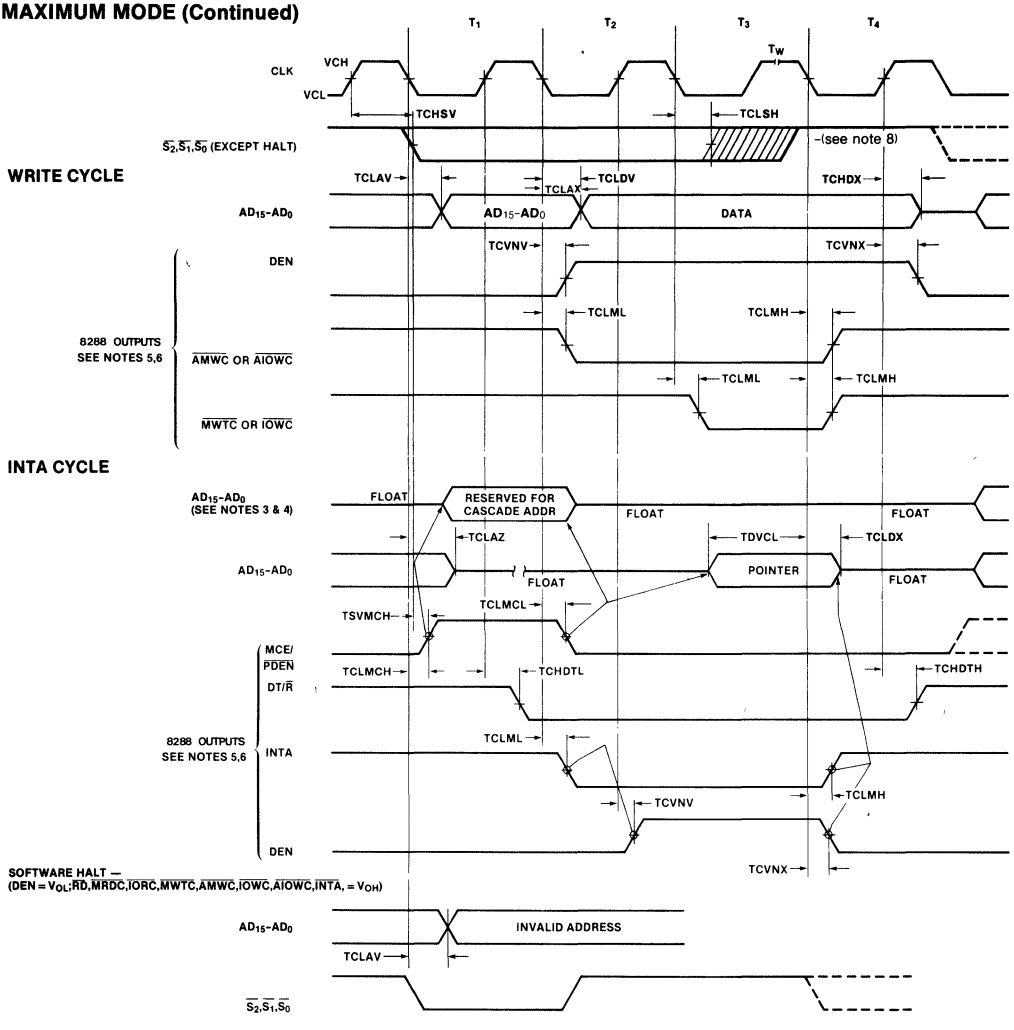
WAVEFORMS

MAXIMUM MODE



WAVEFORMS (Continued)

MAXIMUM MODE (Continued)

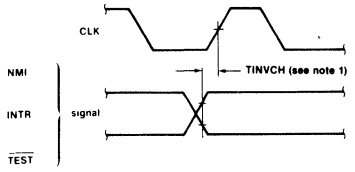


NOTES:

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.

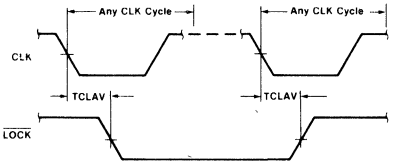
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

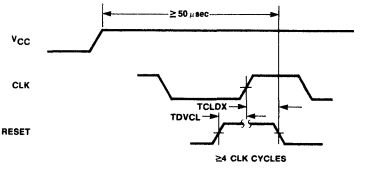


NOTE 1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

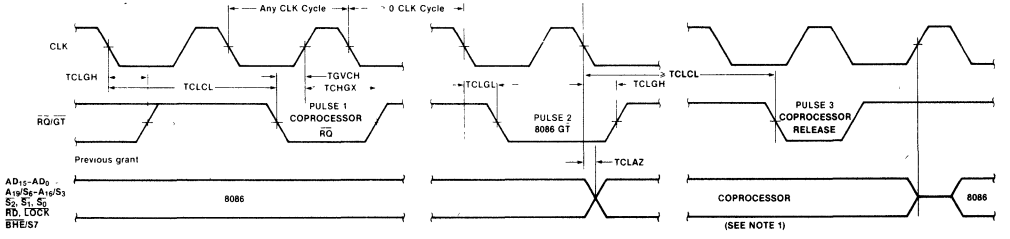
BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)



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)

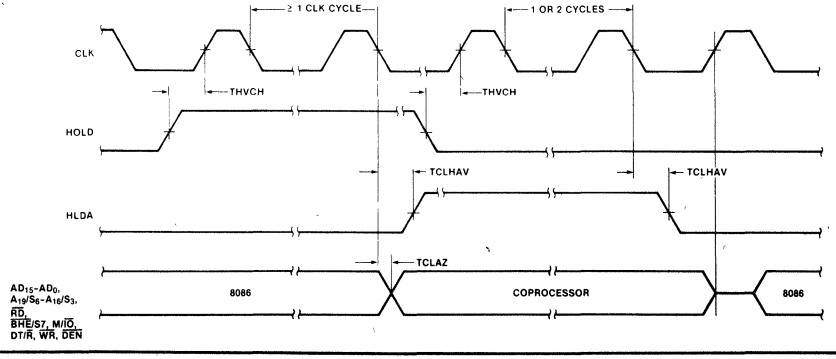


Table 2. Instruction Set Summary

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	mod0reg r/m		
Segment register to register/memory	10001100	mod0reg r/m		

PUSH = Push

Register/memory	11111111	mod 110 r/m
Register	01010 reg	
Segment register	000reg110	

POP = Pop

Register/memory	10001111	mod 000 r/m
Register	01011 reg	
Segment register	000reg111	

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	mod000 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	mod010 r/m	data	data if s w=01
Immediate to accumulator	0001010w	data	data if w=1	

INC = Increment

Register/memory	1111111w	mod000 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	mod101 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	mod011 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	mod001 r/m		
Register	01001 reg			
NEG Change sign	1111011w	mod011 r/m		

CMP Compare

Register/memory and register	001110dw	mod reg r/m		
Immediate with register/memory	100000sw	mod111 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	mod100 r/m		
IMUL Integer multiply (signed)	1111011w	mod101 r/m		
AAM ASCII adjust for multiply	11010100	00001010		
DIV Divide (unsigned)	1111011w	mod110 r/m		
IDIV Integer divide (signed)	1111011w	mod111 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	mod010 r/m

SHL/SAL Shift logical/arithmetic left	110100vw	mod100 r/m
SHR Shift logical right	110100vw	mod101 r/m
SAR Shift arithmetic right	110100vw	mod111 r/m
ROL Rotate left	110100vw	mod000 r/m
ROR Rotate right	110100vw	mod001 r/m
RCL Rotate through carry flag left	110100vw	mod010 r/m
RCR Rotate through carry right	110100vw	mod011 r/m

AND And

Reg /memory and register to either	001000dw	mod reg r/m		
Immediate to register/memory	1000000w	mod100 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	mod000 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	mod001 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	mod110 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)

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 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**

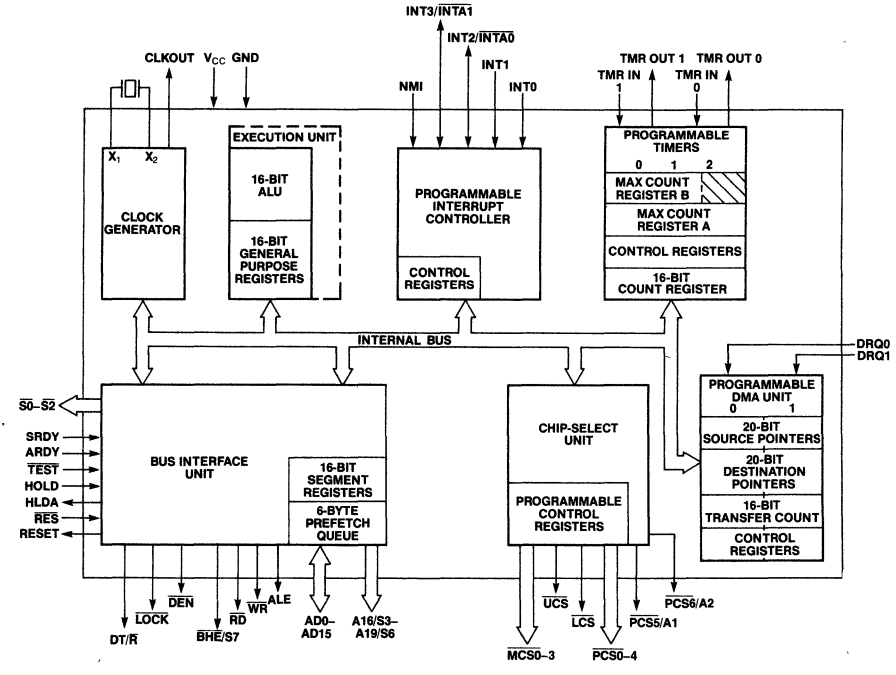


Figure 1. iAPX 186 Block Diagram

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.

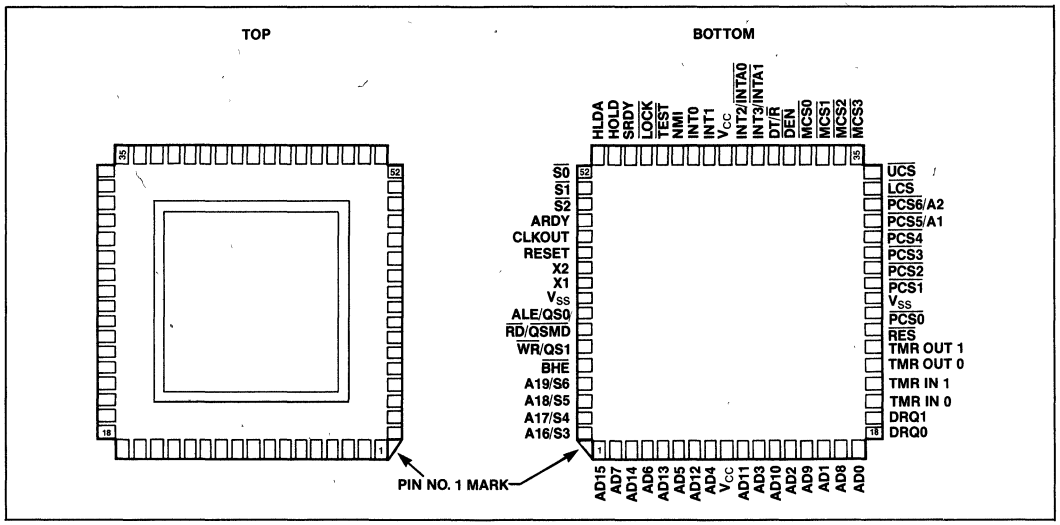


Figure 2. 80186 Pinout Diagram

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 80186 will drive the status lines to an inactive level for one clock, and then tri-state them.

Table 1. 80186 Pin Description (Continued)

<table>
<tr><th>Symbol</th><th>Pin No.</th><th>Type</th><th>Name and Function</th></tr>
<tr><td>$\overline{TEST}$</td><td>47</td><td>I</td><td>$\overline{TEST}$ 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.</td></tr>
<tr><td>TMR IN 0,
TMR IN 1</td><td>20
21</td><td>I
I</td><td>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</td></tr>
<tr><td>TMR OUT 0,
TMR OUT 1</td><td>22
23</td><td>O
O</td><td>Timer outputs are used to provide single pulse or continuous waveform generation, depending upon the timer mode selected.</td></tr>
<tr><td>DRQ0
DRQ1</td><td>18
19</td><td>I
I</td><td>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.</td></tr>
<tr><td>NMI</td><td>46</td><td>I</td><td>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.</td></tr>
<tr><td>INT0, INT1,
INT2/$\overline{INTA0}$
INT3/$\overline{INTA1}$</td><td>45,44
42
41</td><td>I
I/O
I/O</td><td>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).</td></tr>
<tr><td>A19/S6,
A18/S5,
A17/S4,
A16/S3</td><td>65
66
67
68</td><td>O
O
O
O</td><td>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:
<table>
<tr><th></th><th>Low</th><th>High</th></tr>
<tr><td>S6</td><td>Processor Cycle</td><td>DMA Cycle</td></tr>
</table>
S3,S4, and S5 are defined as LOW during T_2–T_4.</td></tr>
<tr><td>AD15–AD0</td><td>10–17,
1–8</td><td>I/O</td><td>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.</td></tr>
<tr><td>$\overline{BHE}$/S7</td><td>64</td><td>O</td><td>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.
<table>
<tr><th colspan="3">$\overline{BHE}$ and A0 Encodings</th></tr>
<tr><th>$\overline{BHE}$ Value</th><th>A0 Value</th><th>Function</th></tr>
<tr><td>0</td><td>0</td><td>Word Transfer</td></tr>
<tr><td>0</td><td>1</td><td>Byte Transfer on upper half of data bus (D15–D8)</td></tr>
<tr><td>1</td><td>0</td><td>Byte Transfer on lower half of data bus (D_7–D_0)</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table></td></tr>
</table>

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 holds 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. If unused, this line should be tied LOW.
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.

210451-004

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 may be 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.

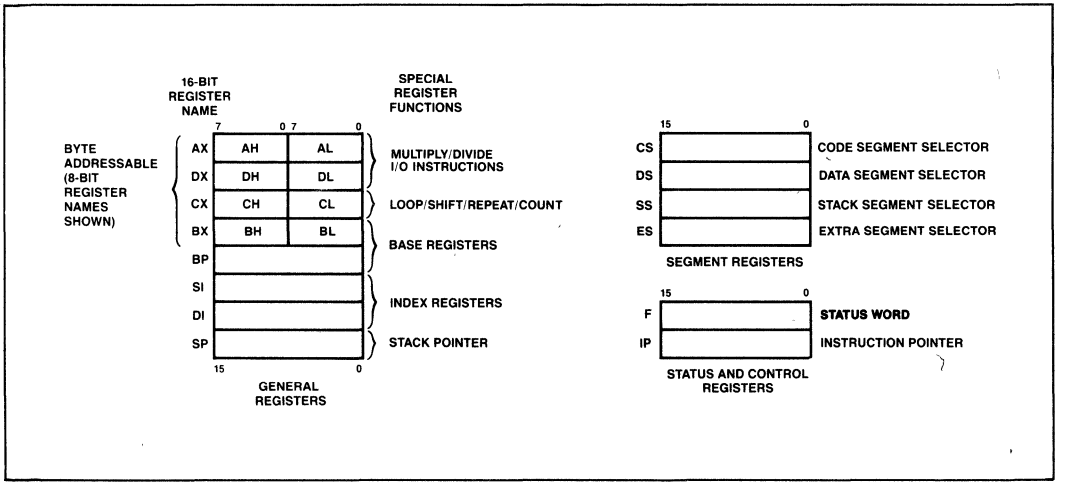


Figure 3a. 80186 General Purpose Register Set

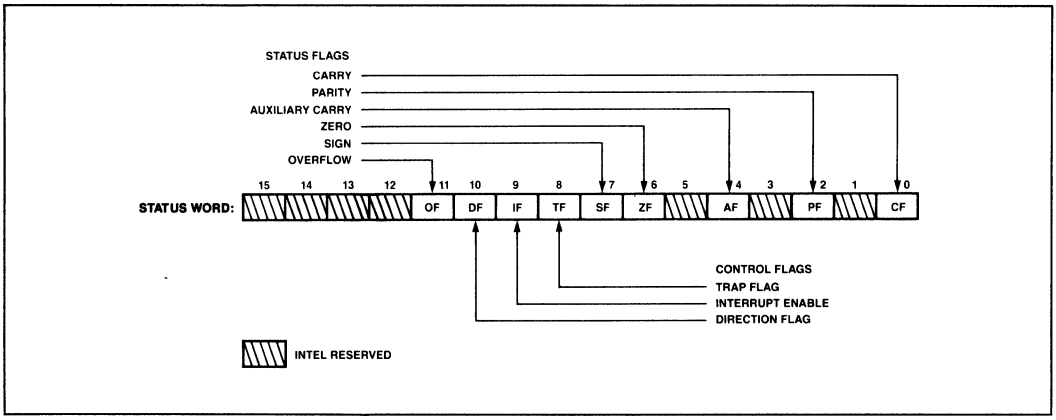


Figure 3b. Status Word Format

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

210451-004

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.

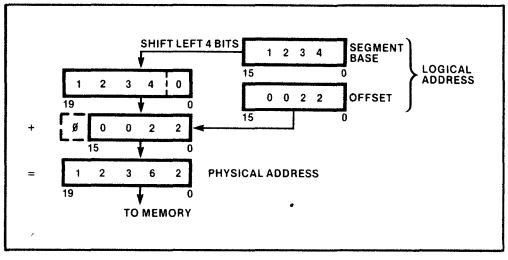


Figure 5. Two Component Address

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.

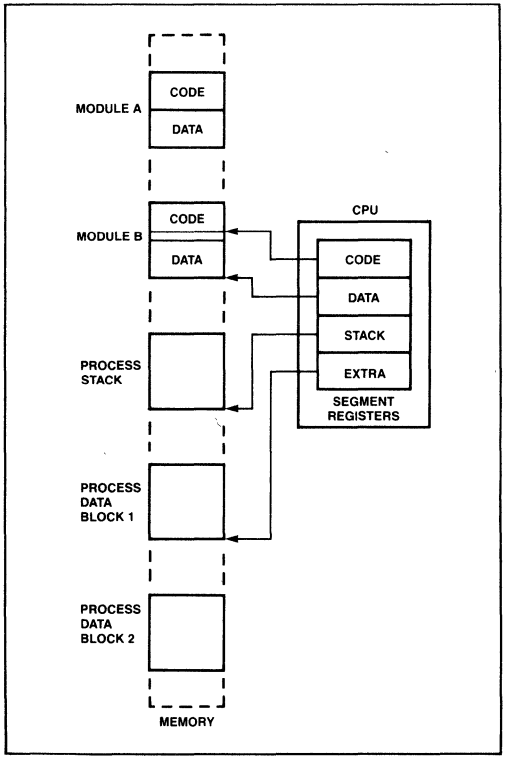


Figure 6. Segmented Memory Helps Structure Software

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.)

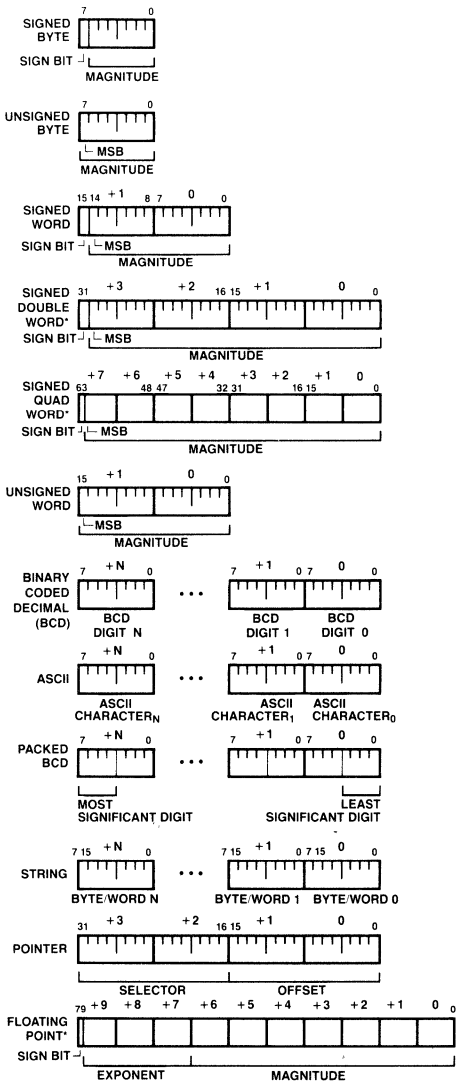
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the iAPX 186.

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.

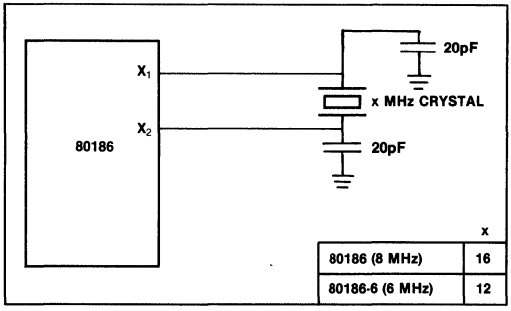


Figure 8. Recommended iAPX 186 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70° C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

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.

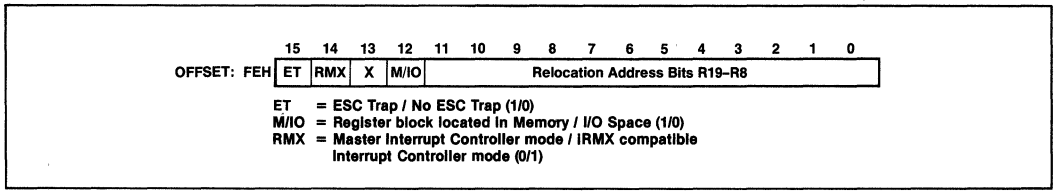


Figure 9. Relocation Register

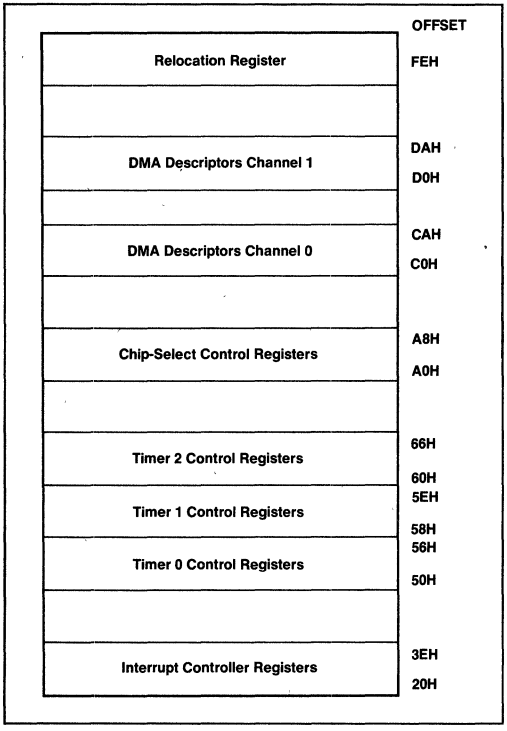


Figure 10. Internal Register Map

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 functionally as described in 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.

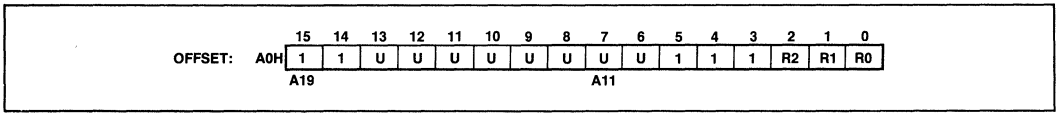


Figure 11. UMCS Register

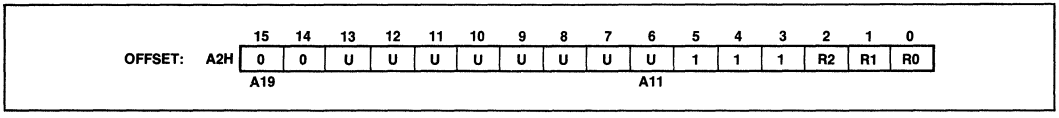


Figure 12. LMCS Register

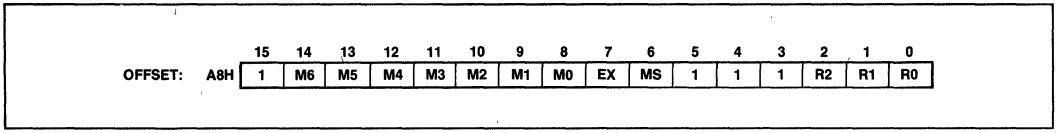


Figure 13. MPCS Register

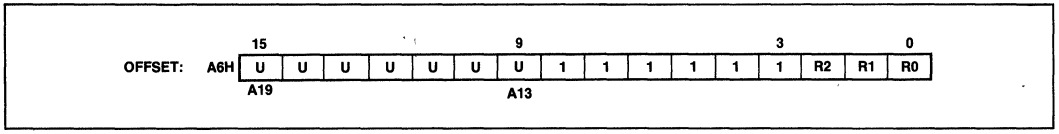


Figure 14. MMCS Register

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.

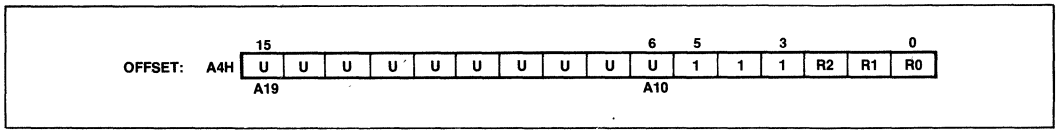


Figure 15. PACS Register

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

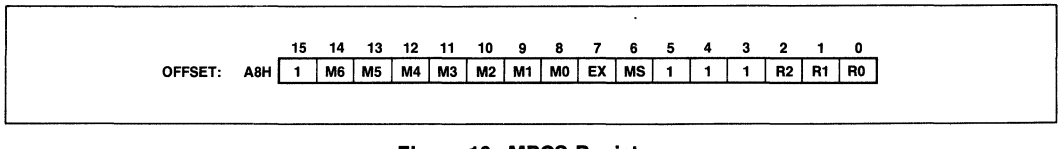


Figure 16. MPCS Register

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

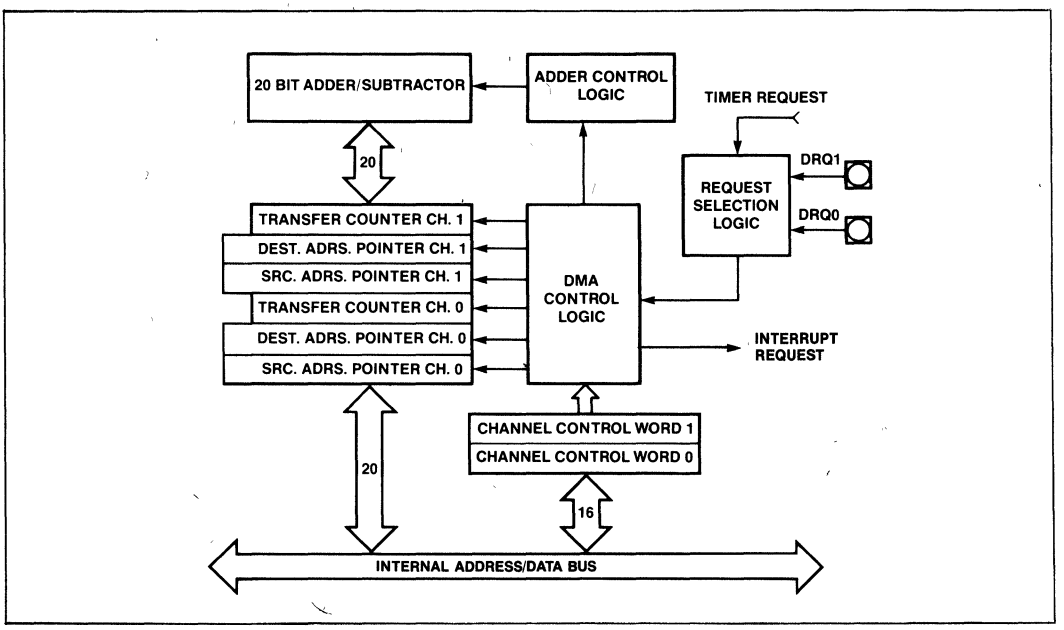


Figure 17. DMA Unit Block Diagram

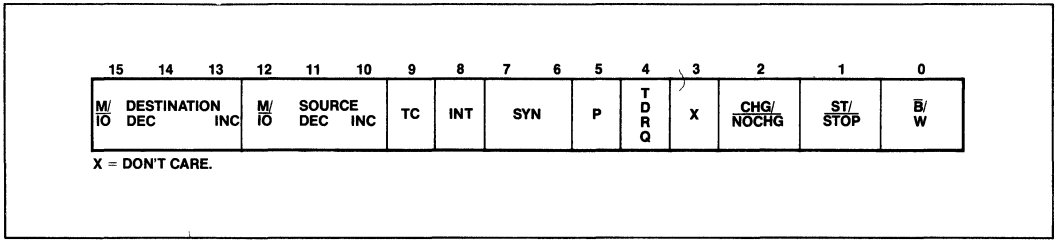


Figure 18. DMA Control Register

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

B/W: Byte/Word (0/1) Transfers.

ST/STOP: Start/stop (1/0) Channel.

CHG/NOCHG: Change/Do not change (1/0) ST/STOP bit. If this bit is set when writing to the control word, the ST/STOP bit will be programmed by the write to the control word. If this bit is cleared when writing the control word, the ST/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/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 B/W) after each transfer.

M/IO Source pointer is in M/IO space (1/0).

DEC Decrement source pointer by 1 or 2 (depends on B/W) after each transfer.

DEST: INC Increment destination pointer by 1 or 2 (B/W) after each transfer.

M/IO Destination pointer is in M/IO space (1/0).

DEC Decrement destination pointer by 1 or 2 (depending on 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

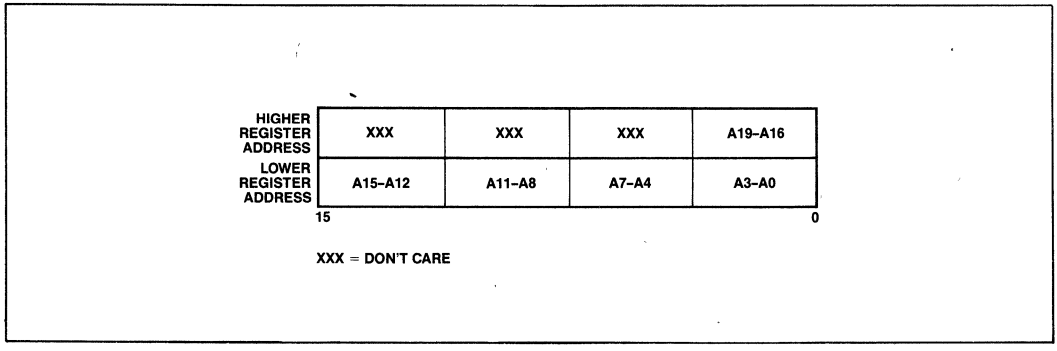


Figure 18a. DMA Memory Pointer Register Format

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.

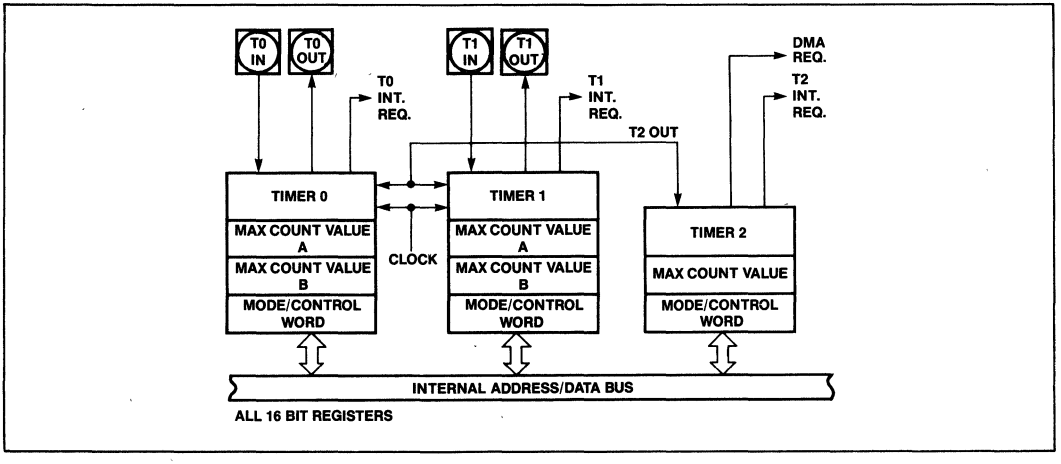


Figure 19. Timer Block Diagram

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, 1 clock after the maximum count value has been reached. In the dual MAX COUNT register mode, the ouput 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

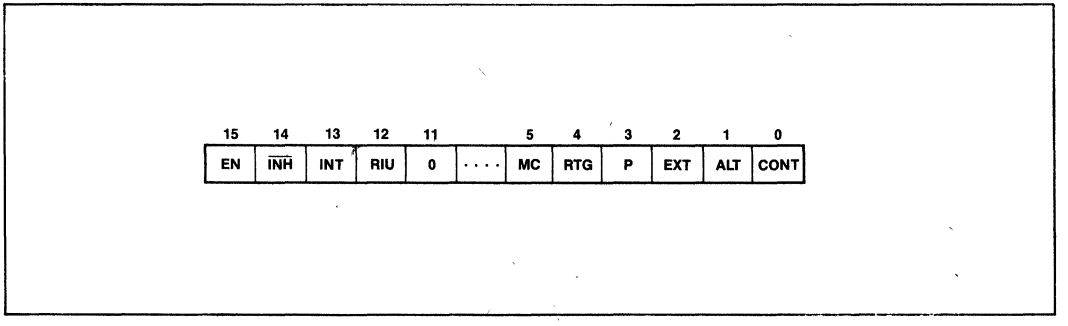


Figure 20. Timer Mode/Control Register

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.

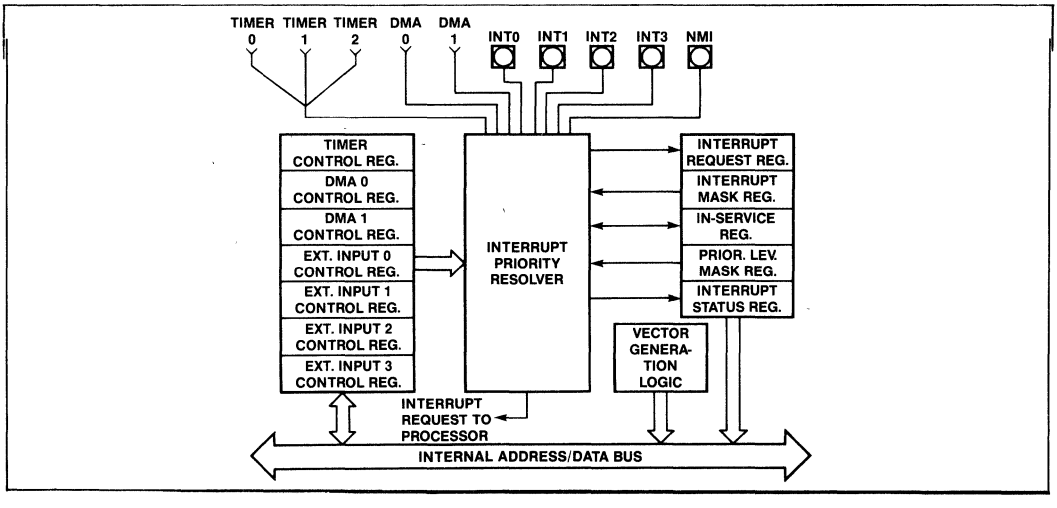


Figure 21. Interrupt Controller Block Diagram

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.

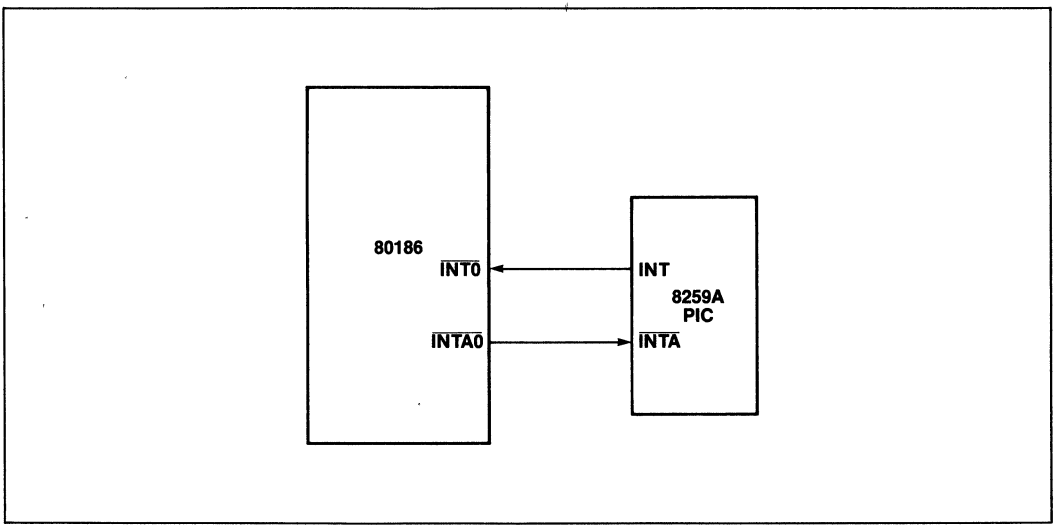


Figure 22. Cascade Mode Interrupt Connection

	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.

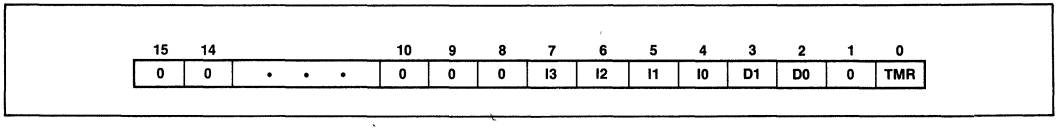


Figure 24. In-Service, Interrupt Request, and Mask Register Formats

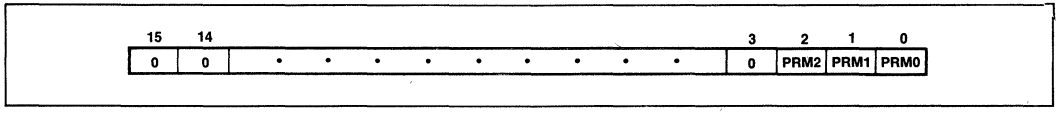


Figure 25. Priority Mask Register Format

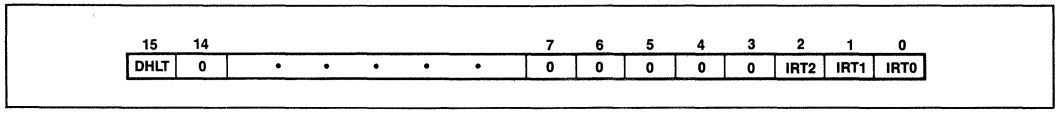


Figure 26. Interrupt Status Register Format

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 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.

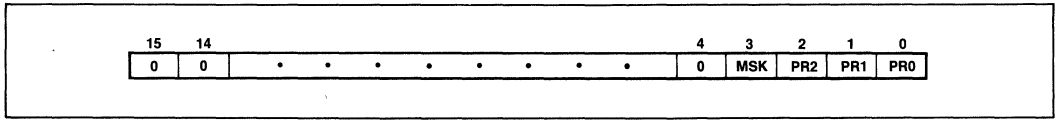


Figure 27. Timer/DMA Control Register Formats

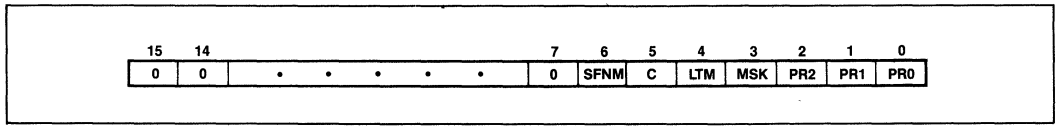


Figure 28. INT0/INT1 Control Register Formats

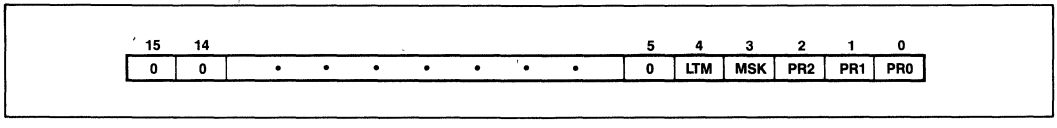


Figure 29. INT2/INT3 Control Register Formats

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.

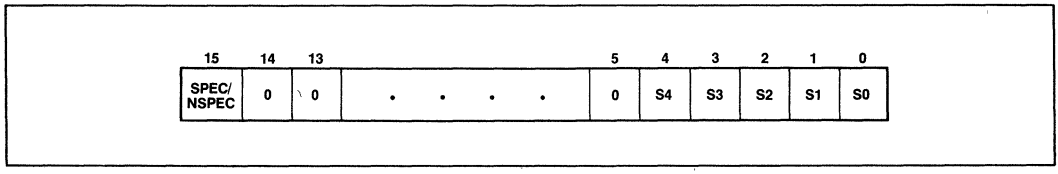


Figure 30. EOI Register Format

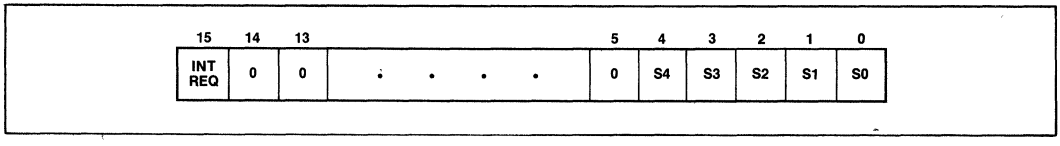


Figure 31. Poll Register Format

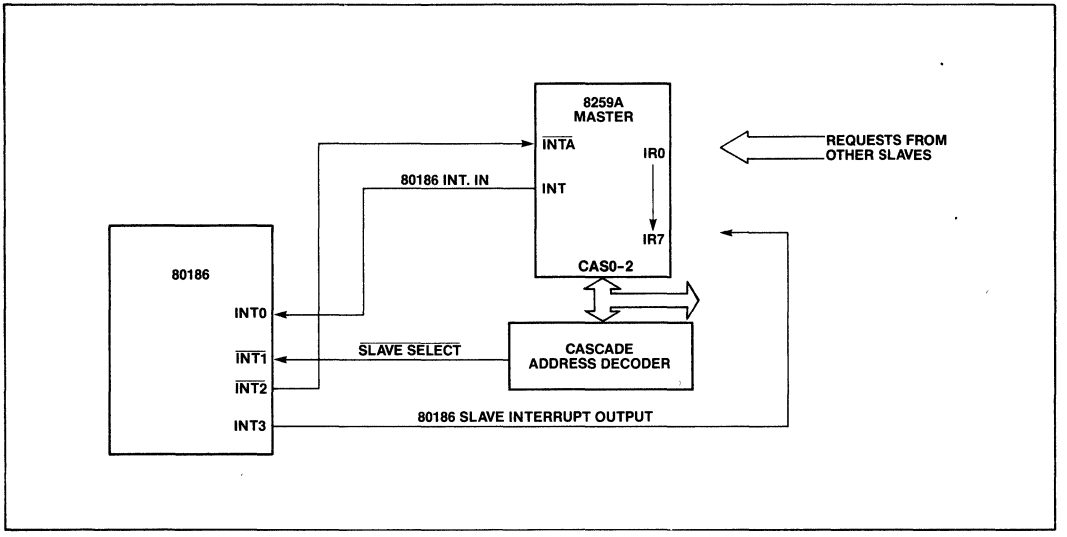


Figure 32. iRMX 86 Interrupt Controller Interconnection

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.

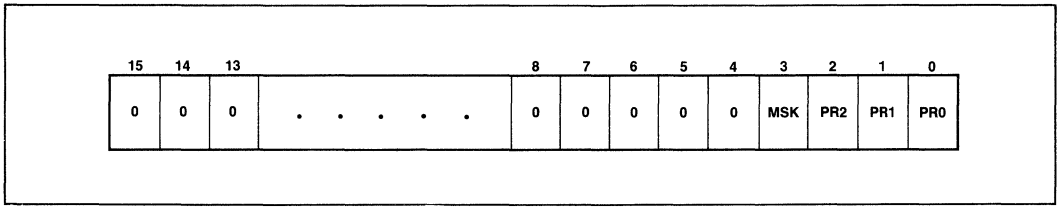


Figure 36. Control Word Format

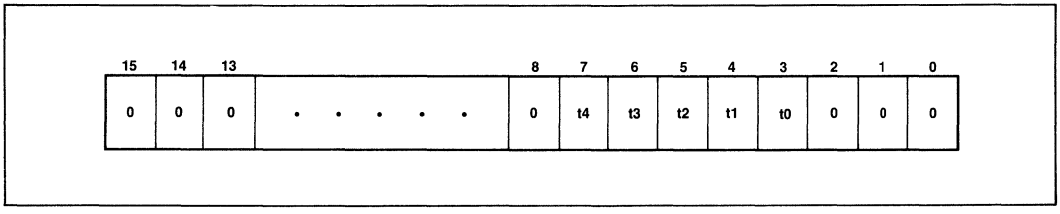


Figure 37. Interrupt Vector Register Format

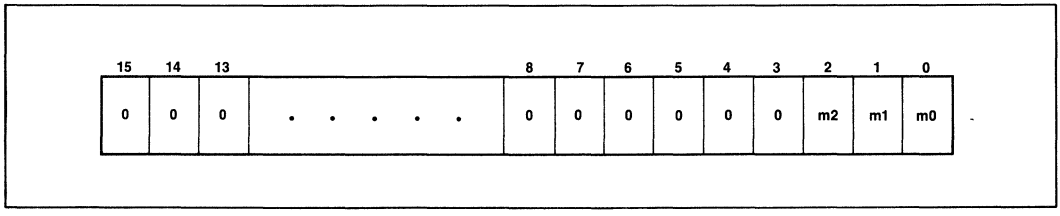


Figure 38. Priority Level Mask Register

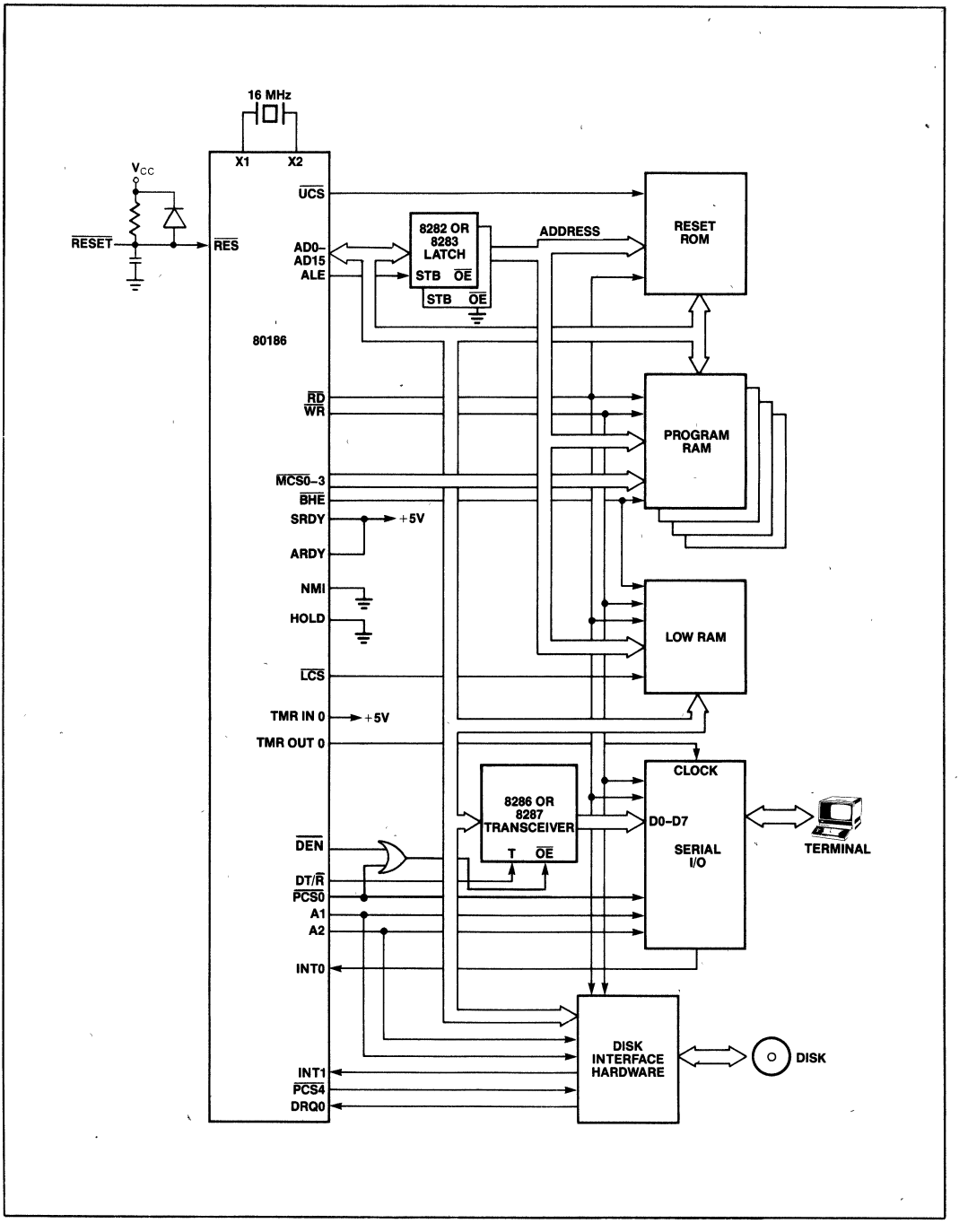


Figure 39. Typical iAPX 186 Computer

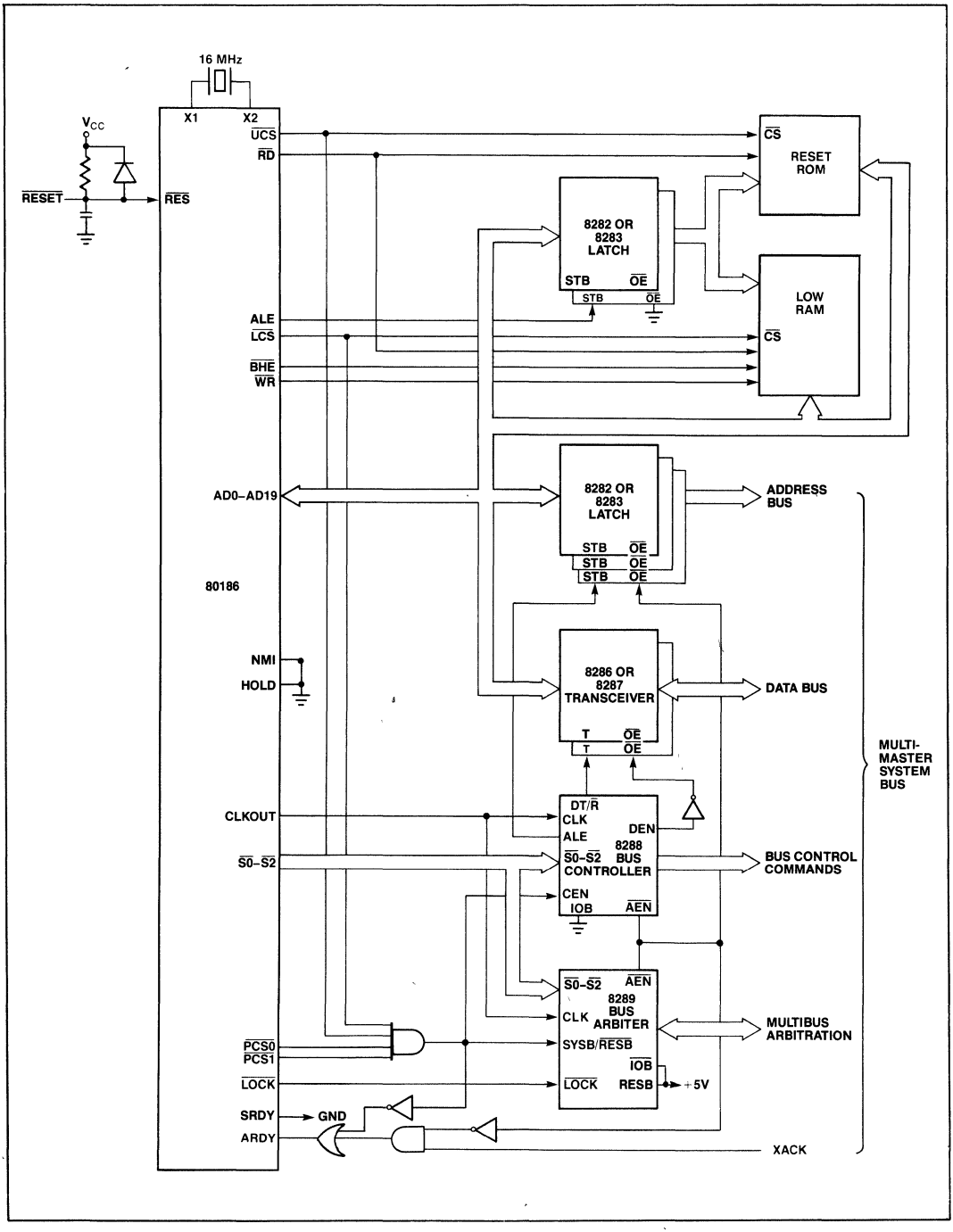


Figure 40. Typical iAPX 186 Multi-Master Bus Interface

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.

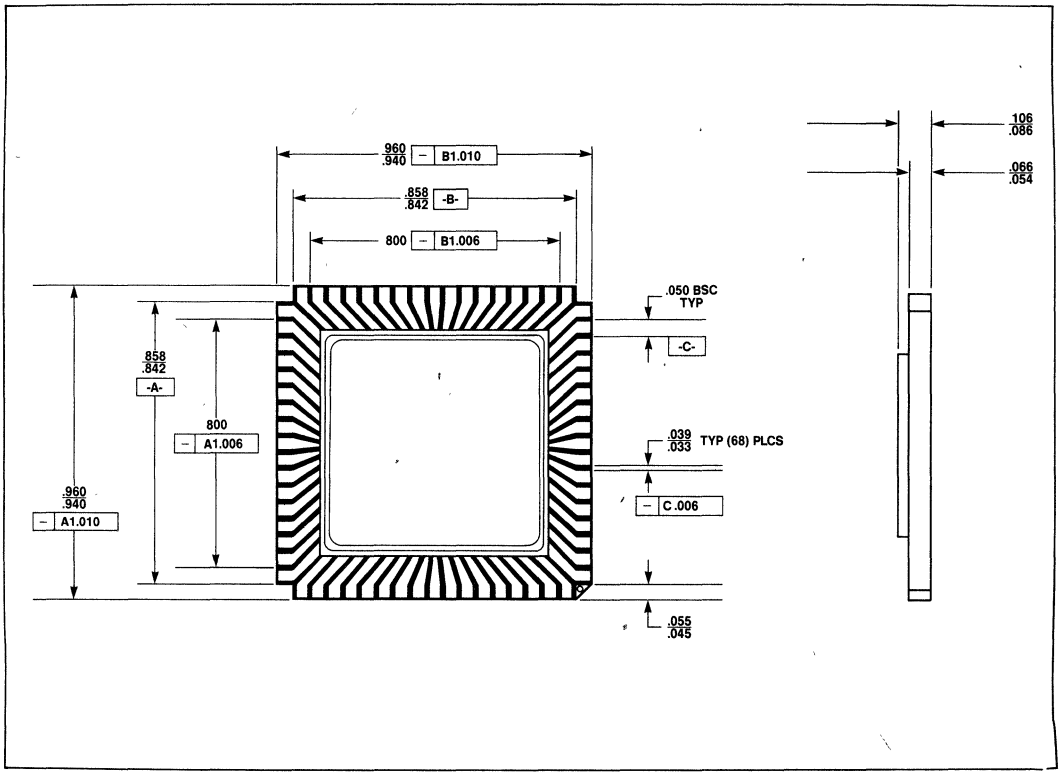
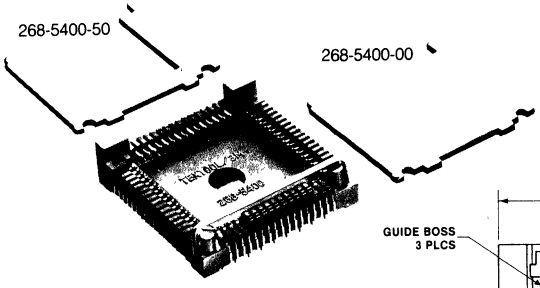
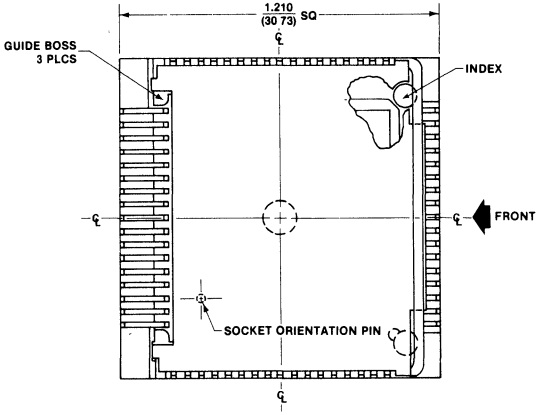
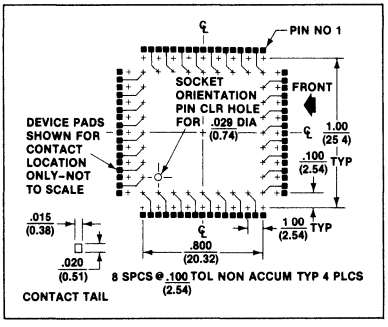


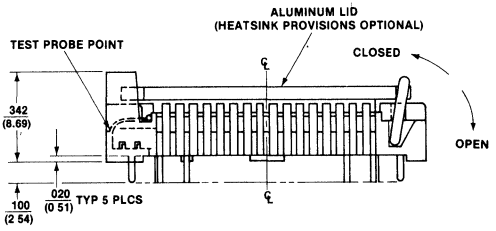
Figure 41. 80186 JEDEC Type A Package





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 ($\overline{RES}$)	3.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		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		$\frac{550}{450}$	mA	Max measured at $\frac{T_A = 0°C}{T_A = 70°C}$
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%)
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
T_{DVCL}	Data in Setup (A/D)	20		ns	
T_{CLDX}	Data in Hold (A/D)	10		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		ns	
T_{ARYLCL}	AREADY inactive setup time	35		ns	
T_{CHARYX}	AREADY hold time	15		ns	
T_{ARYCHL}	Asynchronous Ready inactive hold time	15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		ns	
T_{CLSRY}	SREADY transition hold time	15		ns	
T_{HVCL}	HOLD Setup*	25		ns	

*To guarantee recognition at next clock.

80186 Timing Requirements (Continued)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		ns	

*To guarantee recognition at next clock.

80186 Master Interface Timing Responses

		80186 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CLAV}	Address Valid Delay	5	55	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	10	70	10	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)	10	70	10	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	5	50	5	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	65	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	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to clock high	10		10		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

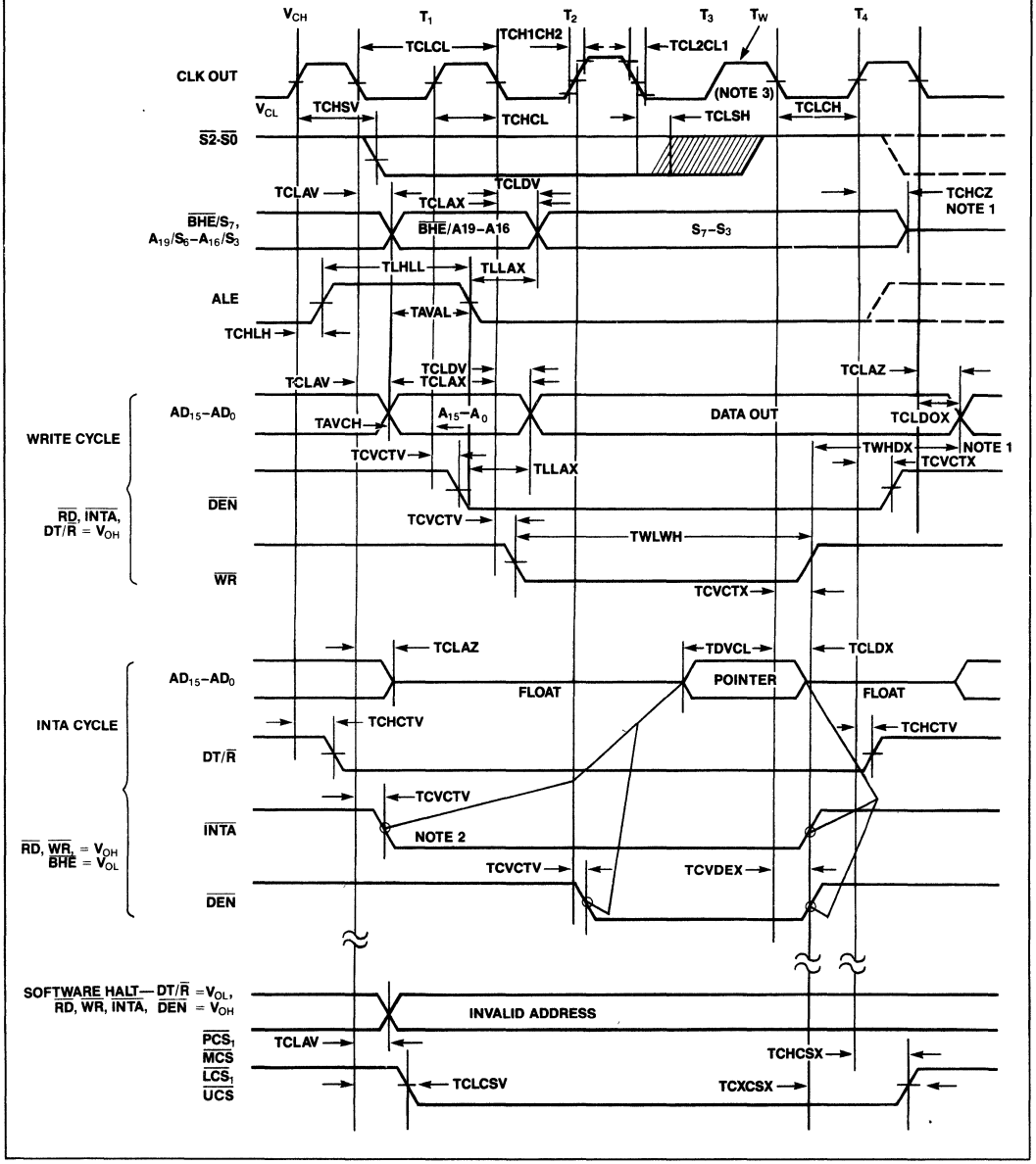
		80186 (8 MHz)		80186-6 (6 MHz)			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CKIN}	CLKIN Period	62.5	250	83	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0 volts
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5 volts
T_{CLCK}	CLKIN Low Time	25		33		ns	1.5 volts
T_{CHCK}	CLKIN High Time	25		33		ns	1.5 volts

80186 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

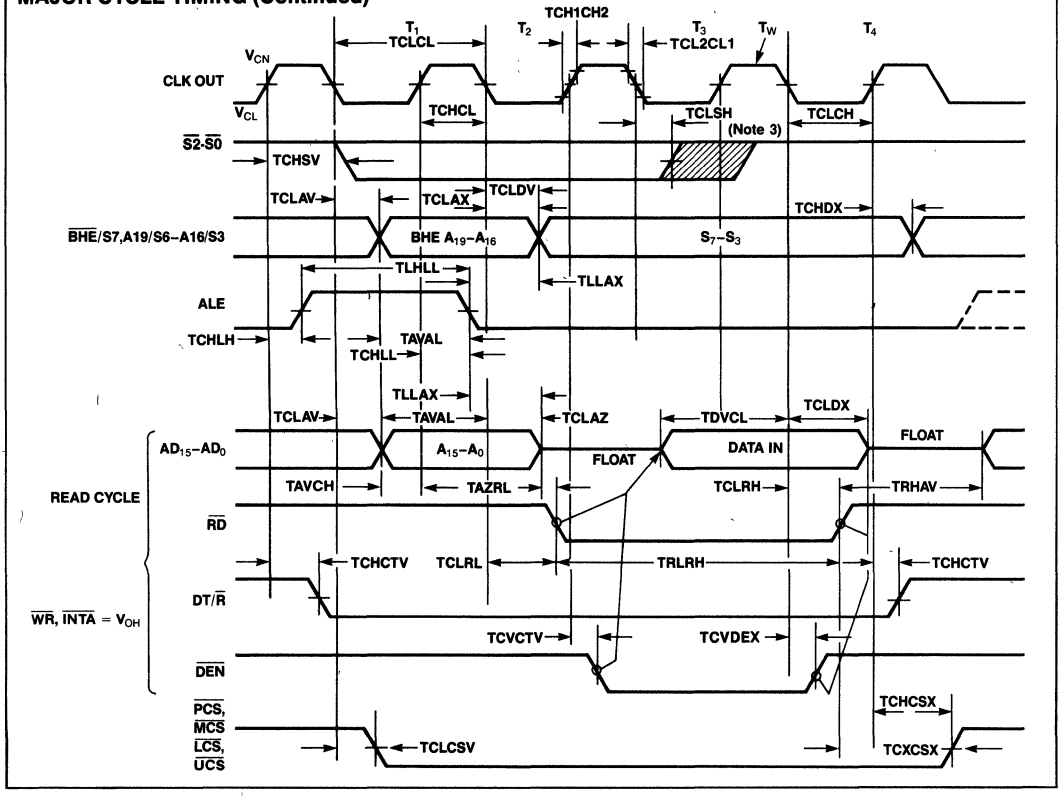
MAJOR CYCLE TIMING



210451-004

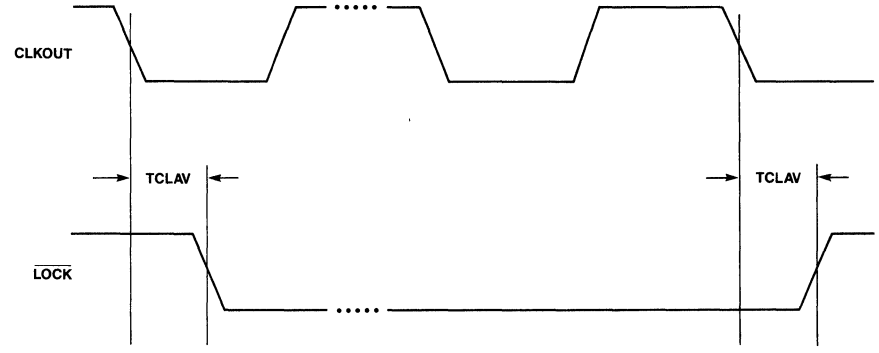
WAVEFORMS (Continued)

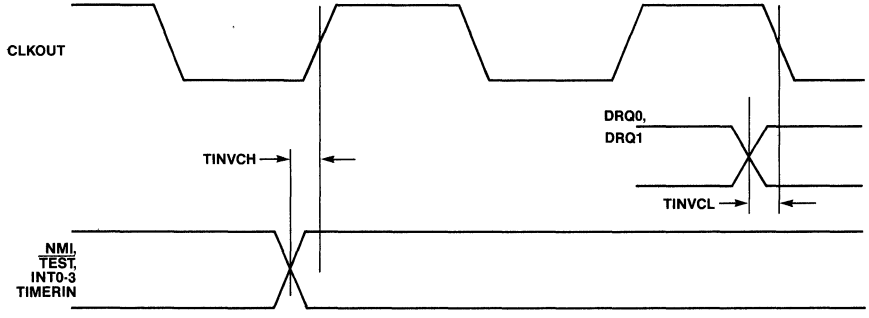
MAJOR CYCLE TIMING (Continued)

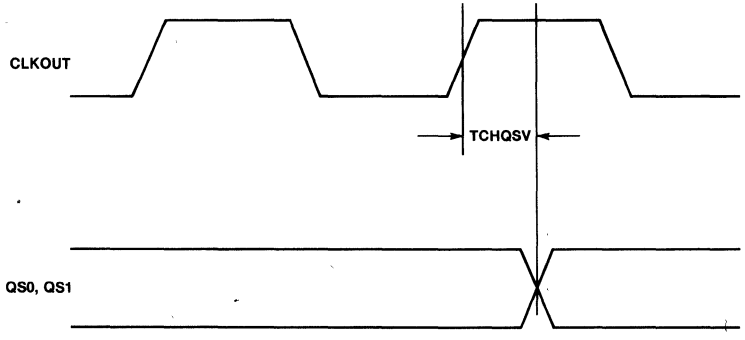


NOTES:
1. Following a Write cycle, the Local Bus is floated by the 80186 only when the 80186 enters a "Hold Acknowledge" state.
2. INTA occurs one clock later in RMX-mode.
3. Status inactive just prior to T_4

WAVEFORMS (Continued)

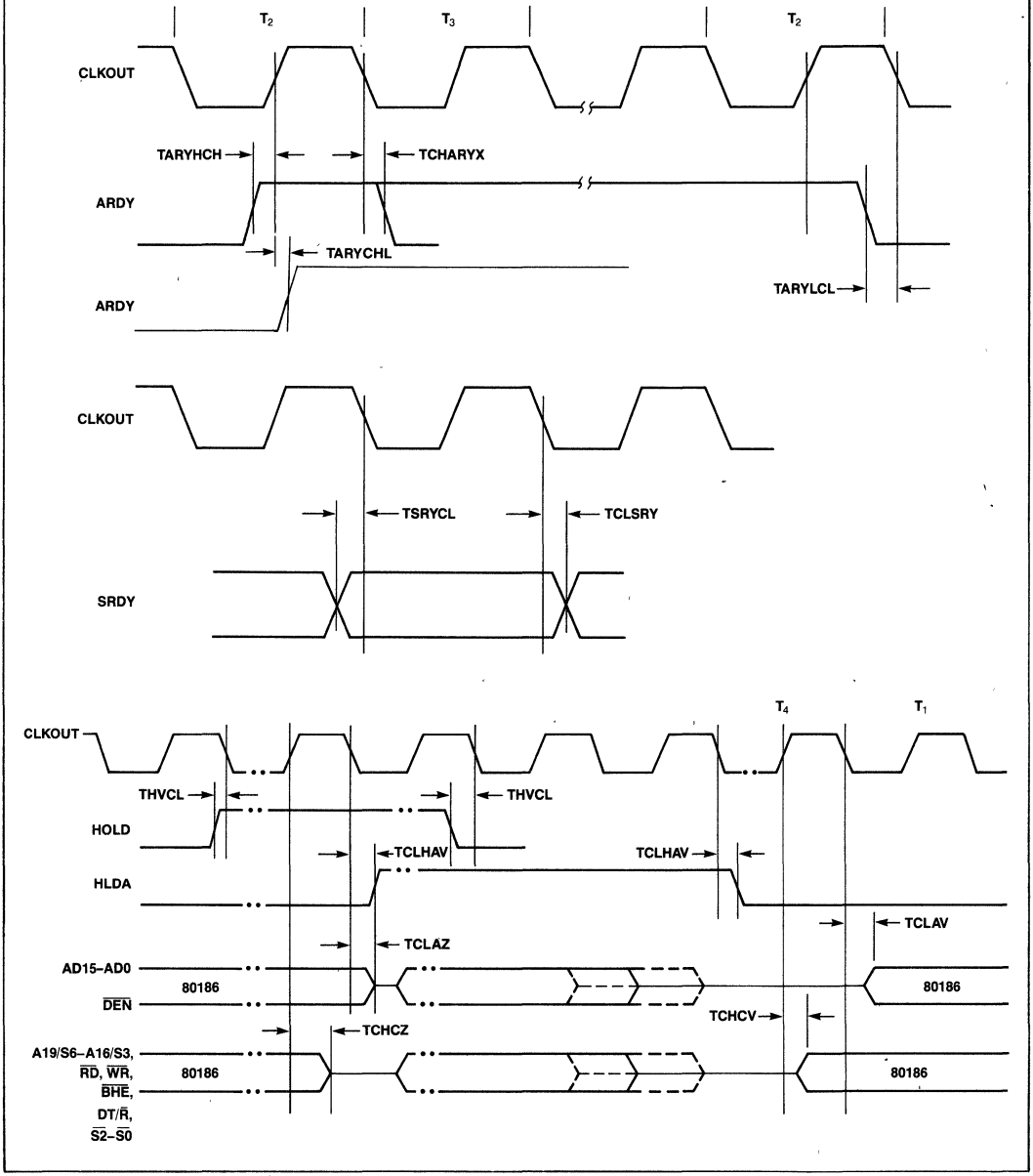






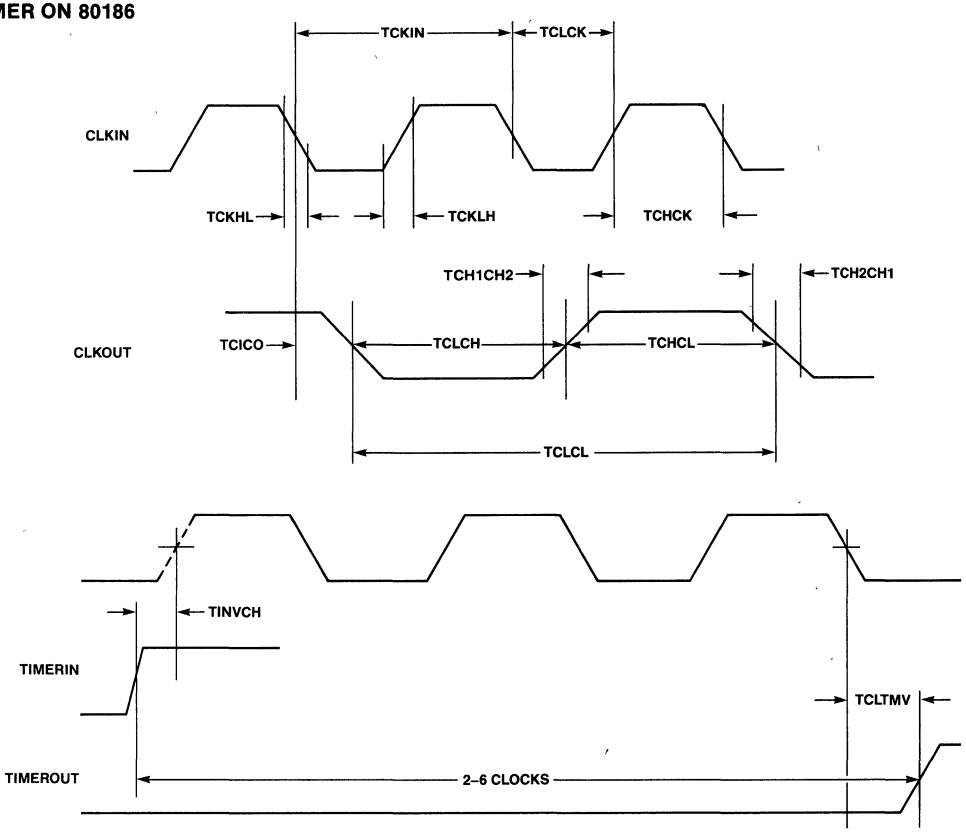
WAVEFORMS (Continued)

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	15	
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	14	
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	14	
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	14	
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 1 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.

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.

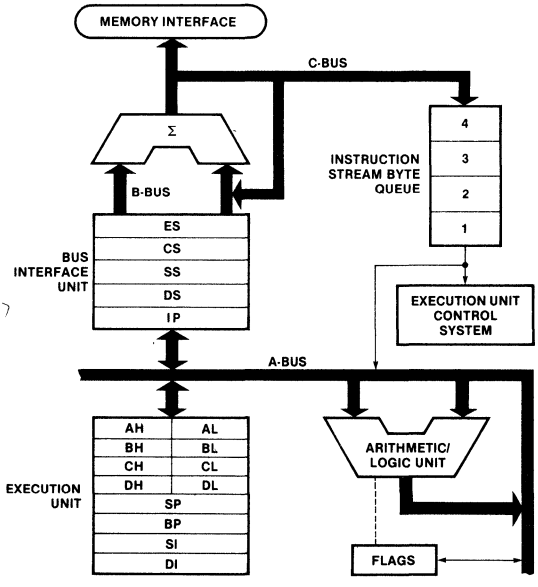


Figure 1. iAPX 88/10 CPU Functional Block Diagram

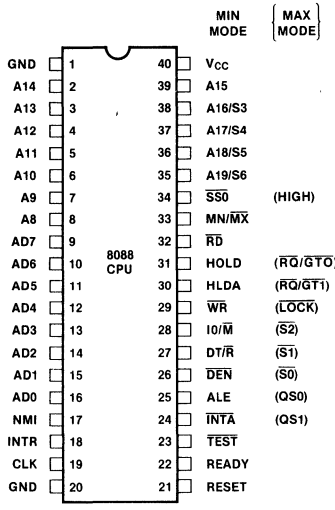


Figure 2. iAPX 88/10 Pin Configuration

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	35-38	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 $\overline{SO}$ 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.

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.

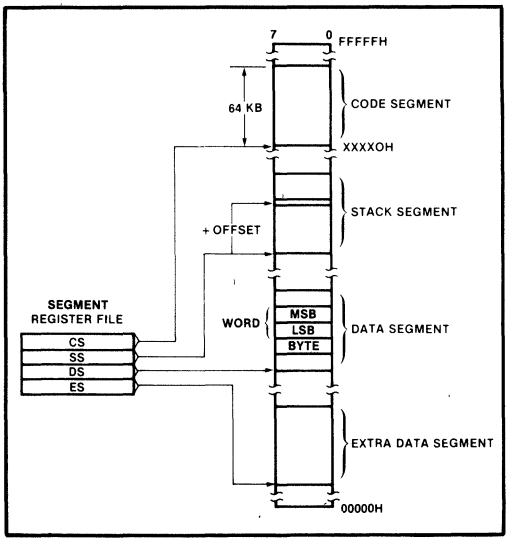


Figure 3. Memory Organization

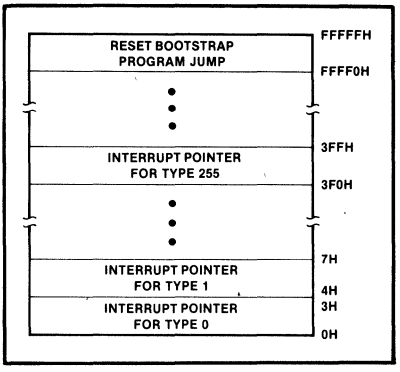


Figure 4. Reserved Memory Locations

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.

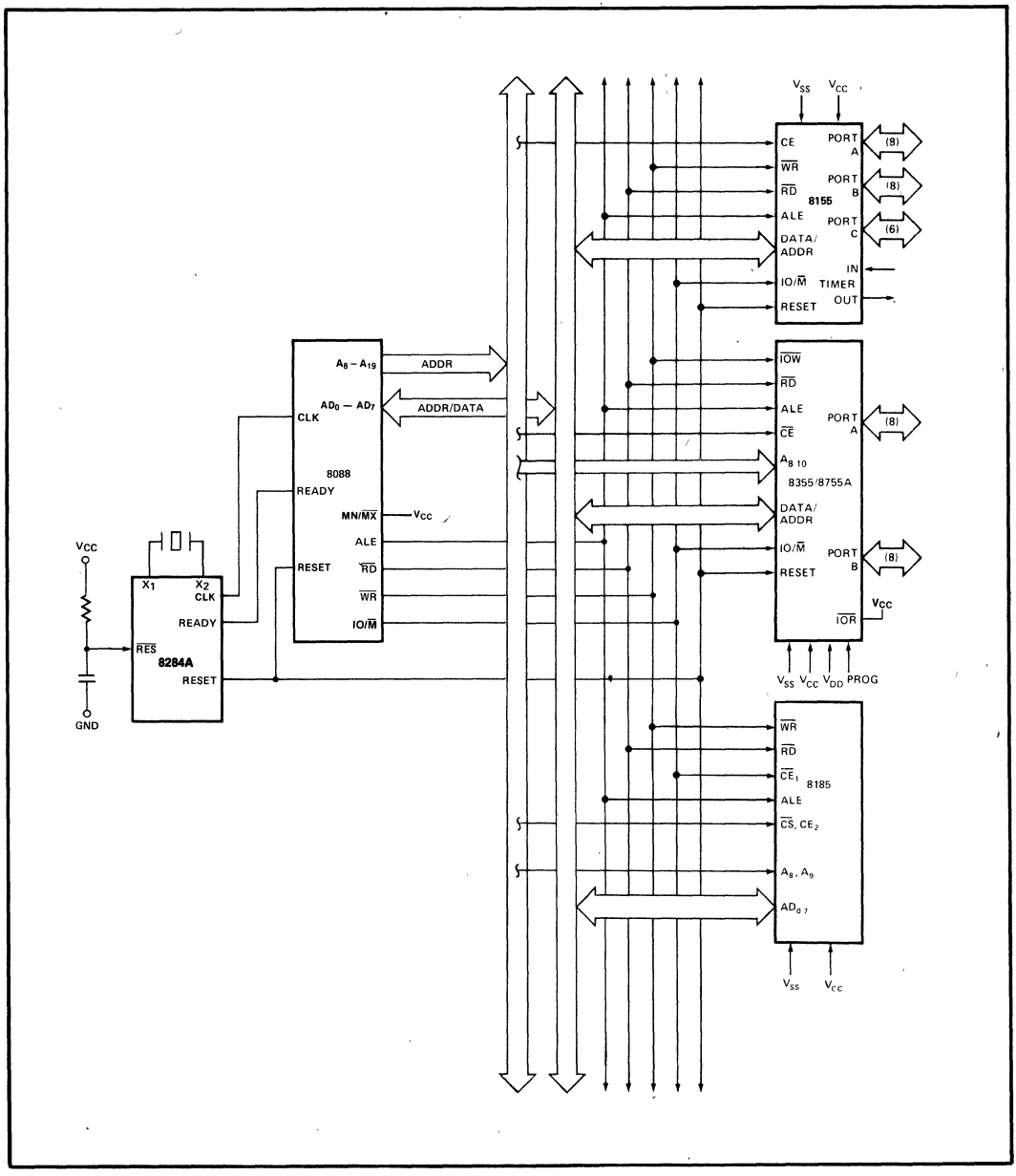


Figure 5. Multiplexed Bus Configuration

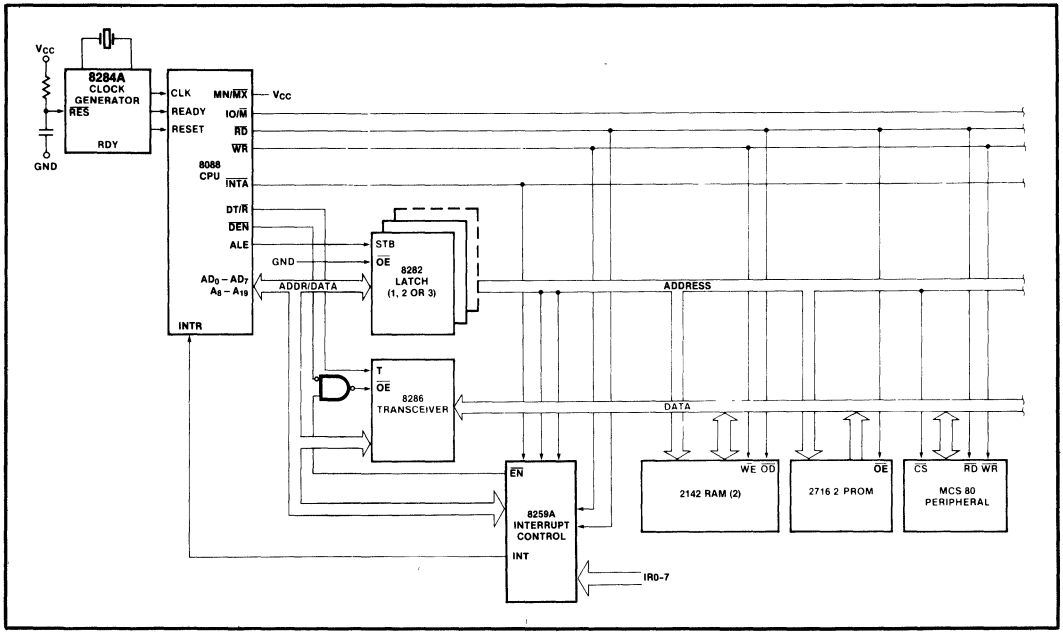


Figure 6. Demultiplexed Bus Configuration

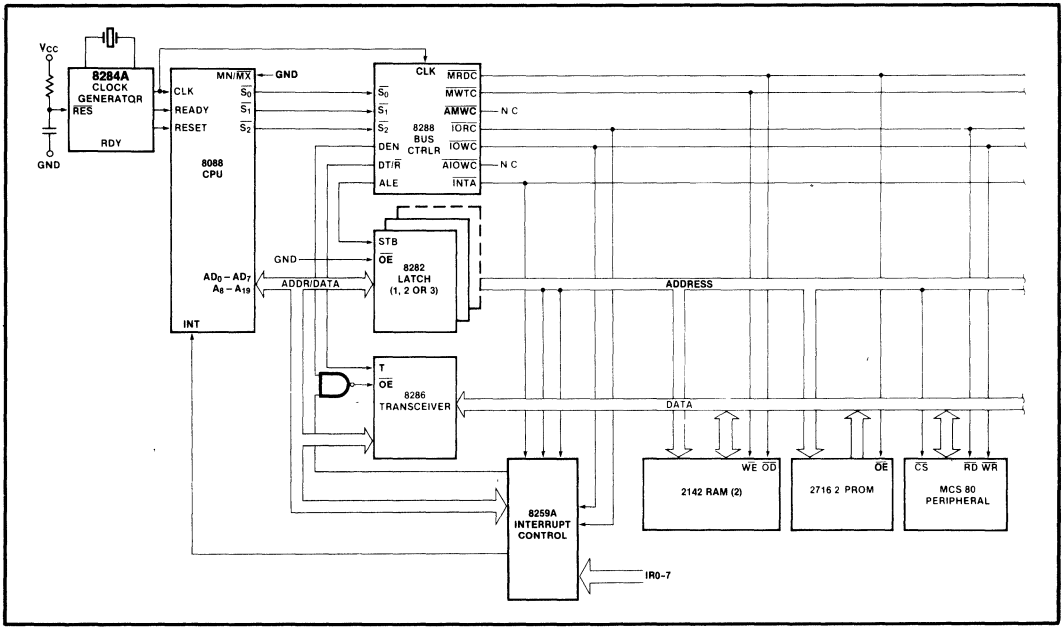


Figure 7. Fully Buffered System Using Bus Controller

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,**

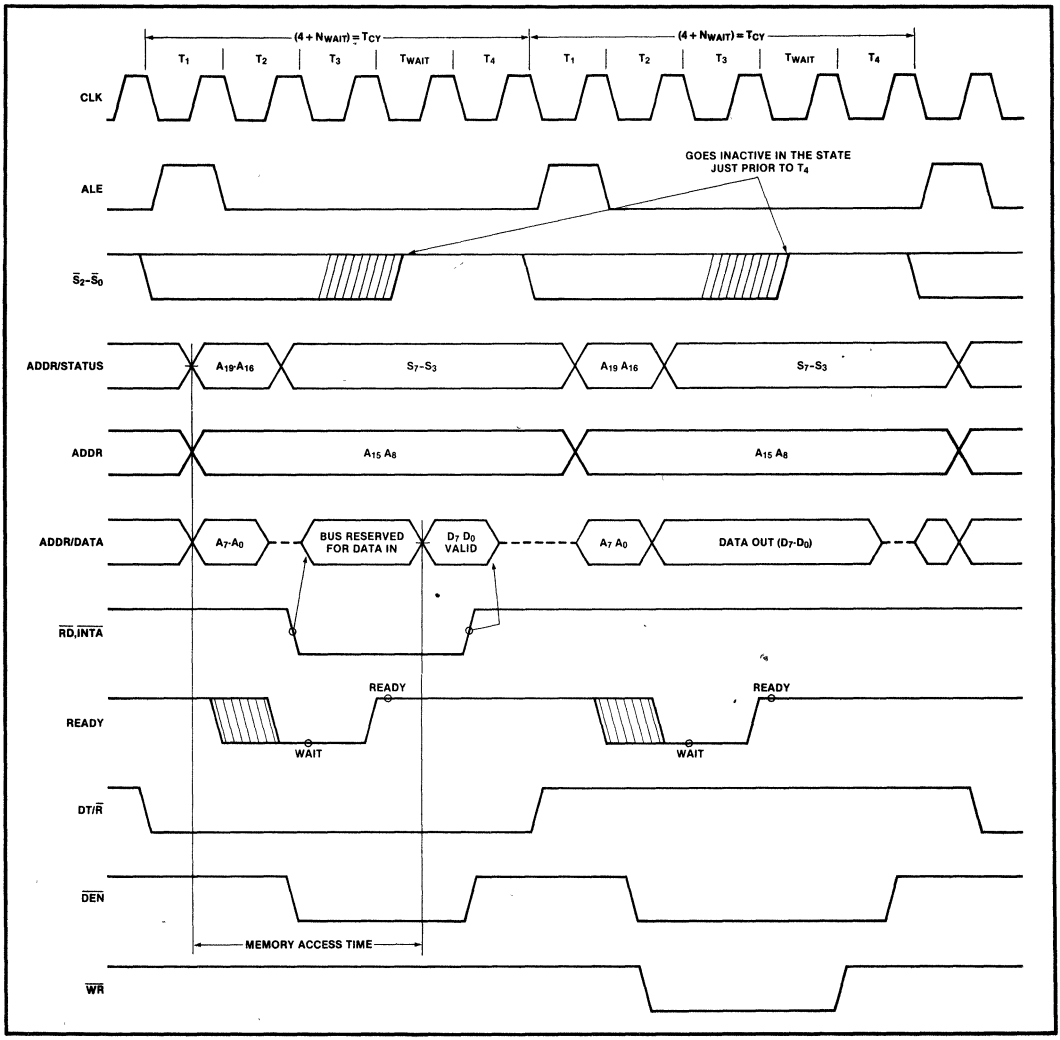


Figure 8. Basic System Timing

"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 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 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.

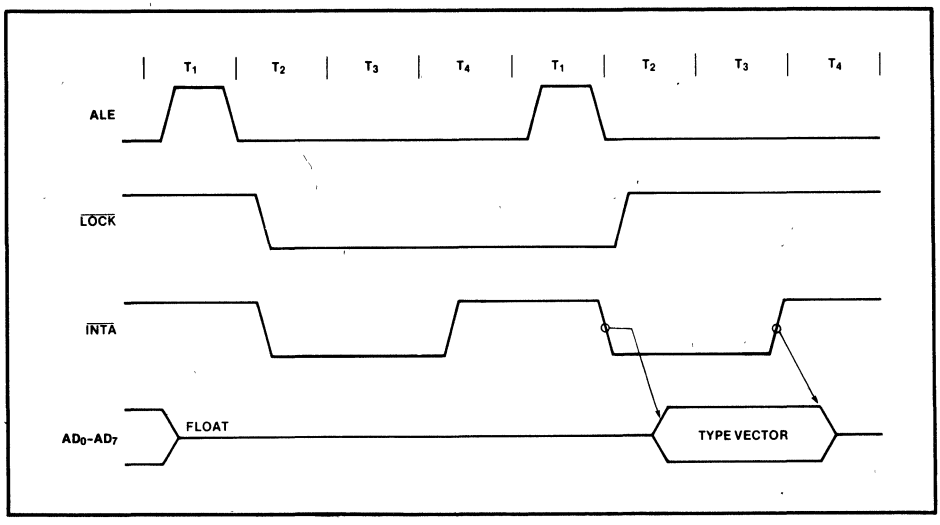


Figure 9. Interrupt Acknowledge Sequence

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 $\overline{SO}$ 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.

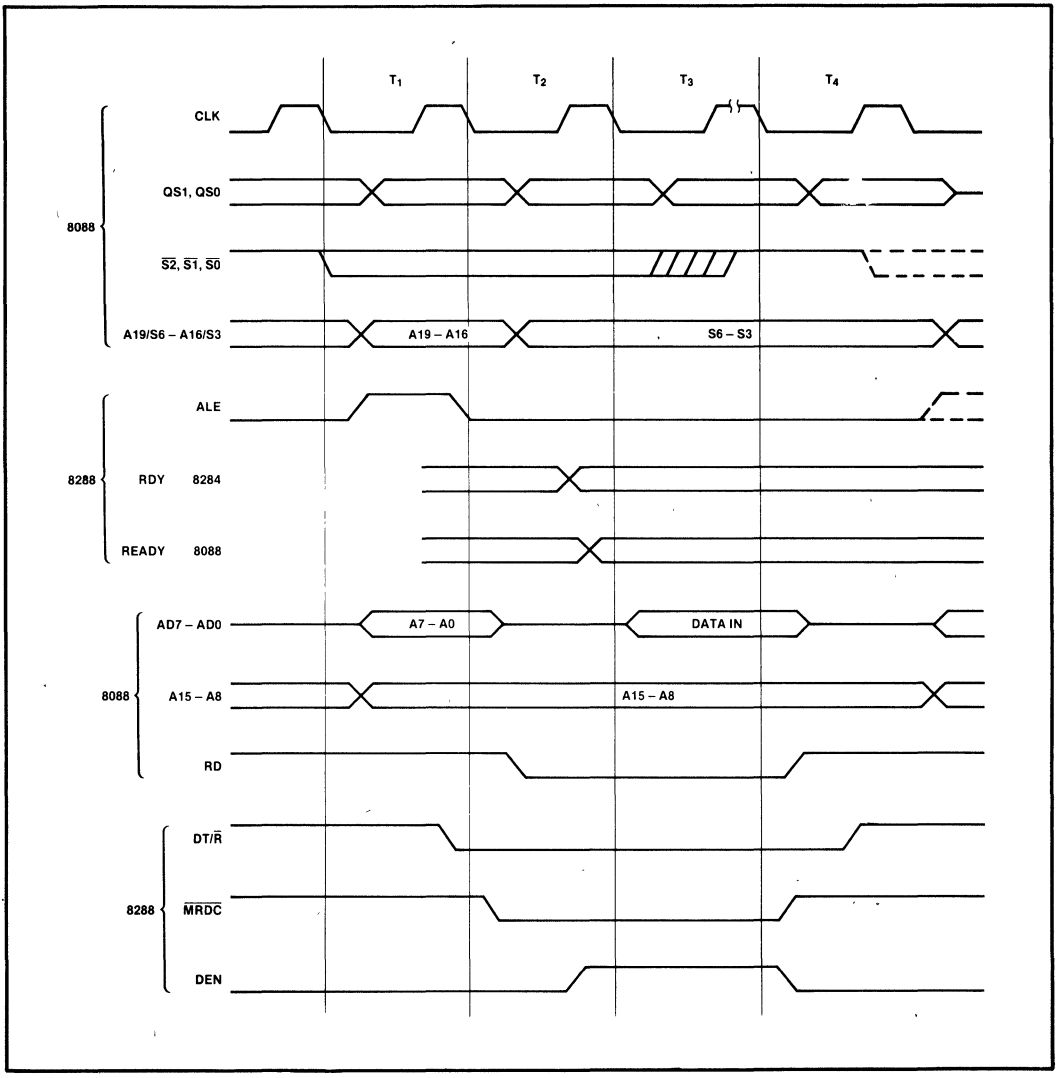


Figure 10. Medium Complexity System Timing

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	(See note 1)
V_{IH}	Input High Voltage	2.0	V_{CC}+0.5	V	(See note 1,2)
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%

Note 1: V_{IL} tested with MN/$\overline{MX}$ Pin = 0V
V_{IH} tested with MN/$\overline{MX}$ Pin = 5V
MN/$\overline{MX}$ Pin is a strap Pin
Note 2: Not applicable to $\overline{RQ}$/$\overline{GT0}$ and $\overline{RQ}$/$\overline{GT1}$ Pins (Pin 30 and 31)

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

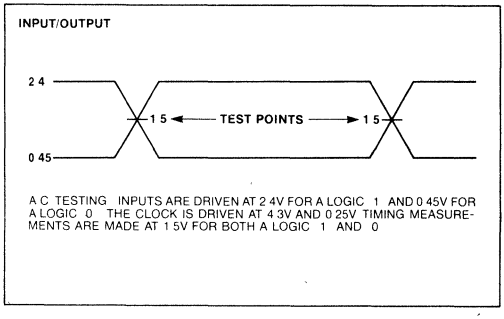
*Note: For Extended Temperature EXPRESS V_{CC} = 5V ± 5%

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES

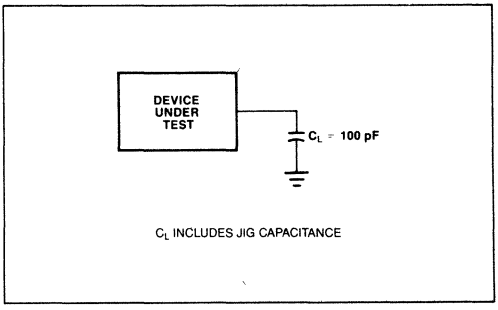
		8088		8088-2			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLAV	Address Valid Delay	10	110	10	60	ns	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
TCLAX	Address Hold Time	10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	ns	
TLHLL	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
TOHOL	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

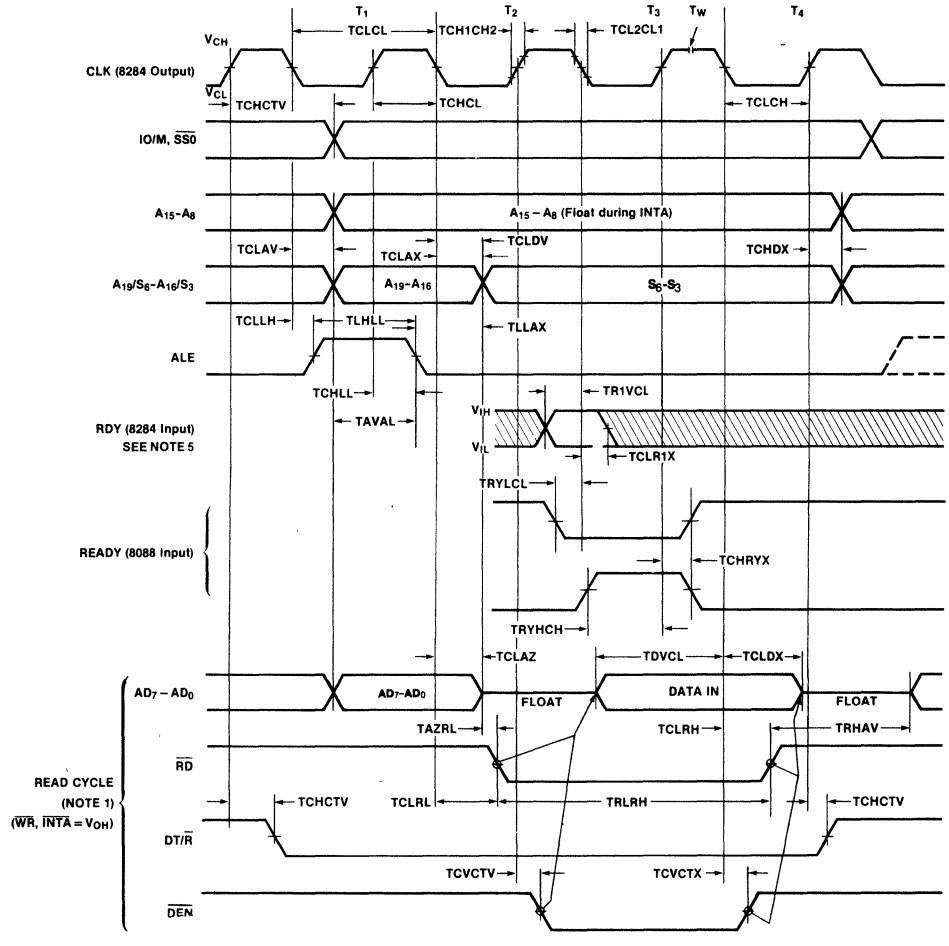
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

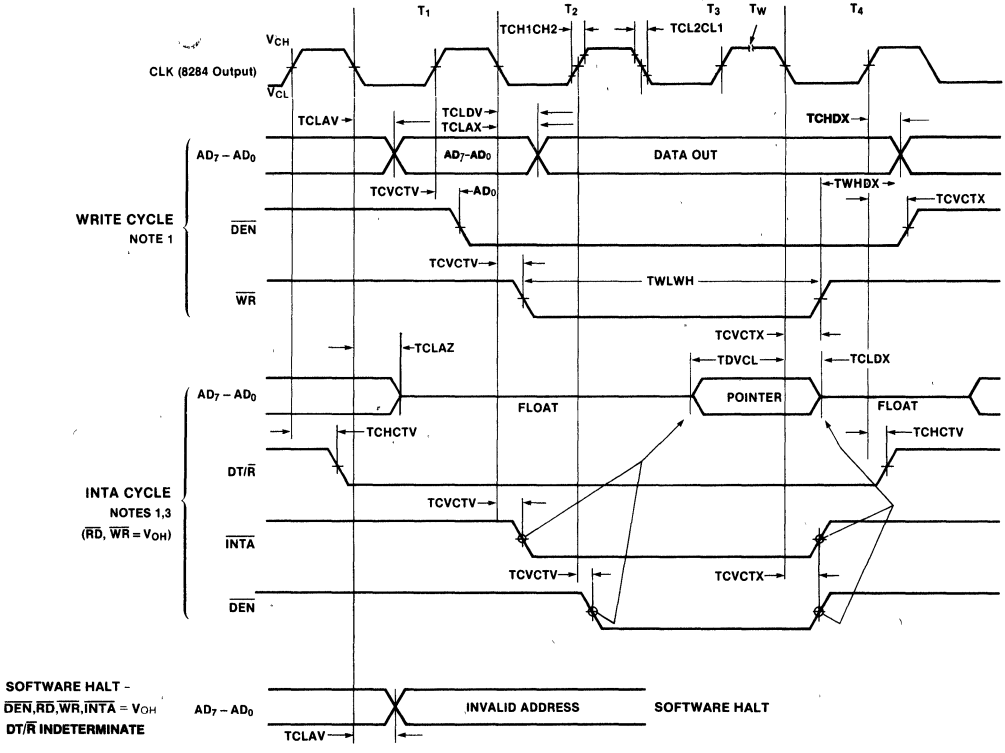
WAVEFORMS

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. ALLTIMING 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).

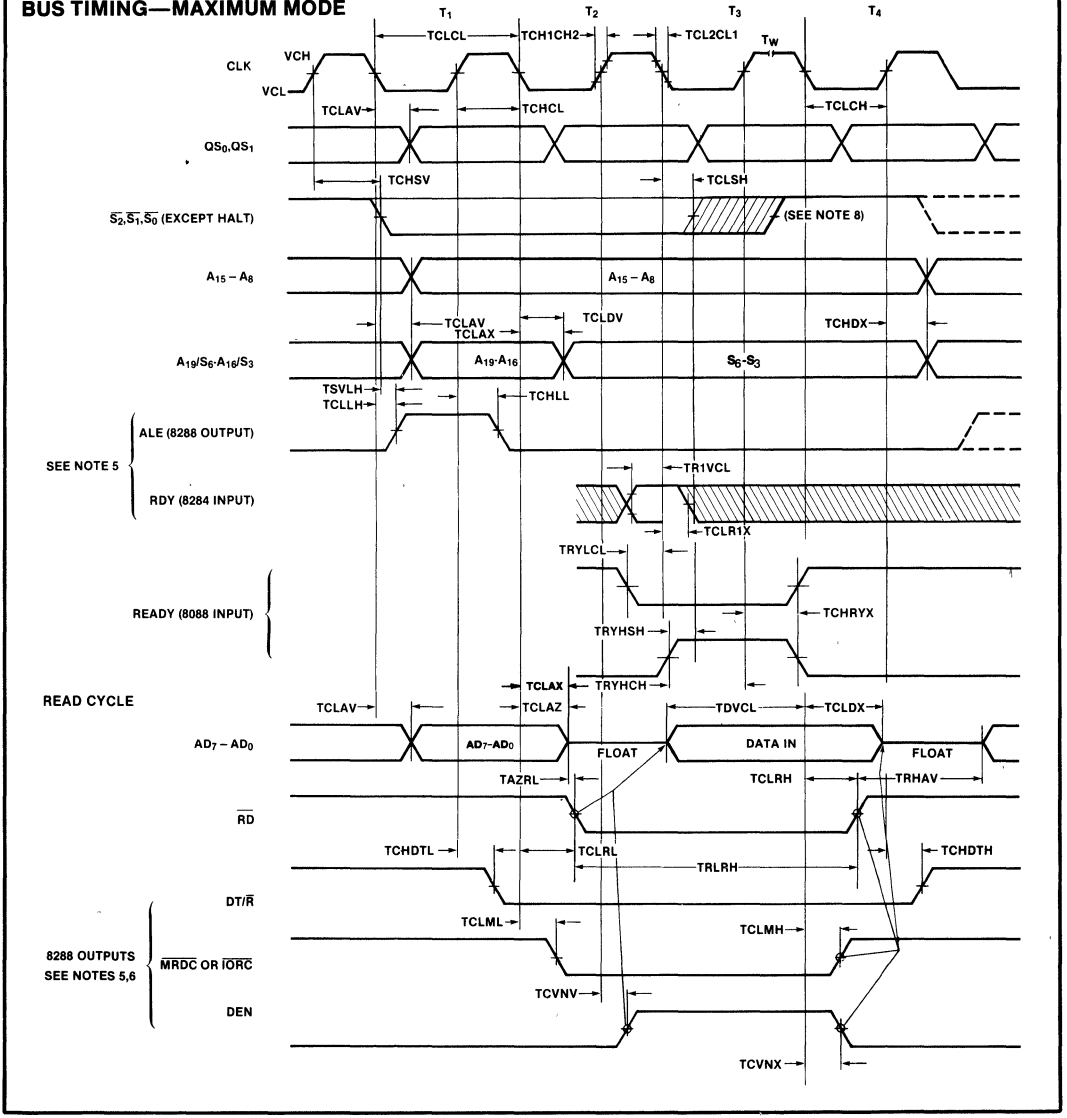
A.C. CHARACTERISTICS

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	
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	C_L = 20-100 pF for all 8088 Outputs in addition to internal loads
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

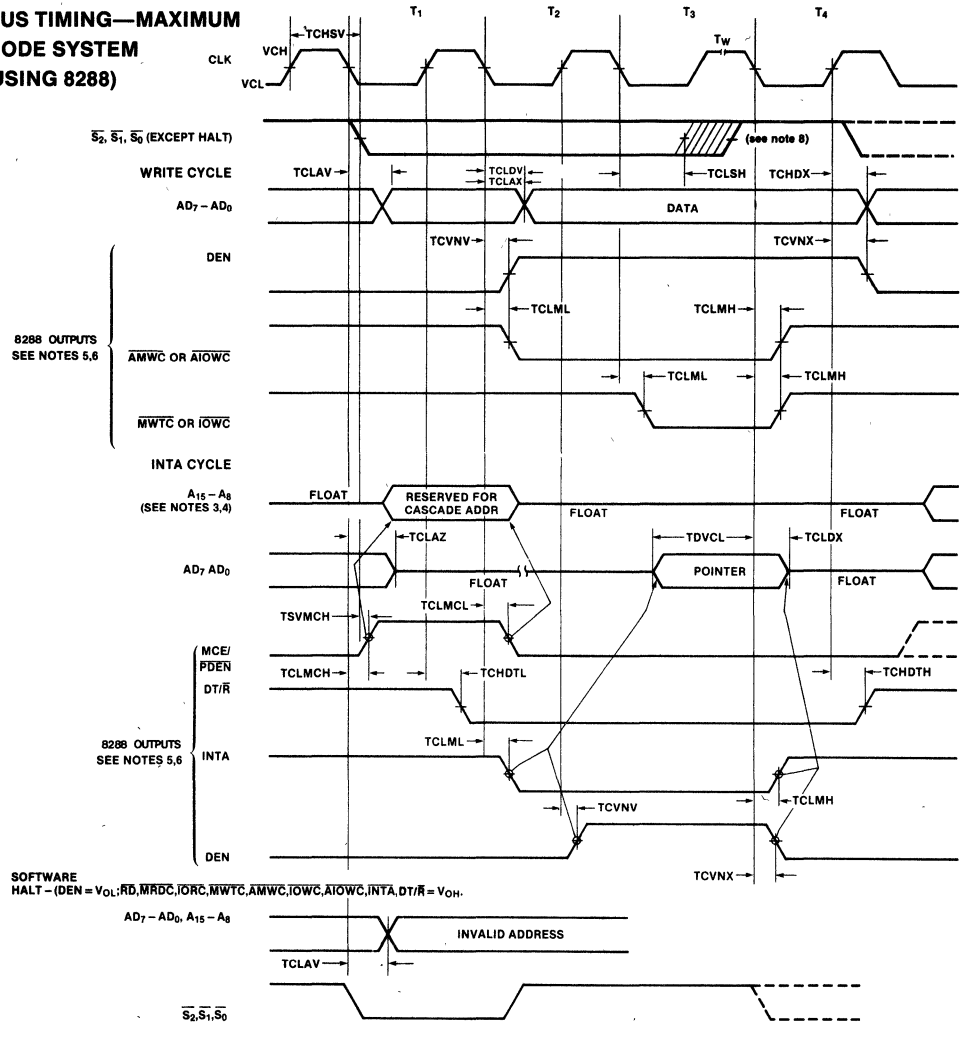
WAVEFORMS

BUS TIMING—MAXIMUM MODE



WAVEFORMS (Continued)

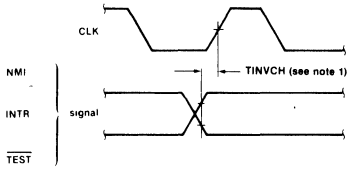
BUS TIMING—MAXIMUM MODE SYSTEM (USING 8288)



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.

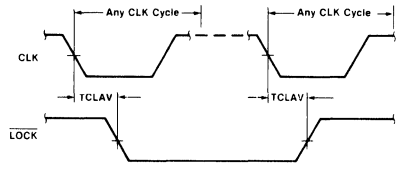
WAVEFORMS (Continued)

ASYNCHRONOUS SIGNAL RECOGNITION

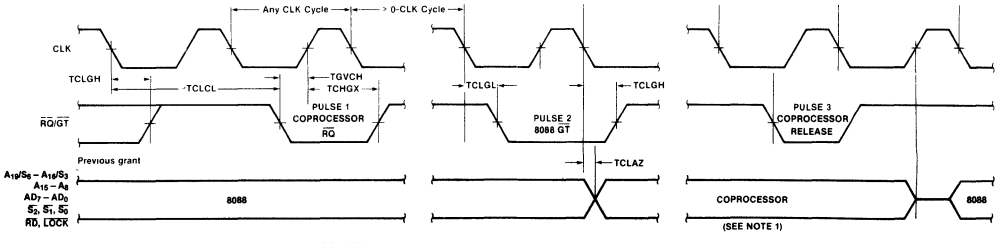


NOTE 1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK

BUS LOCK SIGNAL TIMING (MAXIMUM MODE ONLY)

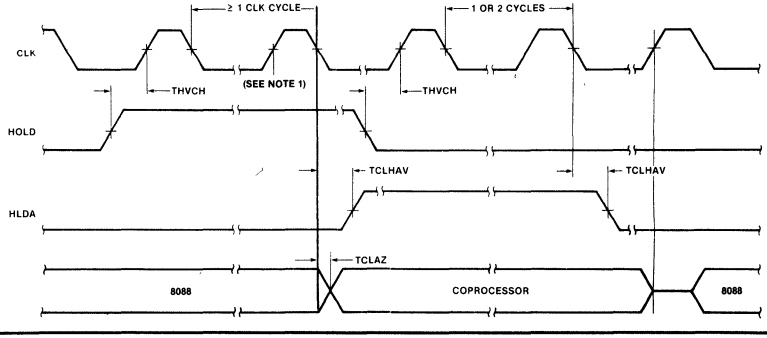


REQUEST/GRANT SEQUENCE TIMING (MAXIMUM MODE ONLY)

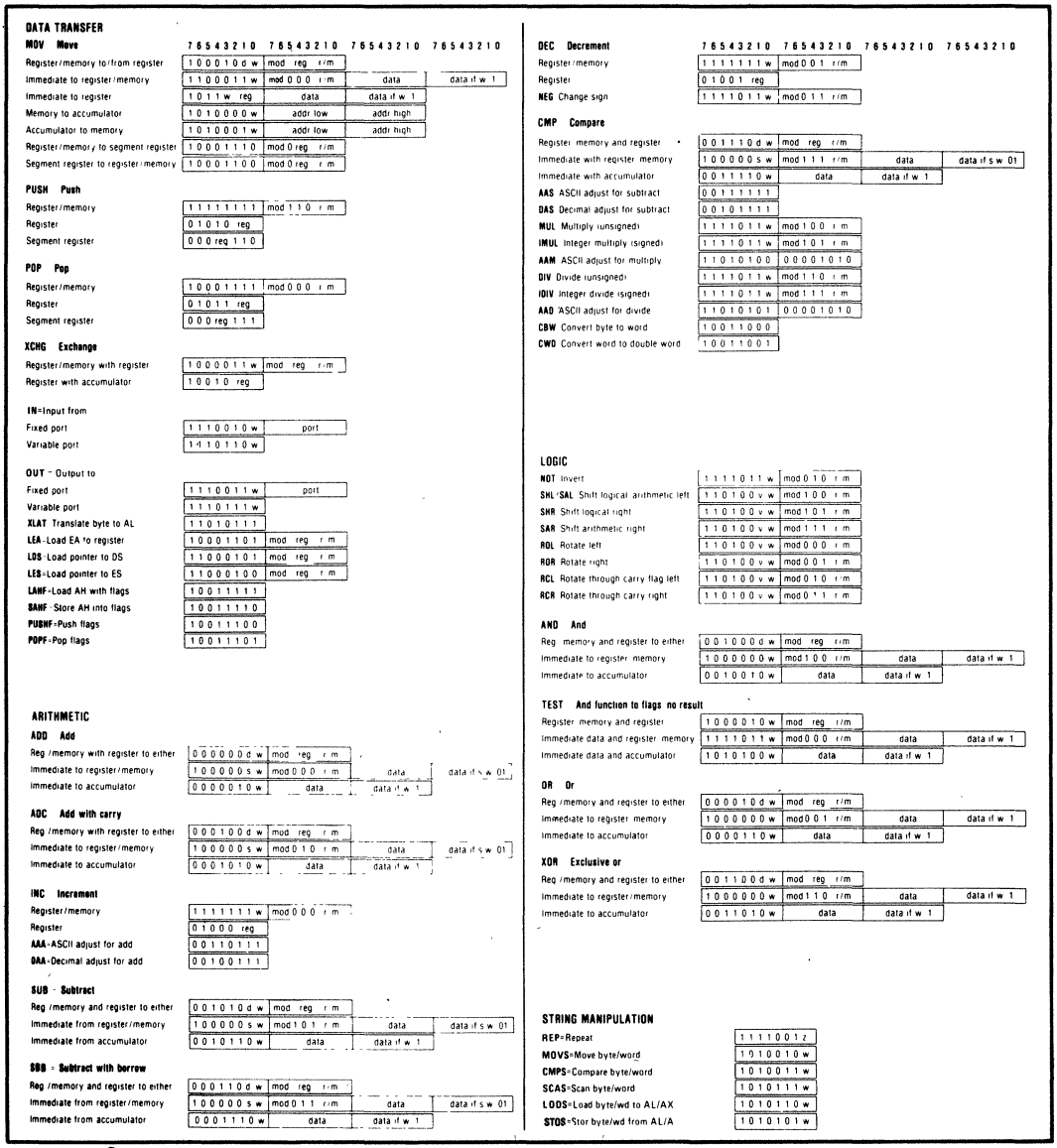


NOTE 1 THE COPROCESSOR MAY NOT DRIVE THE BUSSES OUTSIDE THE REGION SHOWN WITHOUT RISKING CONTENTION

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	mod000 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	mod0reg r/m		
Segment register to register/memory	10001100	mod0reg r/m		

PUSH Push

Register/memory	11111111	mod110 r/m	
Register	01010 reg		
Segment register	000reg110		

POP Pop

Register/memory	10001111	mod000 r/m
Register	01011 reg	
Segment register	000reg111	

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	mod000 r/m	data	data if sw 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	mod010 r/m	data	data if sw 01
Immediate to accumulator	0001010w	data	data if w 1	

INC Increment

Register/memory	1111111w mod000 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	mod101 r/m	data	data if sw 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	mod011 r/m	data	data if sw 01
Immediate from accumulator	0001110w	data	data if w 1	

DEC Decrement

	76543210	76543210	76543210	76543210
Register/memory	1111111w	mod001 r/m		
Register	01001 reg			

NEG Change sign 1111011w mod011 r/m

CMP Compare

Register/memory and register	0011100w	mod reg r/m		
Immediate with register/memory	100000sw	mod111 r/m	data	data if sw 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 mod100 r/m
IMUL Integer multiply (signed)	1111011w mod101 r/m
AAM ASCII adjust for multiply	11010100 00001010
DIV Divide (unsigned)	1111011w mod110 r/m
IDIV Integer divide (signed)	1111011w mod111 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 mod010 r/m
SHL/SAL Shift logical/arithmetic left	110100vw mod100 r/m
SHR Shift logical right	110100vw mod101 r/m
SAR Shift arithmetic right	110100vw mod111 r/m
ROL Rotate left	110100vw mod000 r/m
ROR Rotate right	110100vw mod001 r/m
RCL Rotate through carry flag left	110100vw mod010 r/m
RCR Rotate through carry right	110100vw mod011 r/m

AND And

Reg. memory and register to either	001000dw	mod reg r/m		
Immediate to register/memory	1000000w	mod100 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	mod000 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	mod001 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	mod110 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 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 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

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

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 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 (80188) and cost effective 6 MHz (80188-6) 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**

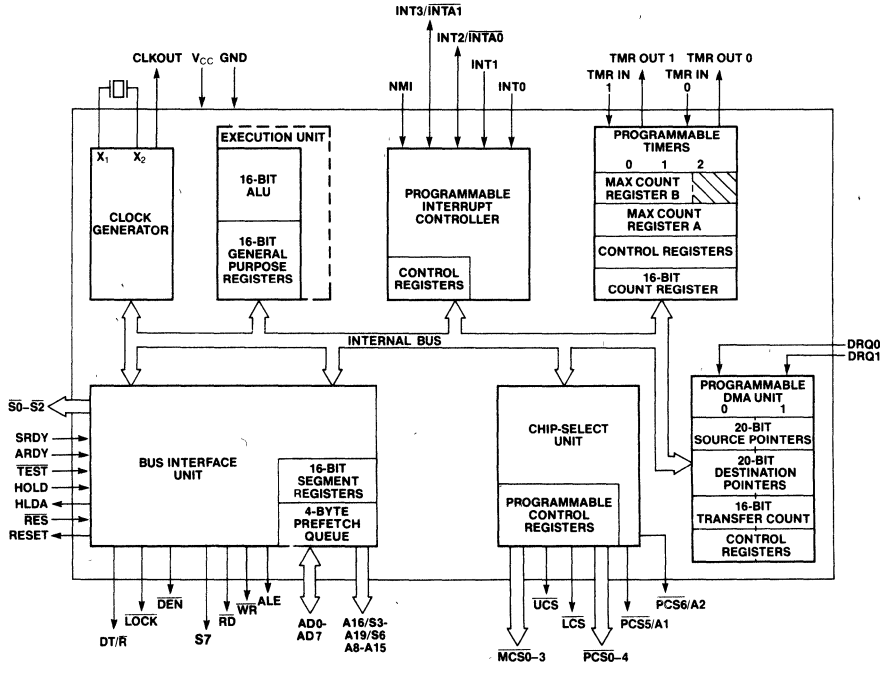


Figure 1. iAPX 188 Block Diagram

OCTOBER 1984
ORDER NUMBER: 210706-004

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.

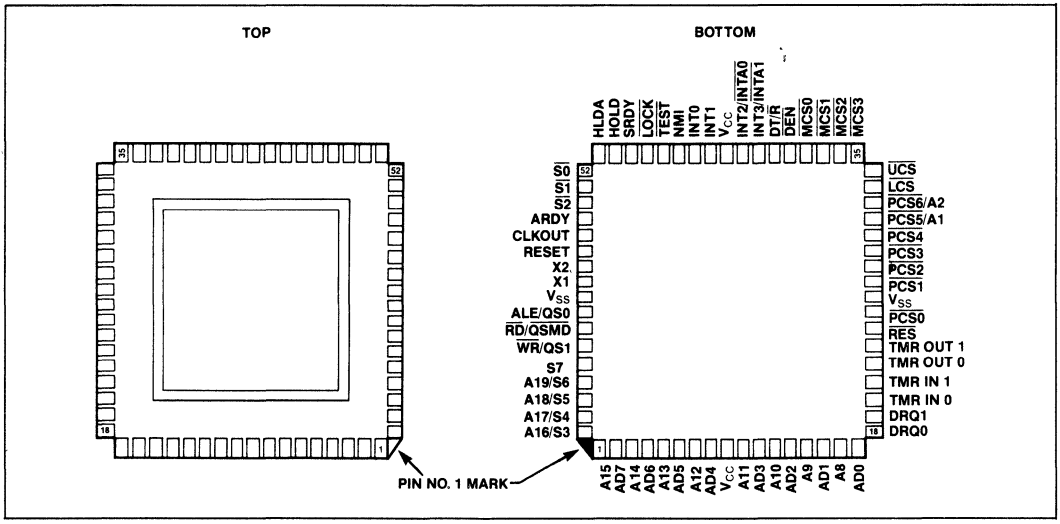


Figure 2. 80188 Pinout Diagram

Table 1. 80188 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 80188 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 80188 to immediately terminate its present activity, clear the internal logic, and enter a dormant state. This signal may be asynchronous to the 80188 clock. The 80188 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 IN 1	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: S6: Low = Processor Cycle; High = DMA Cycle S3,S4, and S5 are defined as LOW during T_2–T_4.
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 holds 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.

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 may be 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.

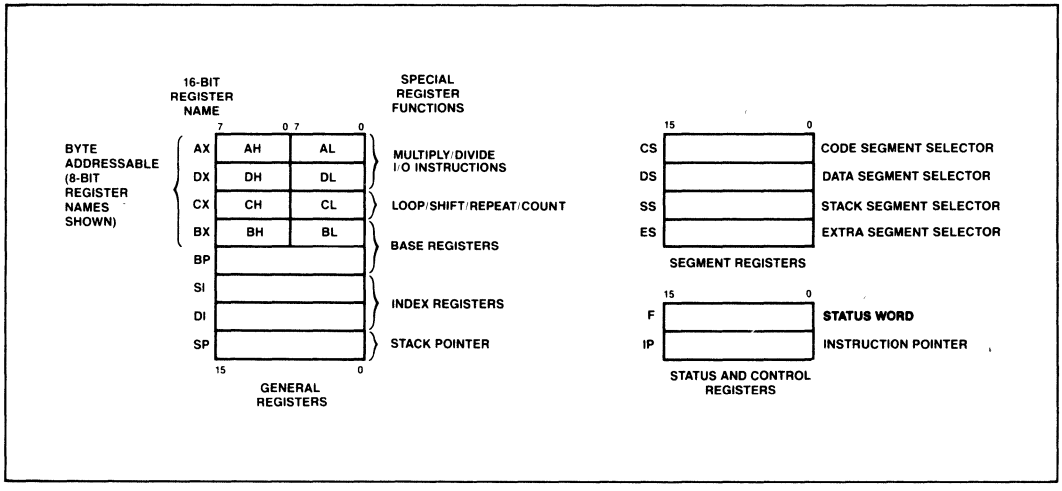


Figure 3a. 80188 General Purpose Register Set

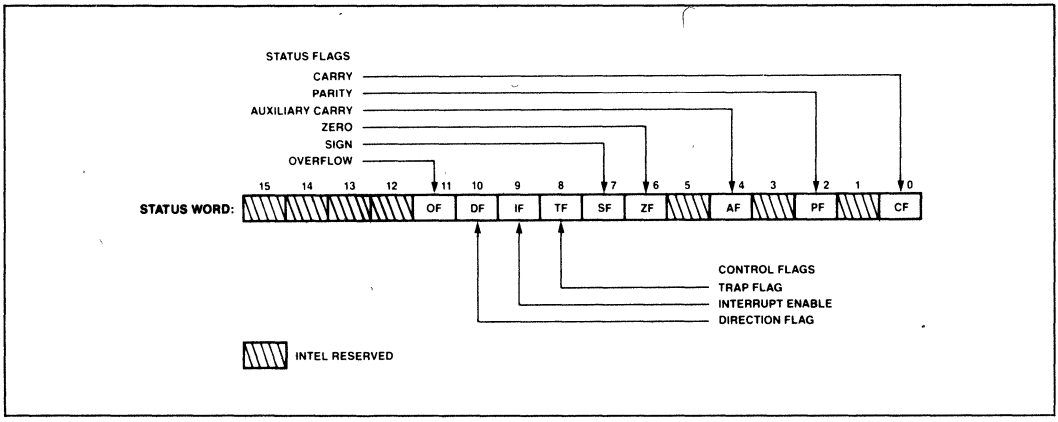


Figure 3b. Status Word Format

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 $\overline{TEST}$ 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

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.

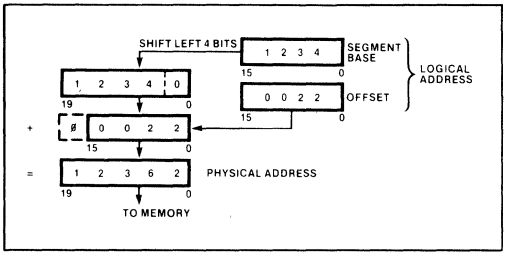


Figure 5. Two Component Address

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.

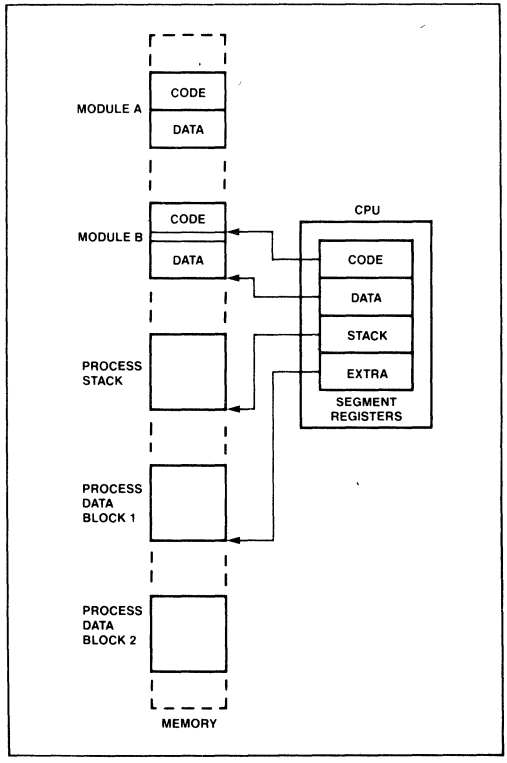


Figure 6. Segmented Memory Helps Structure Software

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.)

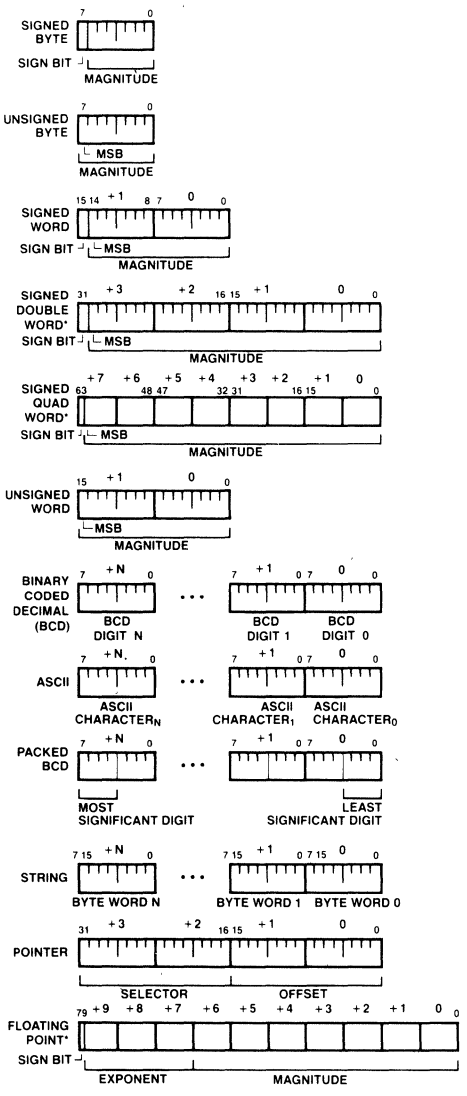
In general, individual data elements must fit within defined segment limits. Figure 7 graphically represents the data types supported by the iAPX 188.

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.

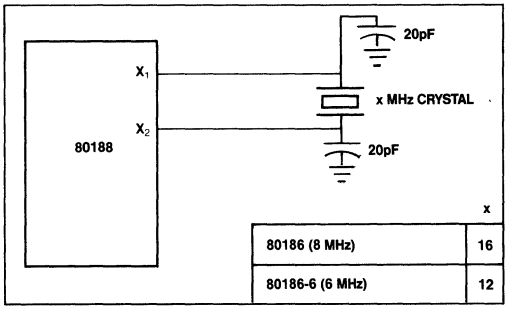


Figure 8. Recommended iAPX 188 Crystal Configuration

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70°C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_L (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

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.

LOCAL BUS CONTROLLER

The iAPX 188 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 188 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 188 or to strobe data from the iAPX 188 to memory. The ALE line provides a strobe to address latches for the multiplexed address/data bus. The iAPX 188 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 188 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 188 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 188 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 188 when there is more than one alternate local bus master. When the iAPX 188 relinquishes control of the local bus, it floats $\overline{DEN}$, $\overline{RD}$, $\overline{WR}$, $\overline{S0}$-$\overline{S2}$, $\overline{LOCK}$, AD0-AD-15, A16-A19, $\overline{S7}$, and DT/$\overline{R}$ to allow another master to drive these lines directly.

The iAPX 188 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 188 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 AD0-7, A8-19, S7, DT/$\overline{R}$.
- Drive ALE LOW (ALE is never floated).
- Drive HLDA LOW.

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).

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

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. Externally the BIU will still run two bus cycles for each 16-bit operation.

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.

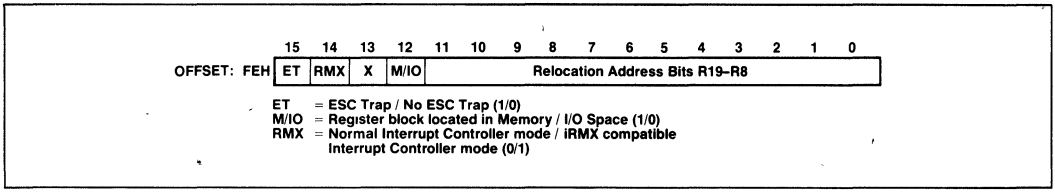


Figure 9. Relocation Register

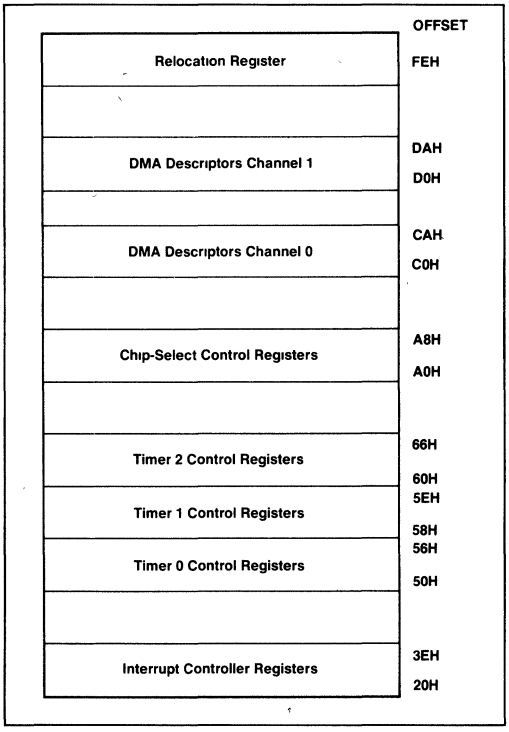


Figure 10. Internal Register Map

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.

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 described in 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.

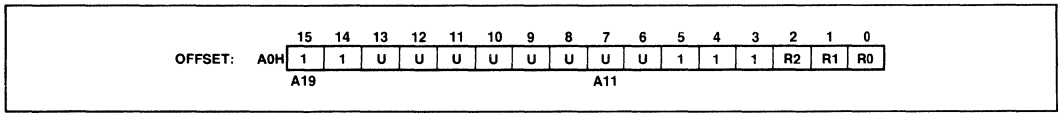


Figure 11. UMCS Register

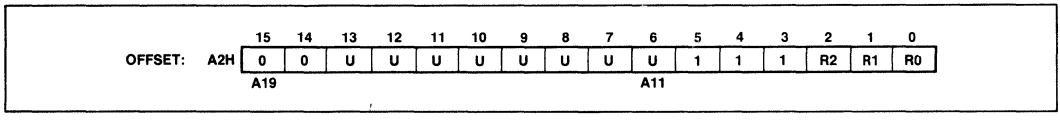


Figure 12. LMCS Register

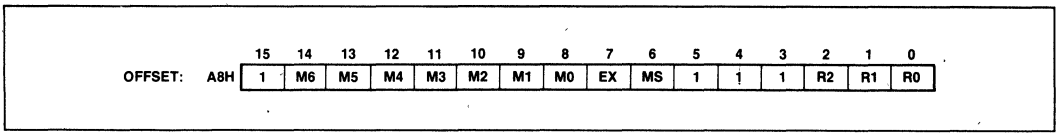


Figure 13. MPCS Register

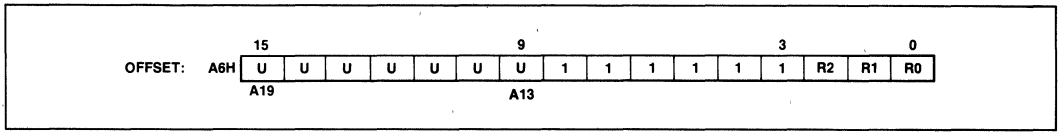


Figure 14. MMCS Register

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 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.

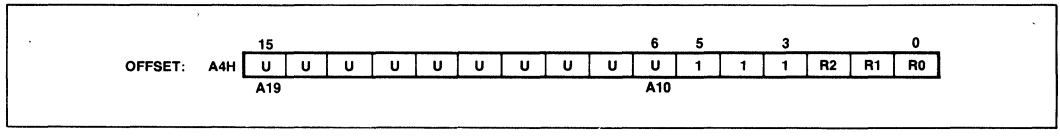


Figure 15. PACS Register

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

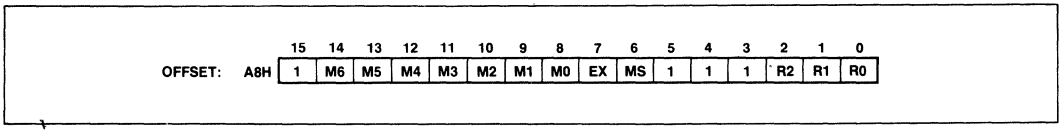


Figure 16. MPCS Register

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

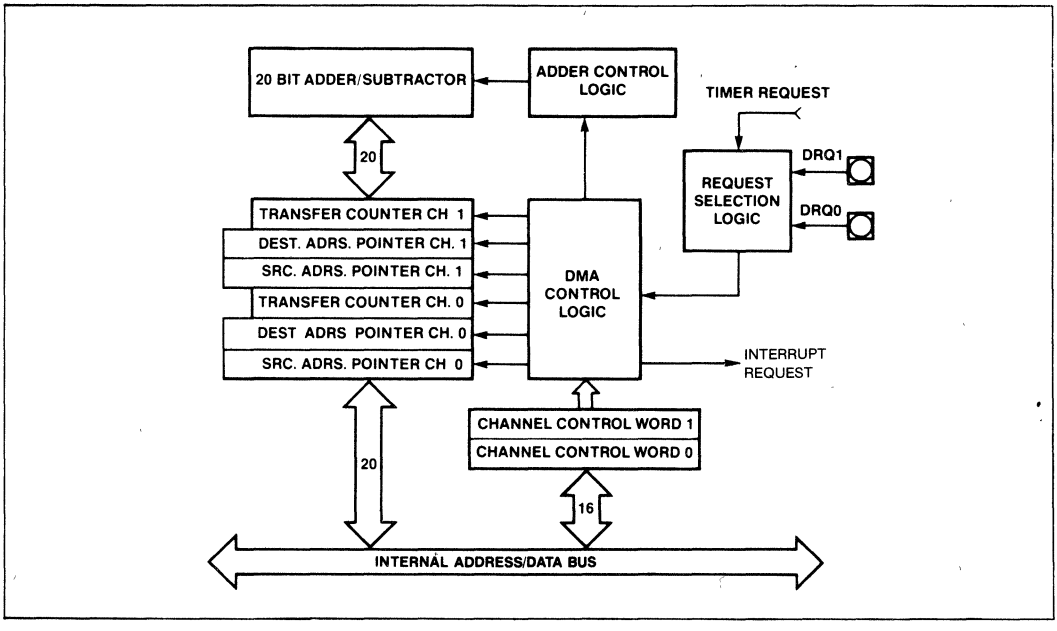


Figure 17. DMA Unit Block Diagram

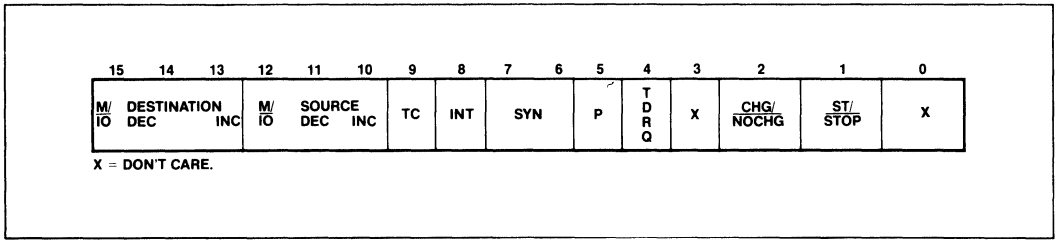


Figure 18. DMA Control Register

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.

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

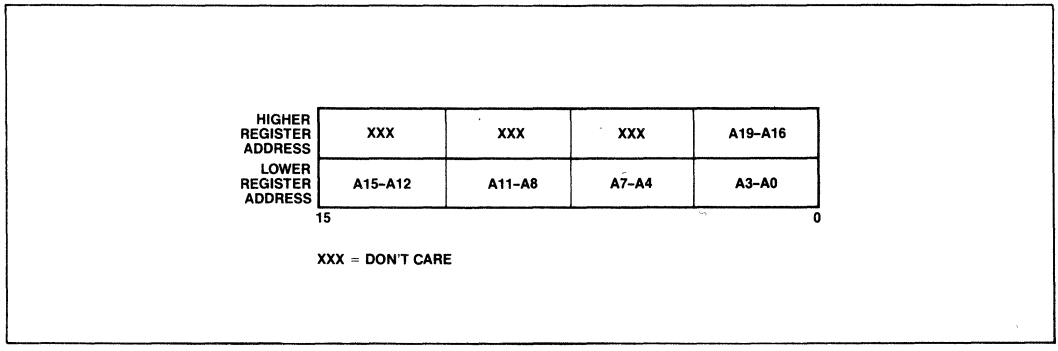


Figure 18a. DMA Memory Pointer Register Format

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.

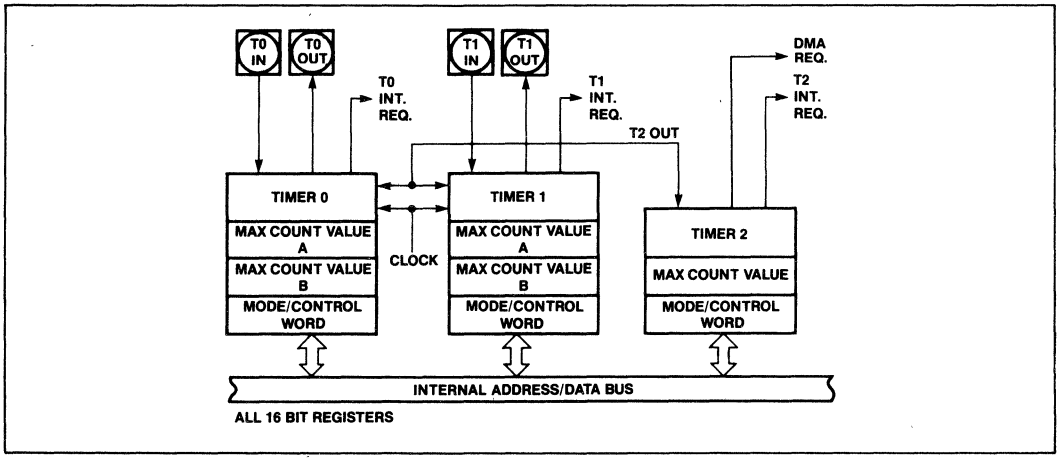


Figure 19. Timer Block Diagram

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

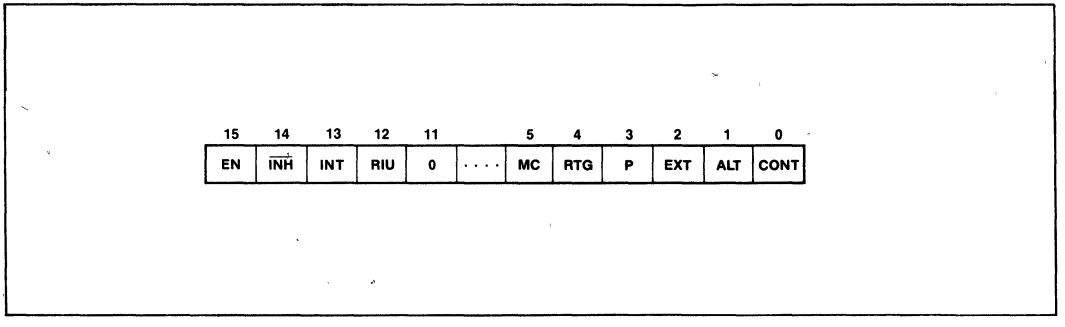


Figure 20. Timer Mode/Control Register

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.

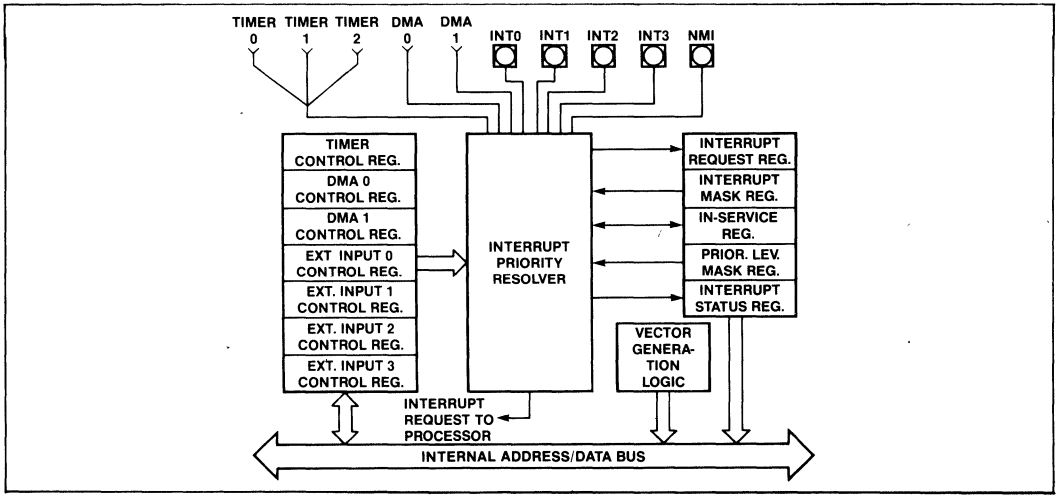


Figure 21. Interrupt Controller Block Diagram

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.

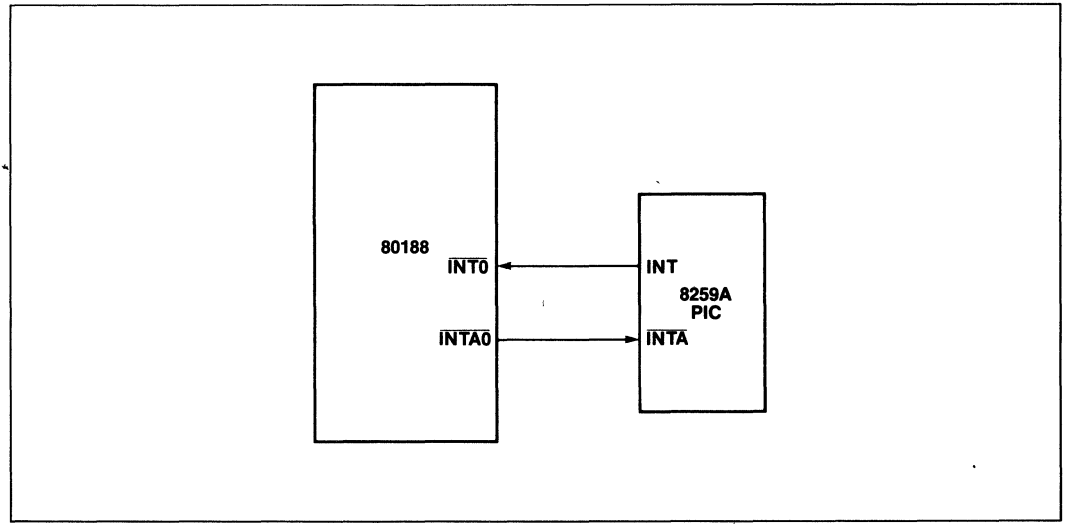


Figure 22. Cascade Mode Interrupt Connection

	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.

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 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.

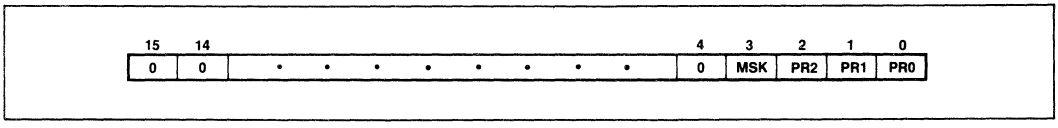


Figure 27. Timer/DMA Control Register Formats

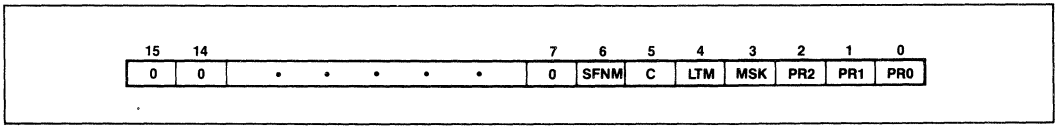


Figure 28. INT0/INT1 Control Register Formats

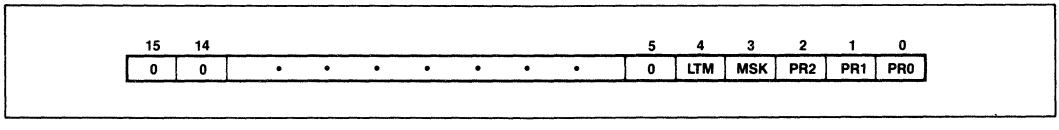


Figure 29. INT2/INT3 Control Register Formats

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.

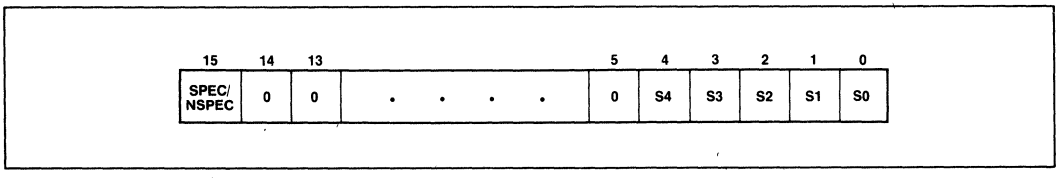


Figure 30. EOI Register Format

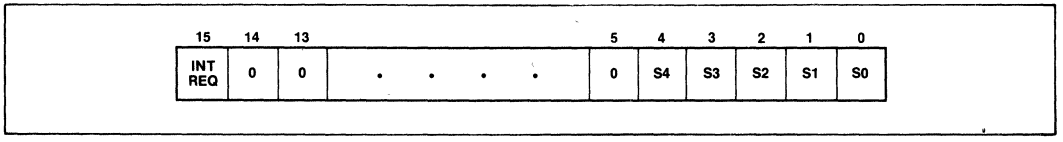


Figure 31. Poll Register Format

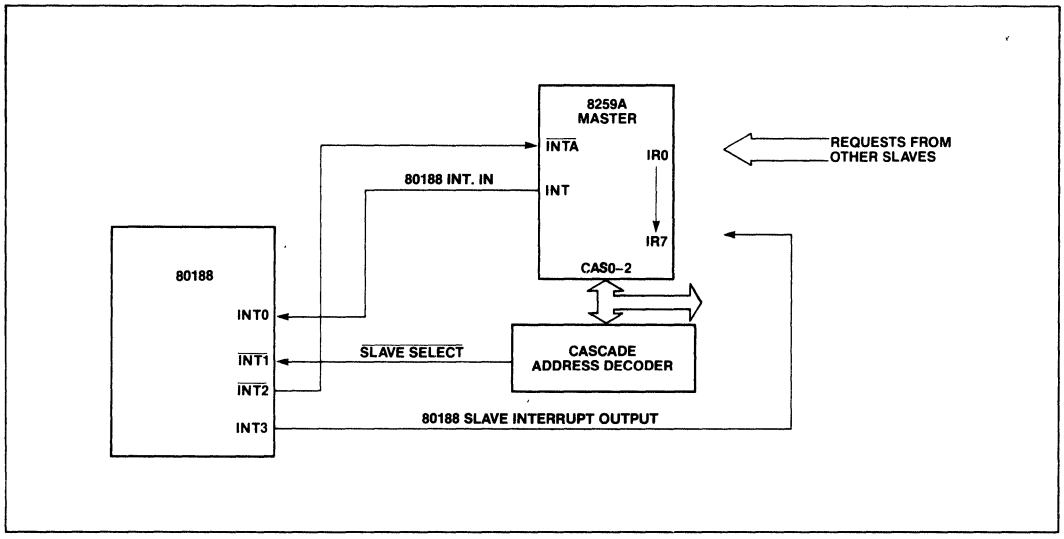


Figure 32. iRMX 86 Interrupt Controller Interconnection

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.

	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 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.

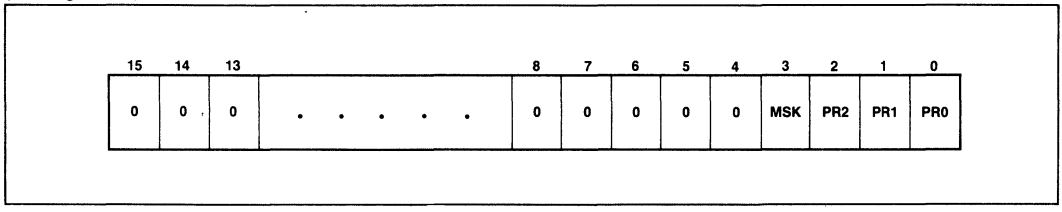


Figure 36. Control Word Format

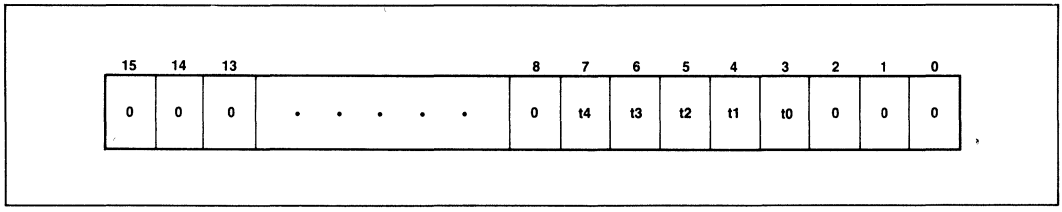


Figure 37. Interrupt Vector Register Format

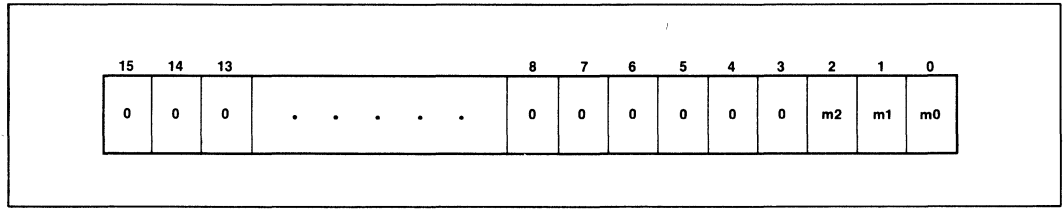


Figure 38. Priority Level Mask Register

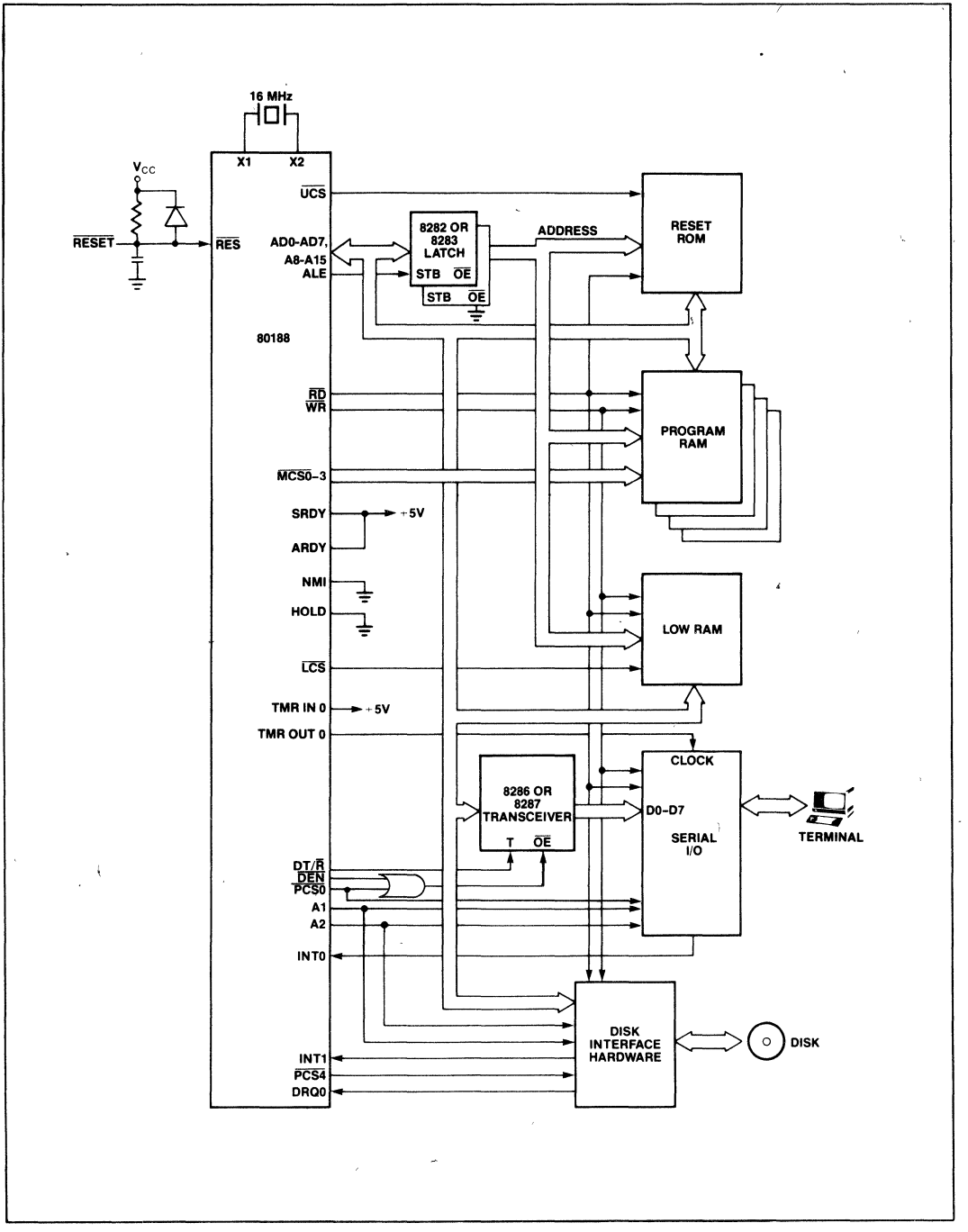


Figure 39. Typical iAPX 188 Computer

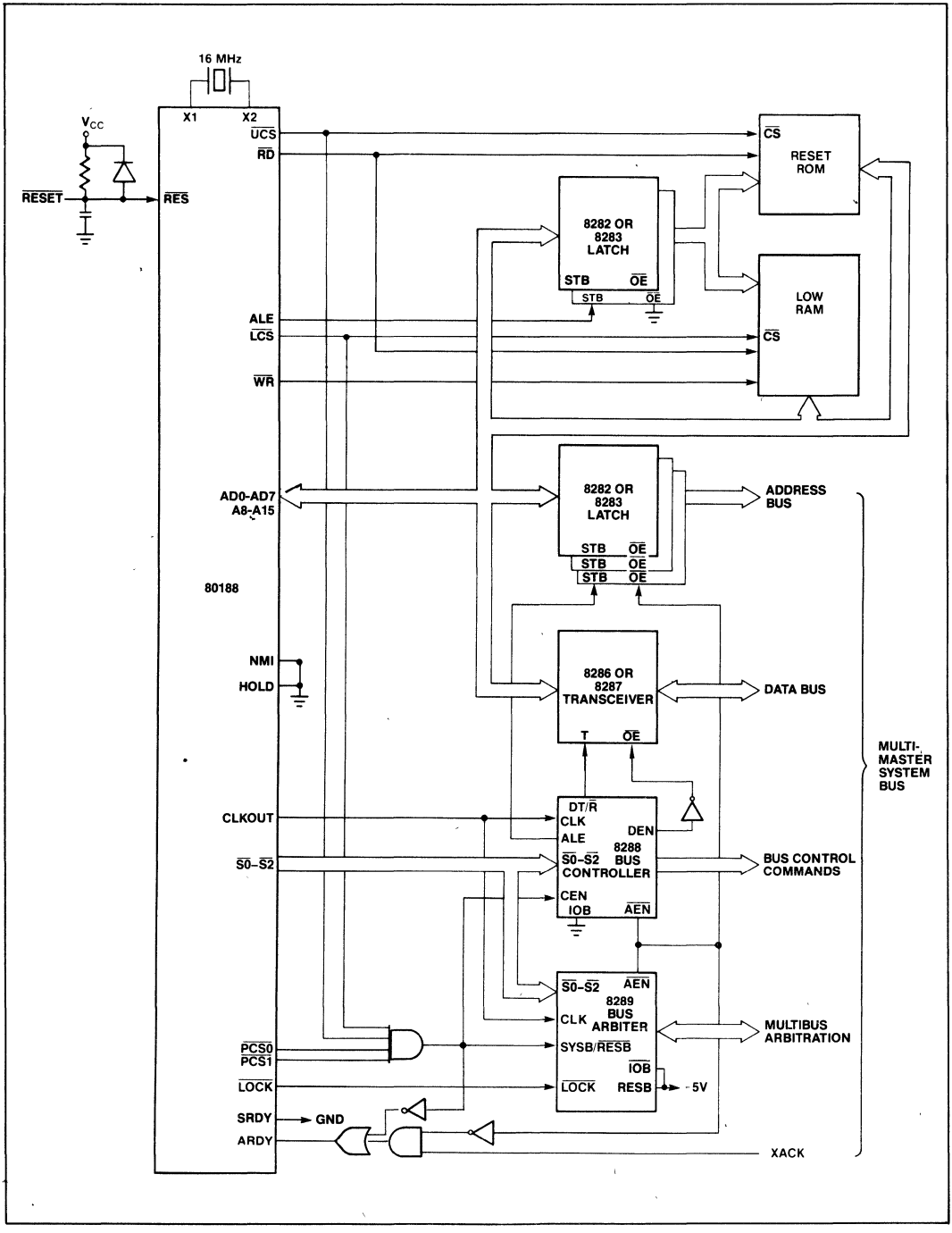


Figure 40. Typical iAPX 188 Multi-Master Bus Interface

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.

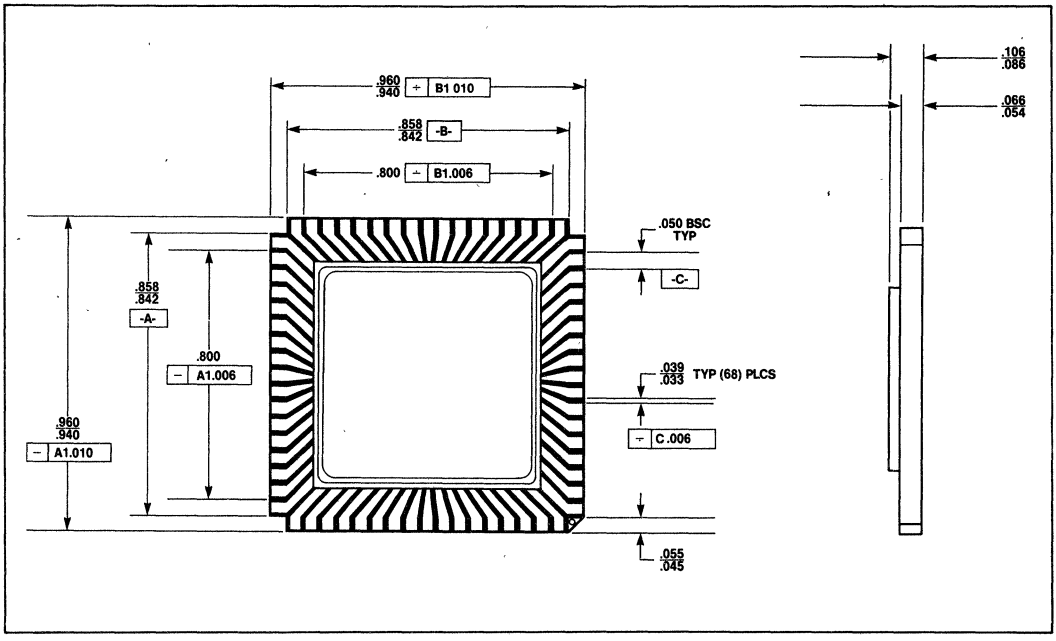
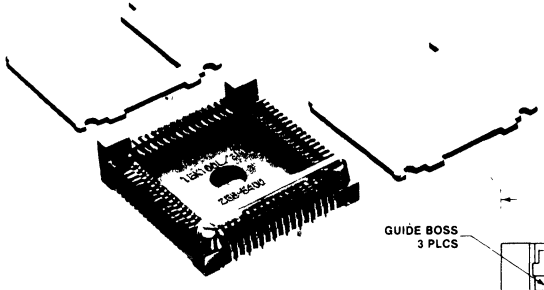
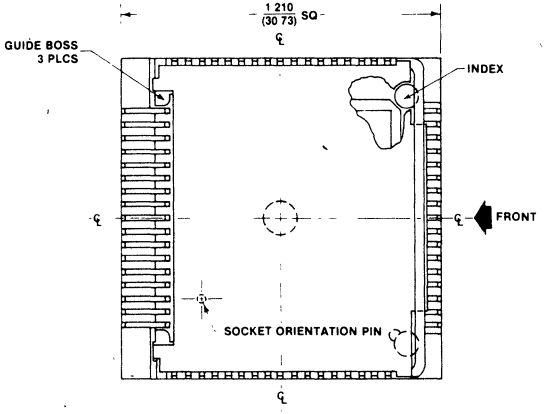
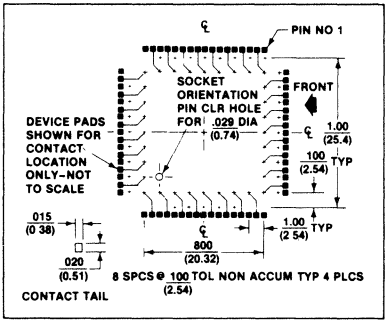


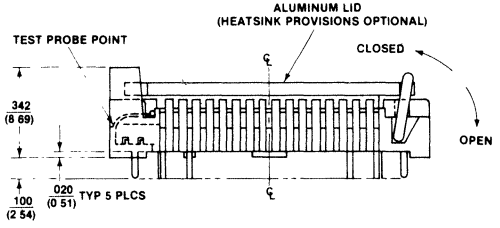
Figure 41. 80188 JEDEC Type A Package





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 ($\overline{RES}$)	3.0	V_{CC} + 0.5	Volts	
V_{OL}	Output Low Voltage		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		$\frac{550}{450}$	mA	Max measured at $\frac{T_A = 0°C}{T_A = 70°C}$
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
T_{DVCL}	Data in Setup (A/D)	20		ns	
T_{CLDX}	Data in Hold (A/D)	10		ns	
T_{ARYHCH}	Asynchronous Ready (AREADY) active setup time*	20		ns	
T_{ARYLCL}	AREADY inactive setup time	35		ns	
T_{ARYCHL}	Asynchronous Ready Inactive hold time	15		ns	
T_{CHARYX}	AREADY hold time	15		ns	
T_{SRYCL}	Synchronous Ready (SREADY) transition setup time	20		ns	
T_{CLSRY}	SREADY transition hold time	15		ns	
T_{HVCL}	HOLD Setup*	25		ns	
T_{INVCH}	INTR, NMI, $\overline{TEST}$, TIMERIN, Setup*	25		ns	
T_{INVCL}	DRQ0, DRQ1, Setup*	25		ns	

*To guarantee recognition at next clock.

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	55	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	10	70	10	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)	10	70	10	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	5	50	5	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	65	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	
T_{CHDX}	Status Hold Time	10		10		ns	
T_{AVCH}	Address Valid to clock high	10		10		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

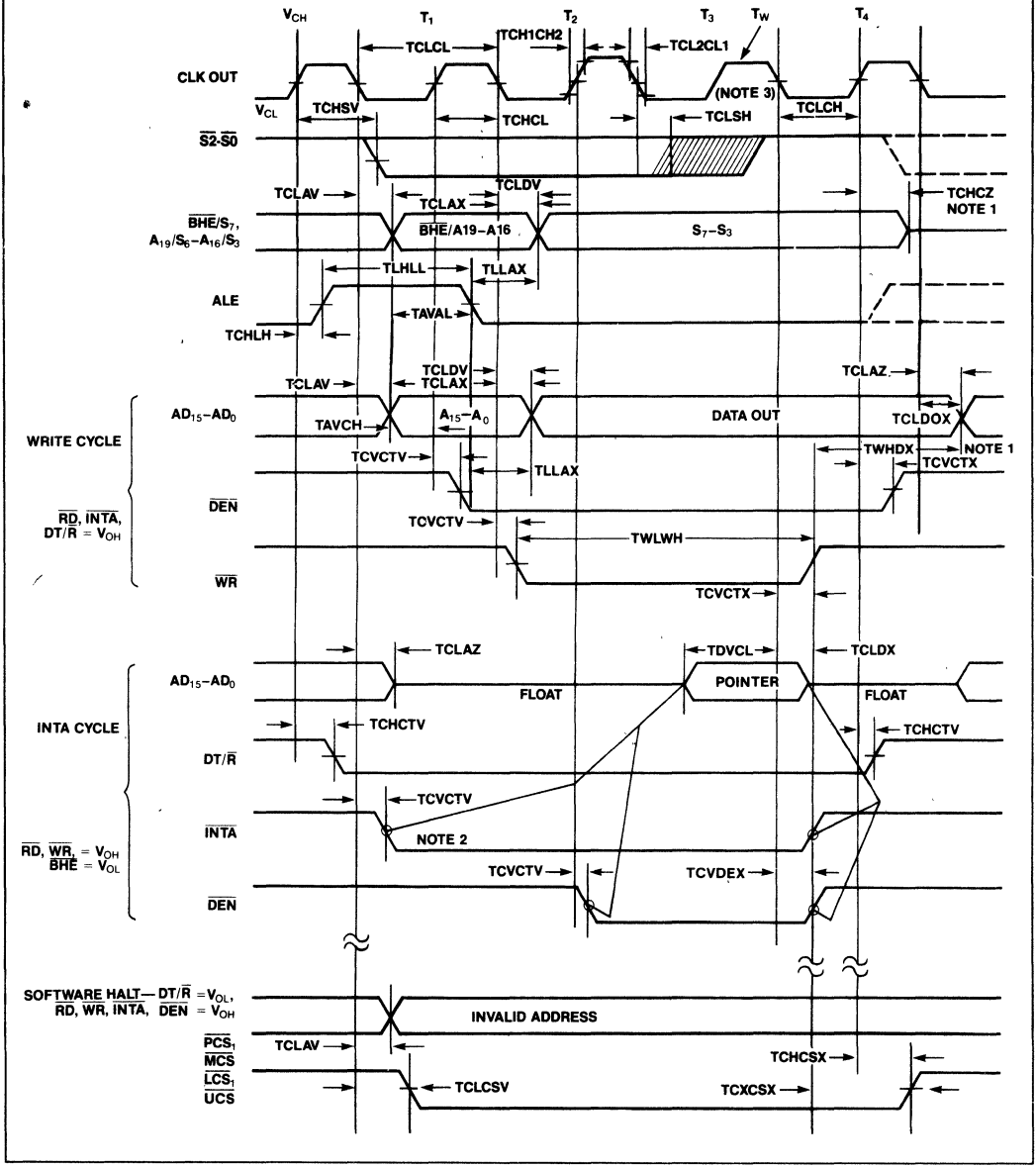
		80188 (8 MHz)		80188-6 (6 MHz)			
Symbol	**Parameters**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
T_{CKIN}	CLKIN Period	62.5	250	83	250	ns	
T_{CKHL}	CLKIN Fall Time		10		10	ns	3.5 to 1.0 volts
T_{CKLH}	CLKIN Rise Time		10		10	ns	1.0 to 3.5 volts
T_{CLCK}	CLKIN Low Time	25		33		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

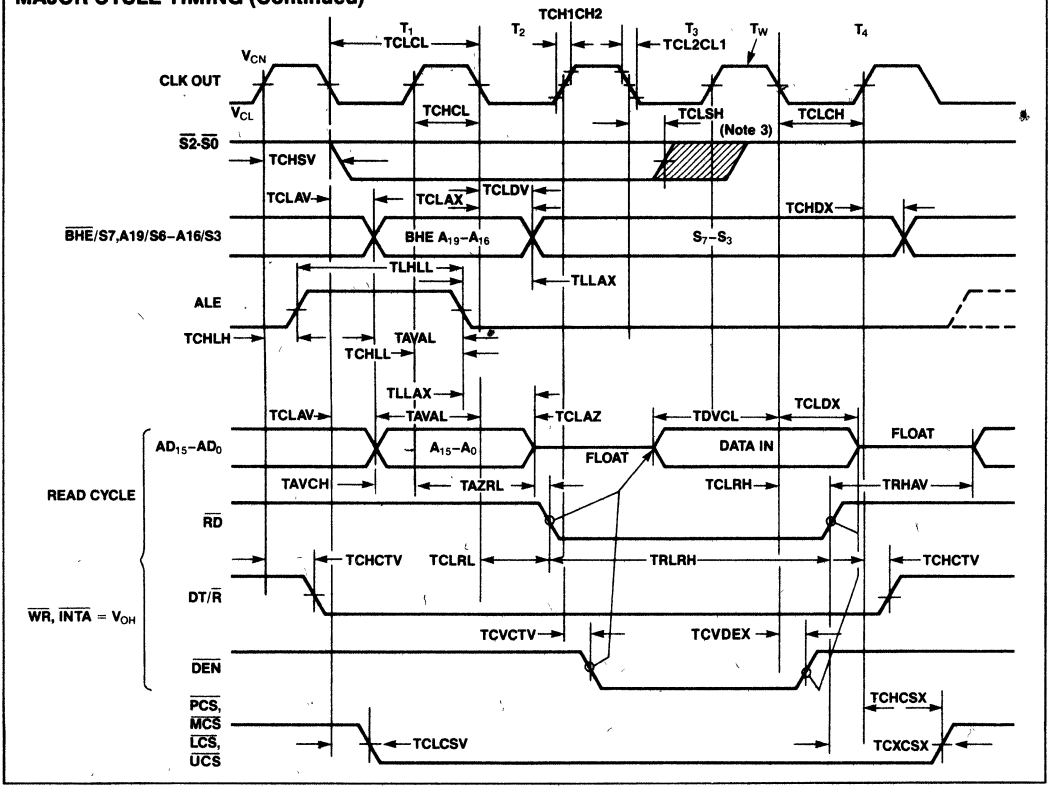
WAVEFORMS

MAJOR CYCLE TIMING



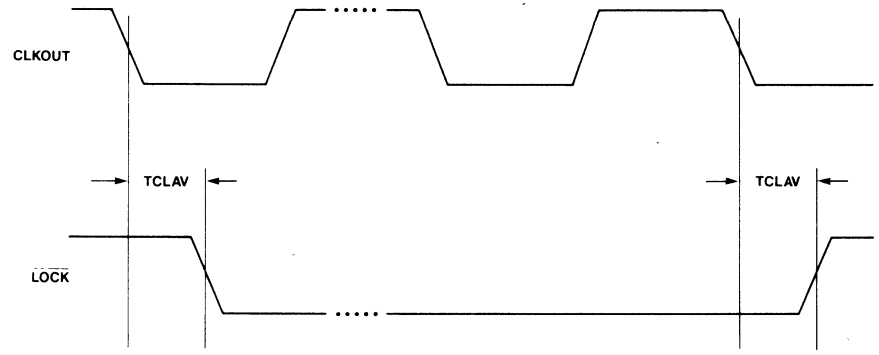
WAVEFORMS (Continued)

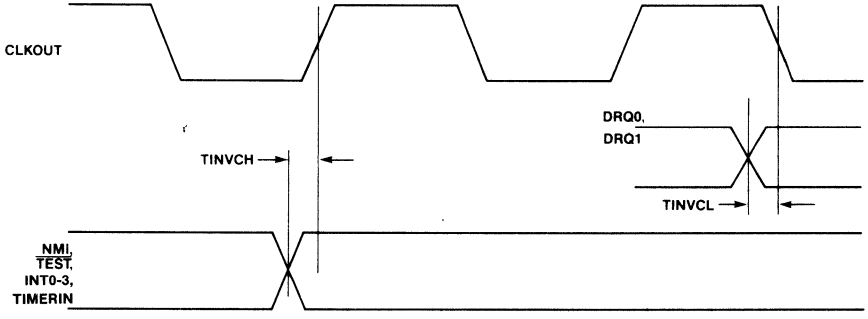
MAJOR CYCLE TIMING (Continued)

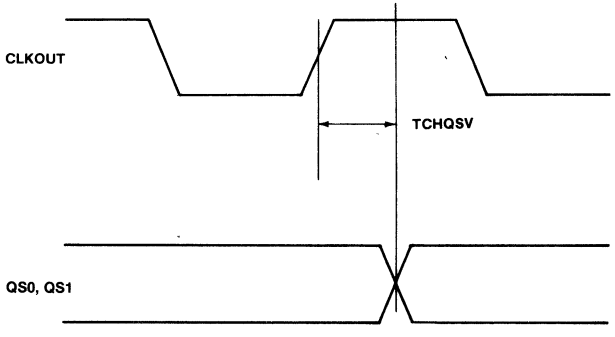


NOTES:
1. Following a Write cycle, the Local Bus is floated by the 80188 only when the 80188 enters a "Hold Acknowledge" state.
2 INTA occurs one clock later in RMX-mode.
3 Status inactive just prior to T_4

WAVEFORMS (Continued)

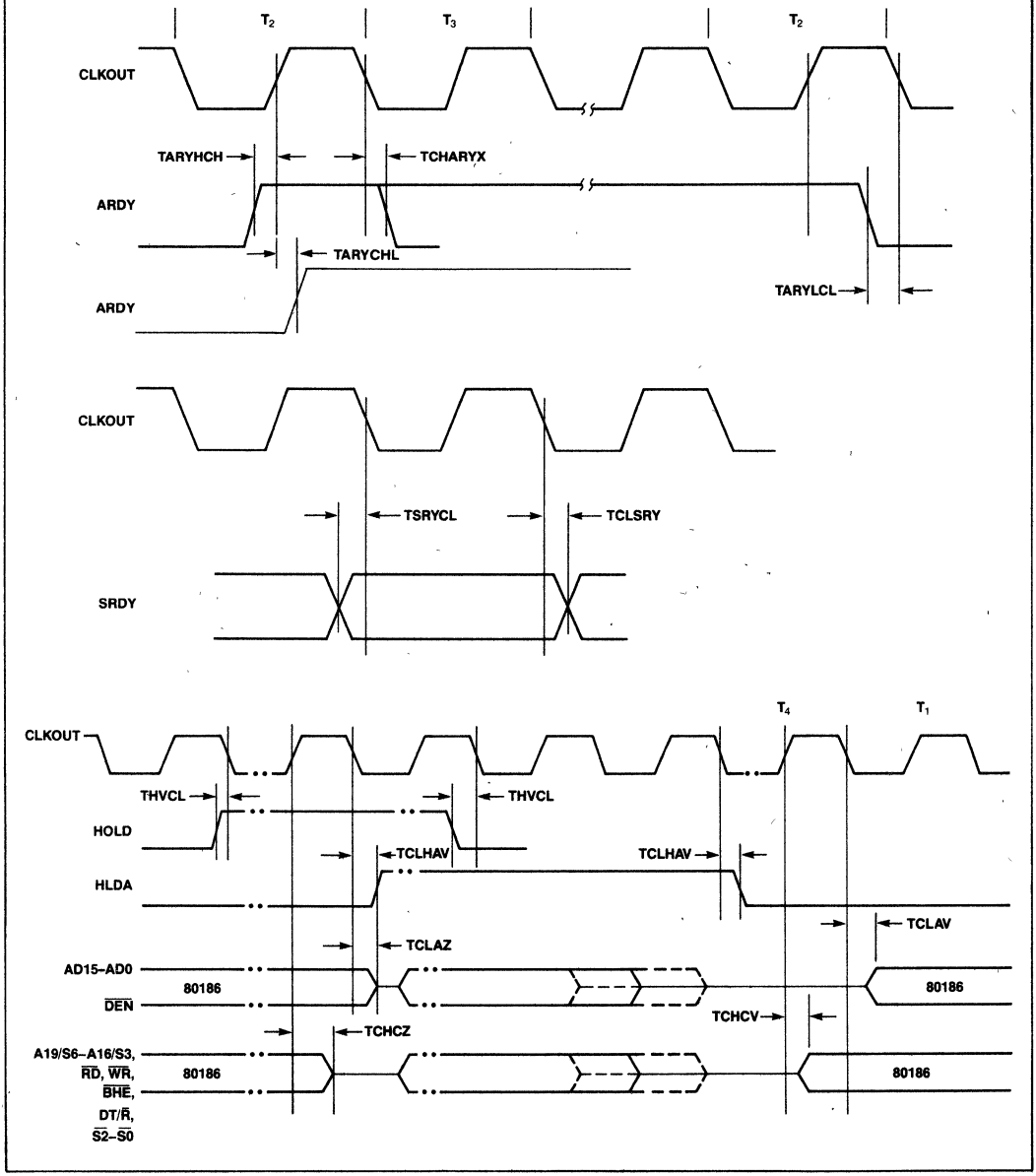






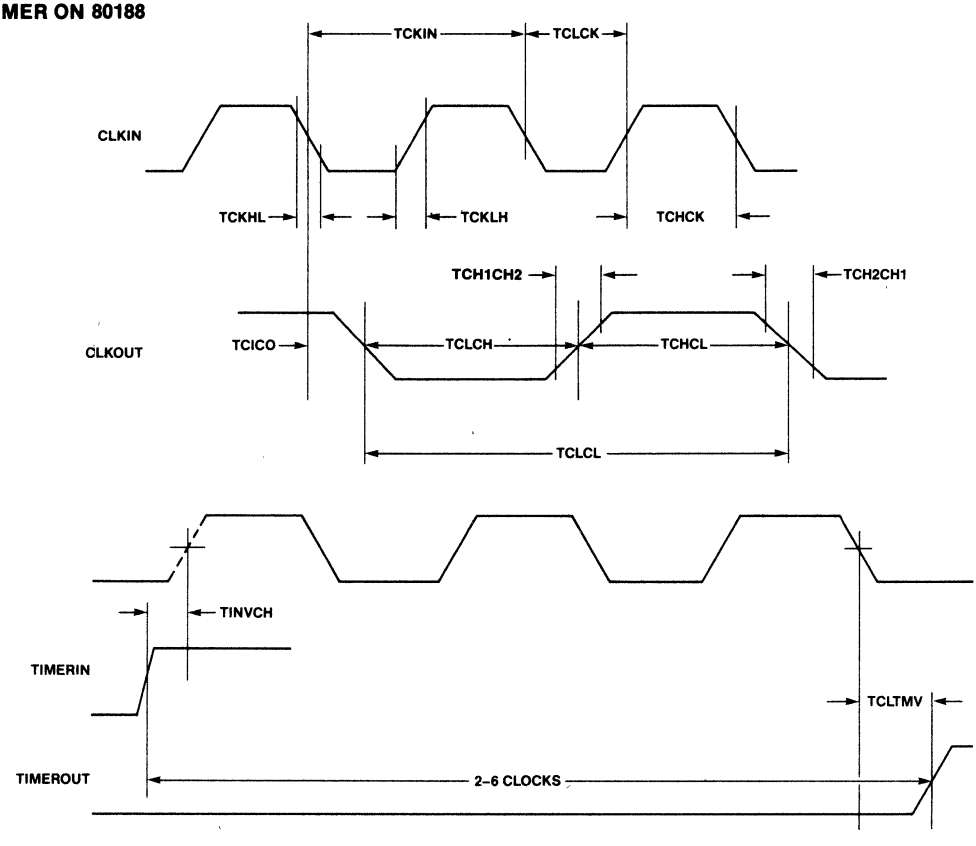
WAVEFORMS (Continued)

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 = 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.

*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	19	
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	14	
Direct within segment	1 1 1 0 1 0 0 1 | disp-low | disp-high	14	
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	14	
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 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 1 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.

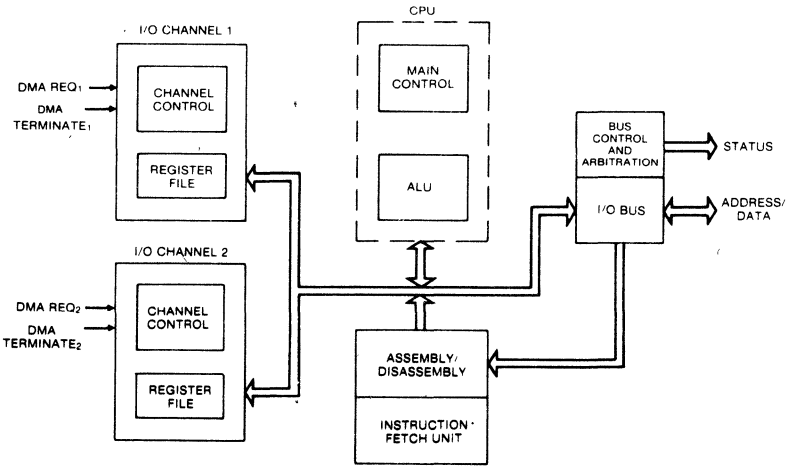


Figure 1. 8089 I/O Processor Block Diagram

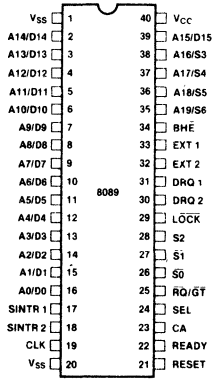


Figure 2. 8089 Pin Configuration

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.

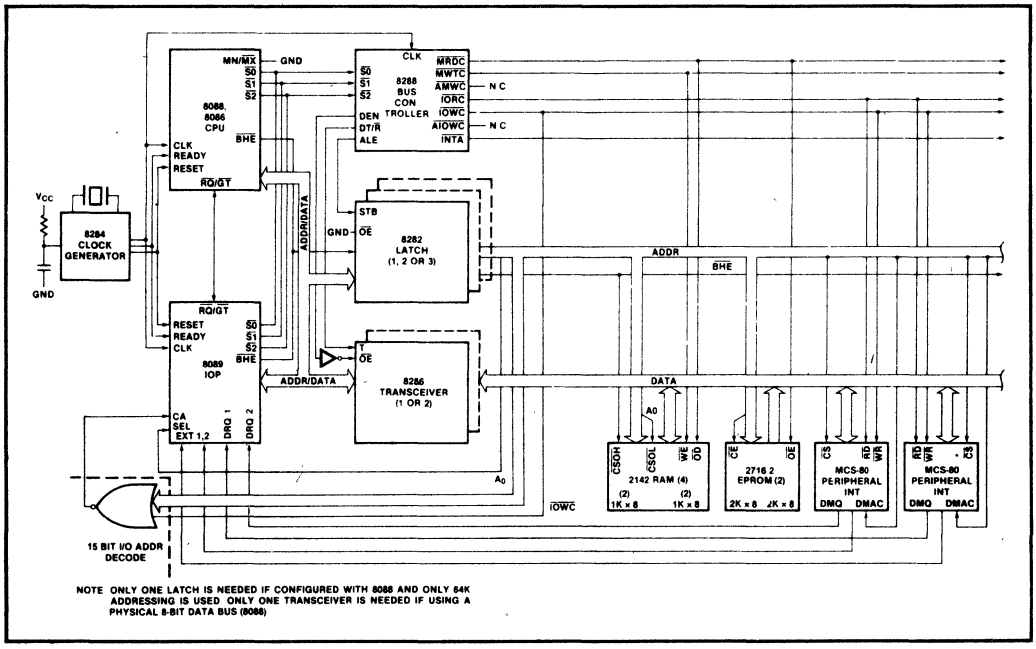


Figure 3. Typical iAPX 86/11, 88/11 Configuration with 8089 in LOCAL Mode, 8088, 8086 in MAX Mode

- 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.

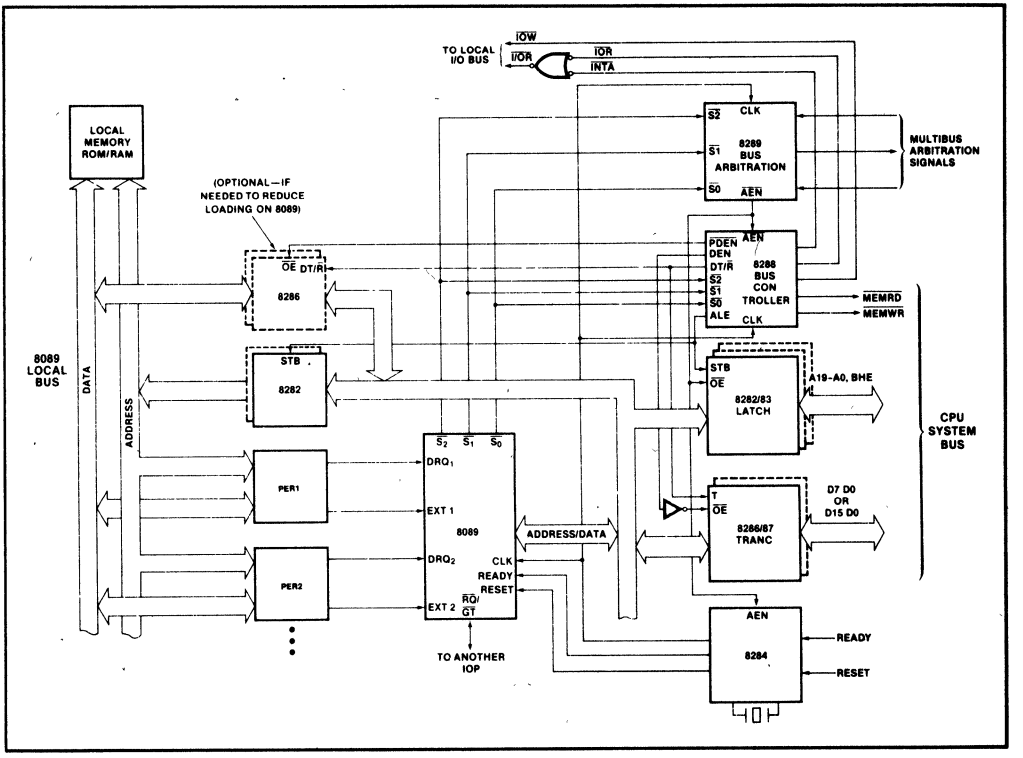


Figure 4. Typical REMOTE Configuration

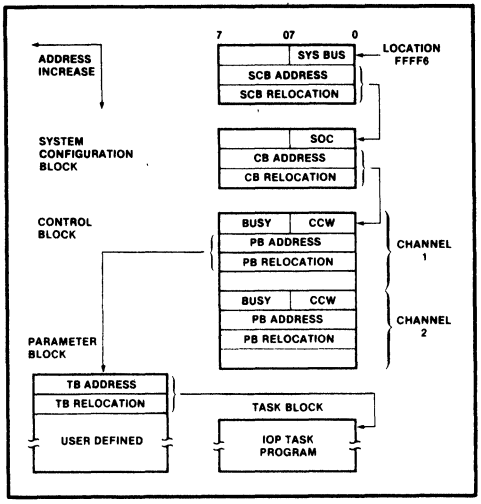


Figure 5. Communication Data Structure Hierarchy

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.

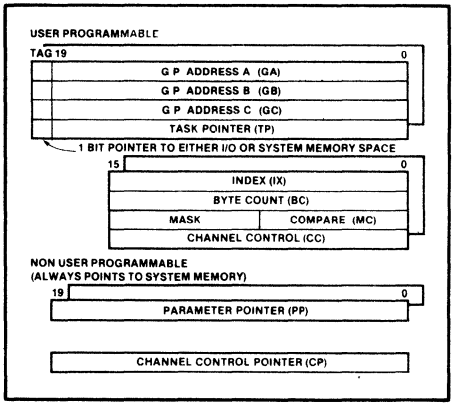


Figure 6. Register Model

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.

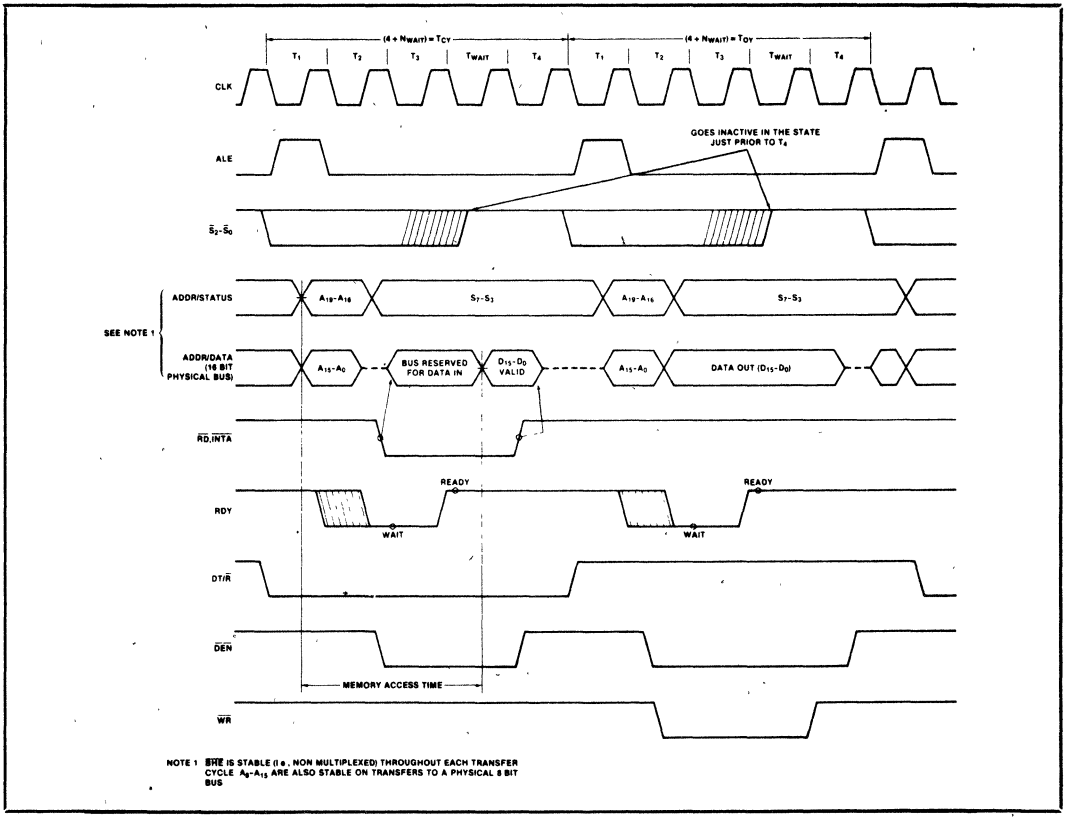


Figure 7. 8089 Bus Operation

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

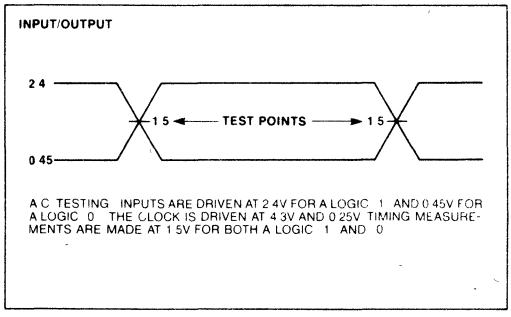
A.C. CHARACTERISTICS (Continued)

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

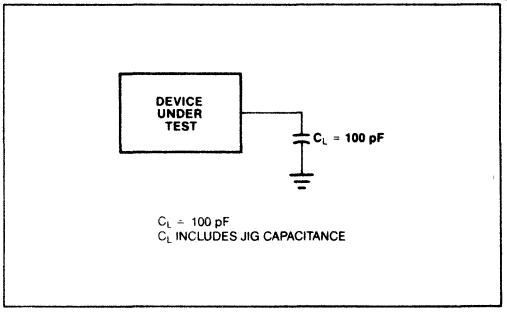
NOTES. 1 Signal at 8284 or 8288 shown for reference only
2 Setup requirement for asynchronous signal only to guarantee recognition at next CLK
3 Aplies only to T3 and TW states
4 Applies only to T2 state
5 Applies only if RQ/GT Mode 1 C_L=30pf, 2 7 KΩ pull up to Vcc

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

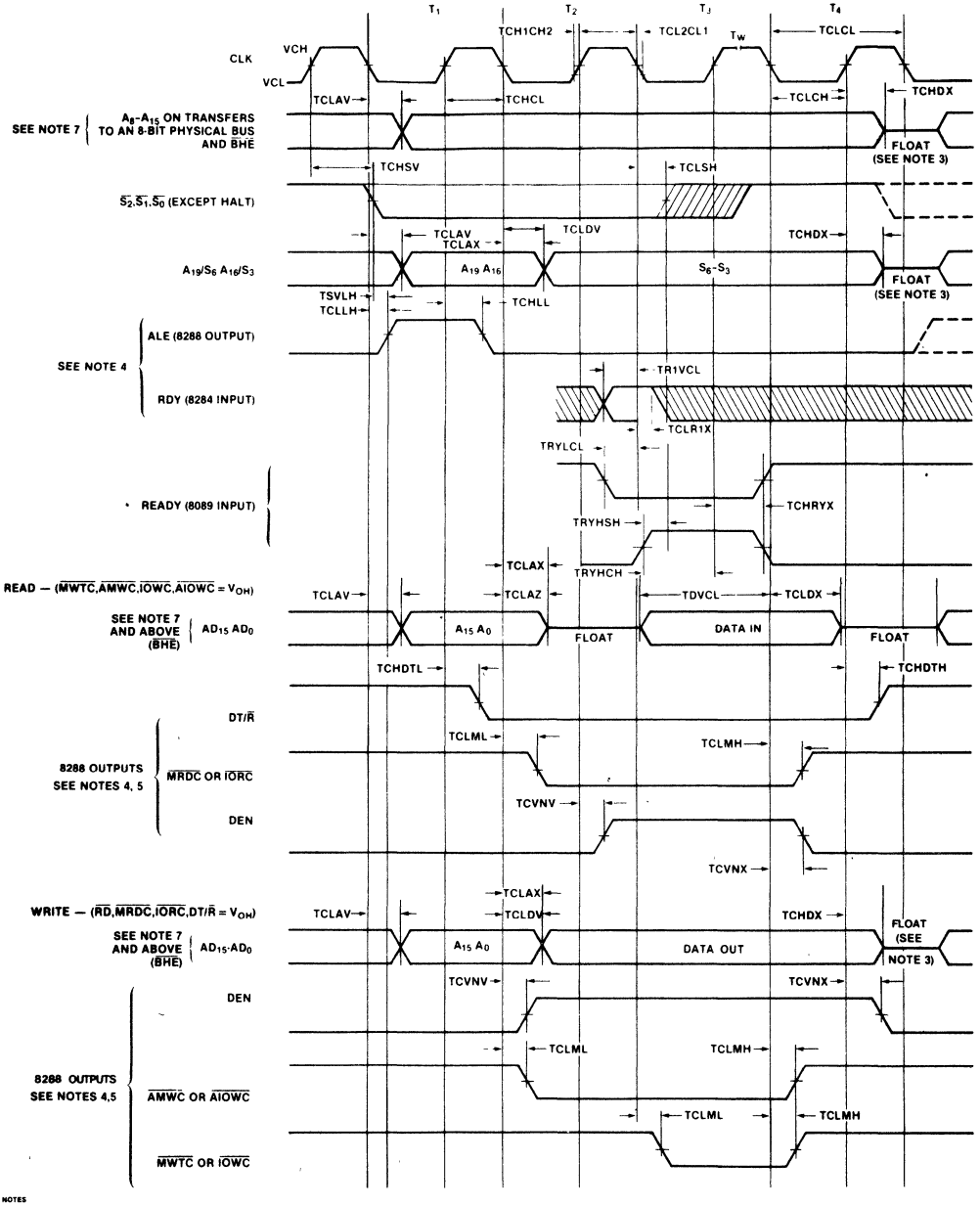
A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

8089 BUS TIMING USING 8288

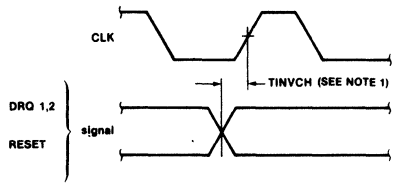


NOTES

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. BHE IS STABLE (NON MULTIPLEXED) FOR ALL TRANSFERS

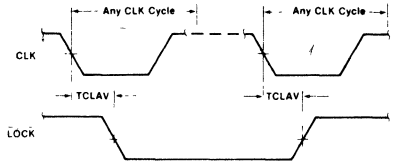
WAVEFORMS (Continued)

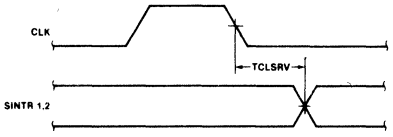
ASYNCHRONOUS SIGNAL RECOGNITION



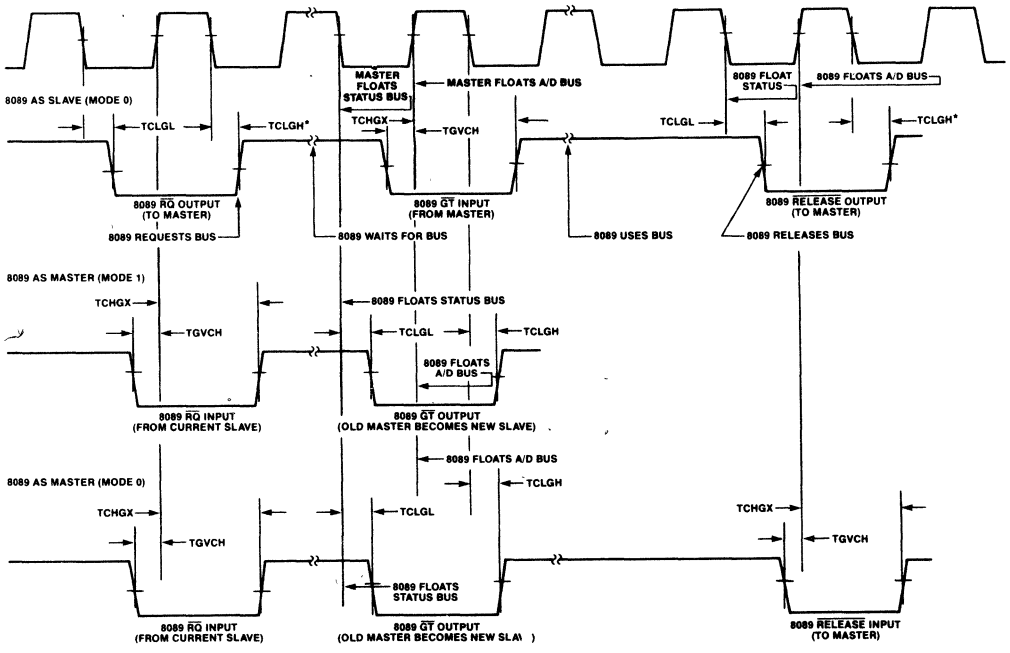
NOTES:
1 SETUP REQUIREMENTS FOR ASYNCHRONOUS SIGNALS ONLY TO GUARANTEE RECOGNITION AT NEXT CLK
2 ALL INPUTS EXCEPT CA ARE LATCHED ON A CLK EDGE THE CA INPUT IS NEGATIVE EDGE TRIGGERED
3 DRQ BECOMING ACTIVE GREATER THAN 30 ns AFTER THE RISING EDGE OF CLK WILL GUARANTEE NON RECOGNITION UNTIL THE NEXT RISING CLOCK EDGE

BUS LOCK SIGNAL TIMING AND SINTR TIMING





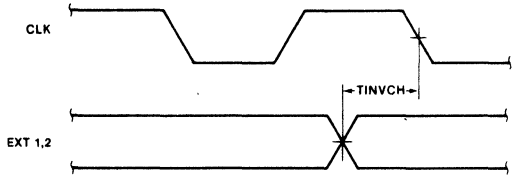
REQUEST/GRANT SEQUENCE TIMINGS



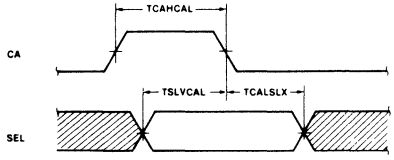
*CPU provides active pull-up.

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—	0 0 0 0 0 A A W | 1 0 0 1 0 0 M M
		Destination—	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	OPCODE 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

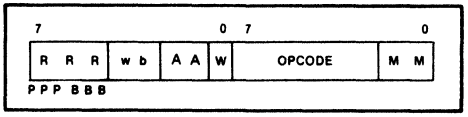
	BIT MANIPULATION	OPCODE 7 0	OPCODE 7 0
SET	Set the Selected Bit	B B B 0 0 A A 0	1 1 1 1 0 1 M M
CLR	Clear the Selected Bit	B B B 0 0 A A 0	1 1 1 1 1 0 M M
	TEST	**OPCODE**	
TSL	Test and Set Lock	0 0 0 1 1 A A 0	1 0 0 1 0 1 M M

Control

	Control	OPCODE 7 0	OPCODE 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

*AA Field 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).

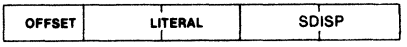
W Width Field

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
8087/8087-2/8087-1

- **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.**
- **Compatible with Proposed IEEE Floating Point Standard**
- **CPU/8087 System Supports 8 Data Types: 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 is compatible with the proposed IEEE Floating Point Standard.

The two-chip numeric data processing systems are referred to as follows;

iAPX 86/20—16-bit 8086 CPU with 8087
iAPX 88/20—8-bit 8088 CPU with 8087
iAPX 186/20—16-bit 80186 CPU with 8087
iAPX 188/20—8-bit 80188 CPU with 8087

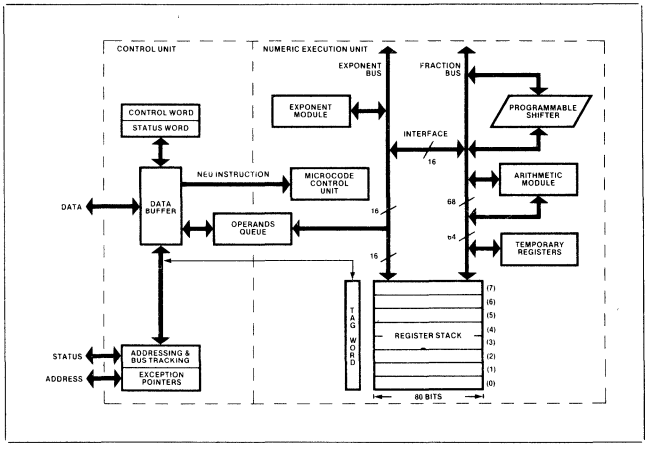


Figure 1. 8087 Block Diagram

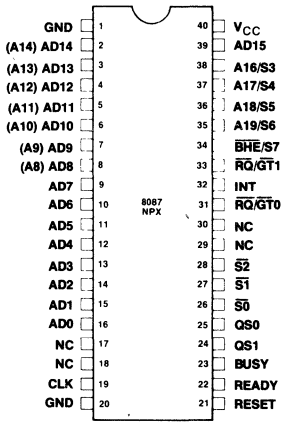


Figure 2. 8087 Pin Configuration

October 1984
ORDER NUMBER 205835-002

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 transferred 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 or iAPX 188/20. 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 Integrated Bus Controller with an 80186/80188 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 186/188 systems, the same sequence applies except $\overline{RQ}/\overline{GT}$ signals are converted to appropriate HOLD, HLDA signals by the 82188 Integrated Bus Controller. This is to conform with iAPX 186/188's HOLD, HLDA bus exchange protocol. Refer to the 82188 data sheet for further information.

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 186/188 systems, the $\overline{RQ}/\overline{GT}1$ line may be connected to the 82188 Integrated 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 Refer to 82188 data sheet for further information.
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 INT0 for iAPX 186/188 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 acknowledgement 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 186/188 systems, RDY is synchronized by the 82188 Integrated 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. For iAPX 186 or iAPX 188 systems, also refer to the 82188 Integrated Bus Controller data sheet.

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 186/188 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 186/188 systems, the queue status signals of the iAPX 186/188 are synchronized to 8087 requirements by the 82188 Integrated 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 can interrupt the CPU when it detects an error or exception. The

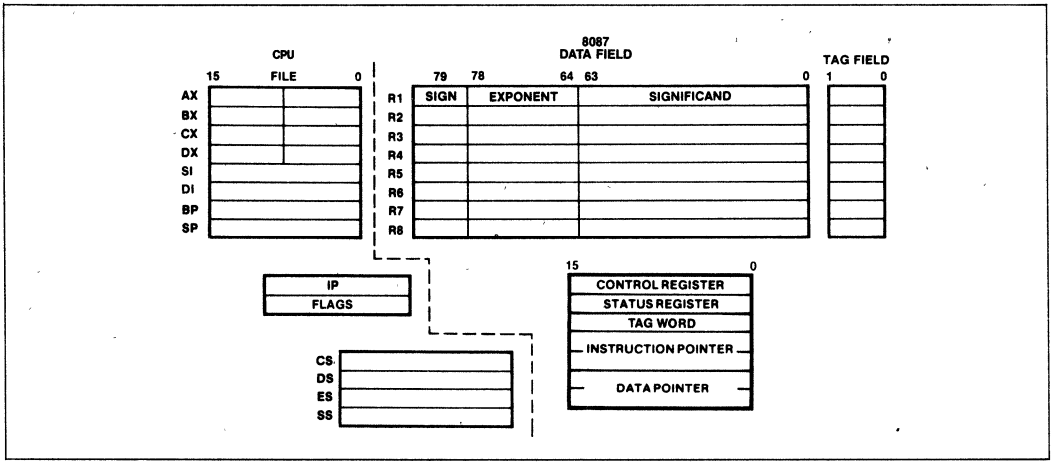


Figure 3. CPU+8087 Architecture

8087's interrupt request line is typically routed to the CPU through an 8259A Programmable Interrupt Controller for 8086, 8088 systems and INT0 for iAPX 186/188.

The 8087 uses one of the request/grant lines of the iAPX 86/88 architecture (typically $\overline{RQ}/\overline{GT0}$) 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 $\overline{RQ}/\overline{GT1}$ 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 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 186/188 systems, $\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$ are connected to the corresponding inputs of the 82188 Integrated Bus Controller. Because the iAPX 186/188 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 functions of the 82188 IBC is to provide this translation. $\overline{RQ}/\overline{GT0}$ is translated to HOLD, HLDA signals which are then directly connected to the iAPX 186/188. The $\overline{RQ}/\overline{GT1}$ line is also translated into HOLD, HLDA signals (referred to as SYSHOLD, SYSHLDA signals) by the 82188 IBC. This allows a third processor (using a HOLD, HLDA bus exchange protocol) to gain control of the bus.

Unlike an iAPX 86/20 system, $\overline{RQ}/\overline{GT1}$ is only used when the 8087 has bus control. If the third processor requests the bus when the current bus master is the iAPX 186/188, the 82188 IBC will directly pass the request onto the iAPX 186/188 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 iAPX 186/188 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 7 0)
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: $(-1)^S(D_{17} \quad D_0)$

Real: $(-1)^S(2^{E-BIAS})(F_0 \cdot F_1 \cdot .)$

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 Integrated 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 performance is up to 100 times that of an iAPX 86/10 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 1, 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

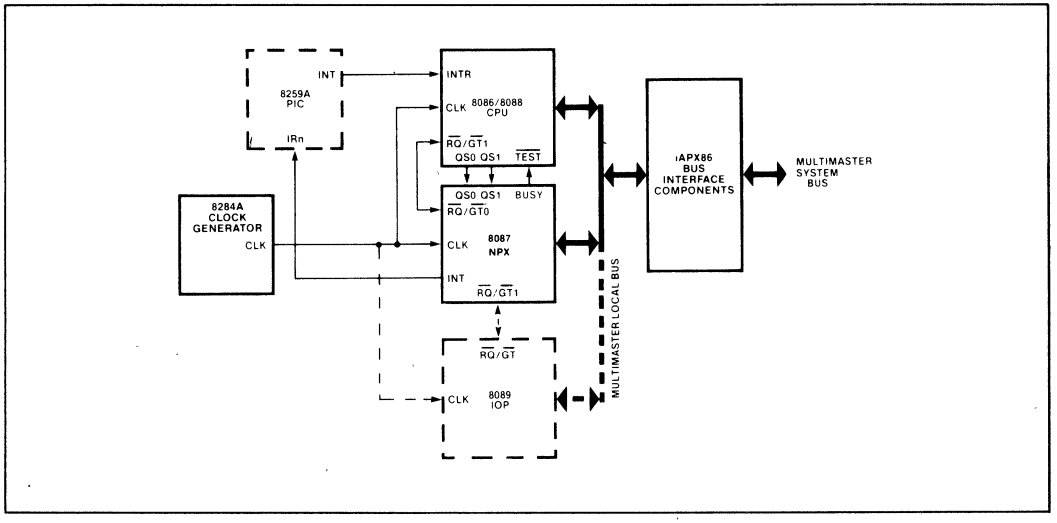
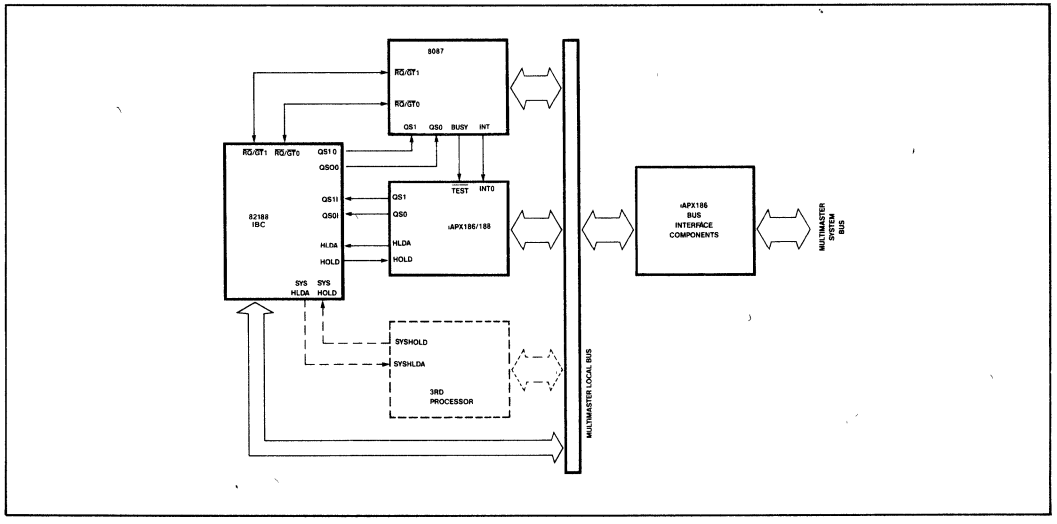


Figure 5. iAPX 186/20, 188/20 System Configuration



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 CPU+8087 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.

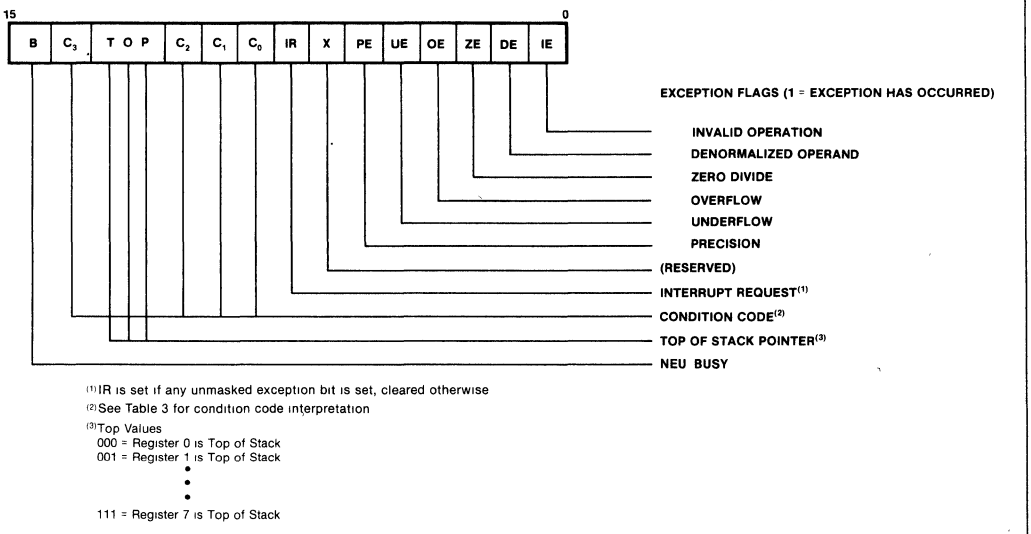


Figure 6. 8087 Status Word

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.

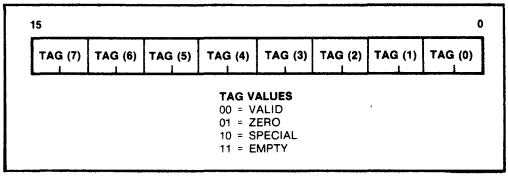


Figure 7. 8087 Tag Word

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.

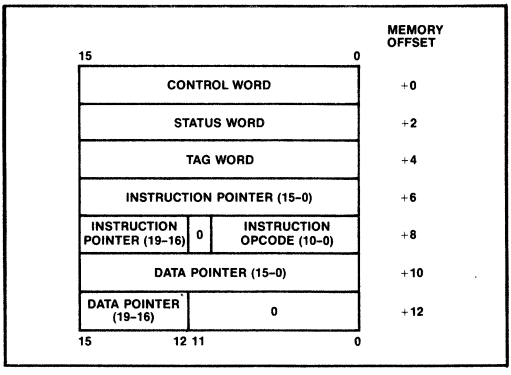


Figure 8. 8087 Instruction and Data Pointer Image in Memory

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.

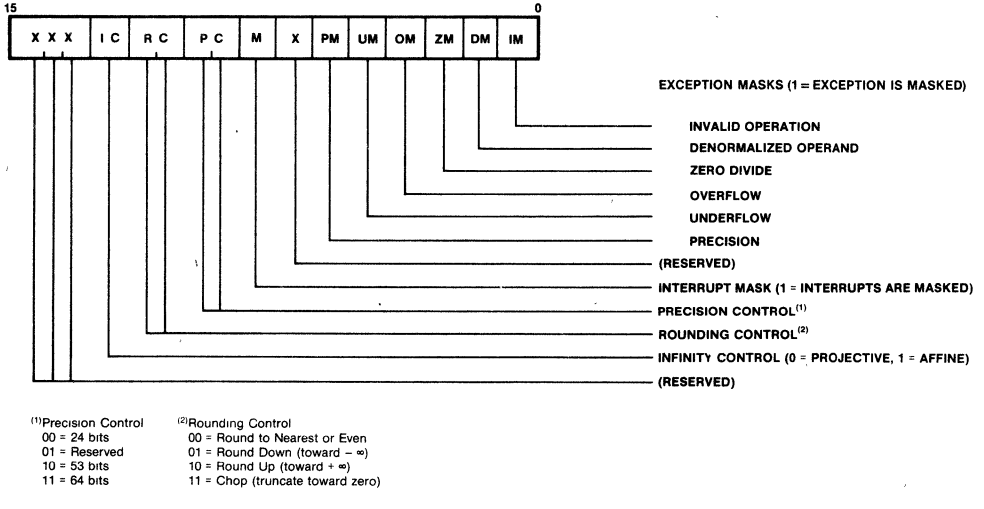


Figure 9. 8087 Control Word

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		8087-1 (Preliminary: See Note 7)			
Symbol	Parameter	Min.	Max.	Min.	Max.	Min.	Max.	Units	Test Conditions
TCLCL	CLK Cycle Period	200	500	125	500	100	500	ns	
TCLCH	CLK Low Time	118		68		53		ns	
TCHCL	CLK High Time	69		44		39		ns	
TCH1CH2	CLK Rise Time		10		10		15	ns	From 1.0V to 3.5V
TCL2CL2	CLK Fall Time		10		10		15	ns	From 3.5V to 1.0V
TDVCL	Data in Setup Time	30		20		15		ns	
TCLDX	Data in Hold Time	10		10		10		ns	
TRYHCH	READY Setup Time	118		68		53		ns	
TCHRYX	READY Hold Time	30		20		5		ns	
TRYLCL	READY Inactive to CLK**	- 8		- 8		-10		ns	
TGVCH	RQ/GT Setup Time	30		15		8		ns	
TCHGX	RQ/GT Hold Time	40		30		20		ns	
TQVCL	QS0-1 Setup Time	30		30		10		ns	
TCLQX	QS0-1 Hold Time	10		10		5		ns	
TSACH	Status Active Setup Time	30		30		30		ns	
TSNCL	Status Inactive Setup Time	30		30		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		15	ns	From 2.0V to 0.8V

**See Note 6

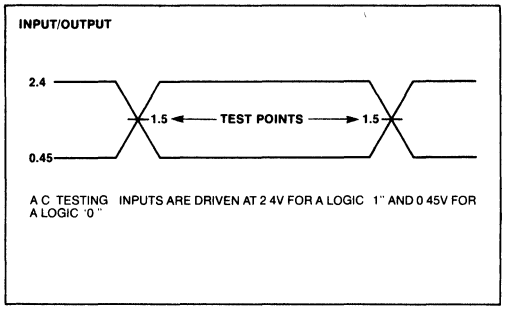
205835-003

A.C. CHARACTERISTICS (Continued)

TIMING RESPONSES		8087		8087-2		8087-1 (Preliminary: See Note 7)			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
TCLML	Command Active Delay (See Notes 1,2)	10/0	35/70	10/0	35/70	10/0	35/70	ns	C_L = 20 - 100pF for all 8087 Outputs (in addition to 8087 self-load)
TCLMH	Command Inactive Delay (See Notes 1,2)	10/0	35/55	10/0	35/55	10/0	35/70	ns	
TRYHSH	Ready Active to Status Passive (See Note 5)		110		65		45	ns	
TCHSV	Status Active Delay	10	110	10	60	10	45	ns	
TCLSH	Status Inactive Delay	10	130	10	70	10	55	ns	
TCLAV	Address Valid Delay	10	110	10	60	10	55	ns	
TCLAX	Address Hold Time	10		10		10		ns	
TCLAZ	Address Float Delay	TCLAX	80	TCLAX	50	TCLAX	45	ns	
TSVLH	Status Valid to ALE High (See Notes 1,2)		15/30		15/30		15/30	ns	
TCLLH	CLK Low to ALE Valid (See Notes 1,2)		15/30		15/30		15/30	ns	
TCHLL	ALE Inactive Delay (See Notes 1,2)		15/30		15/30		15/30	ns	
TCLDV	Data Valid Delay	10	110	10	60	10	50	ns	
TCHDX	Data Hold Time	10		10		10	45	ns	
TCVNV	Control Active Delay (See Notes 1,3)	5	45	5	45	5	45	ns	
TCVNX	Control Inactive Delay (See Notes 1,3)	10	45	10	45	10	45	ns	
TCHBV	BUSY and INT Valid Delay	10	150	10	85	10	65	ns	
TCHDTL	Direction Control Active Delay (See Notes 1,3)		50		50		50	ns	
TCHDTH	Direction Control Inactive Delay (See Notes 1,3)		30		30		30	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay (See Notes 1,4)	0	30	0	30	0	30	ns	
TCLDTV	DT/$\overline{R}$ Active Delay (See Notes 1,4)	0	55	0	55	0	55	ns	
TCHDNV	$\overline{DEN}$ Active Delay (See Notes 1,4)	0	55	0	55	0	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay (See Notes 1,4)	5	55	5	55	5	55	ns	
TCLGL	RQ/GT Active Delay	0	85	0	50	0	41	ns	C_L=40pF (in addition to 8087 self-load)
TCLGH	RQ/GT Inactive Delay	0	85	0	50	0	45	ns	
TOLOH	Output Rise Time		20		20		15	ns	From 0.8V to 2.0V
TOHOL	Output Fall Time		12		12		12	ns	From 2.0V to 0.8V

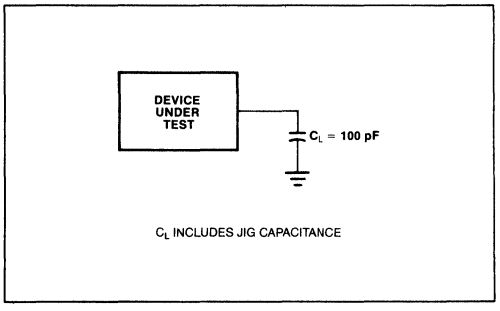
NOTES:
1. Signal at 8284A, 8288, or 82188 shown for reference only.
2. 8288 timing/82188 timing
3. 8288 timing
4. 82188 timing
5. Applies only to T_3 and wait states
6. Applies only to T_2 state (8ns into T_3)
7. IMPORTANT SYSTEM CONSIDERATION: Some 8087-1 timing parameters are constrained relative to the corresponding 8086-1 specifications. Therefore, 8086-1 systems incorporating the 8087-1 should be designed with the 8087-1 specifications.

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"

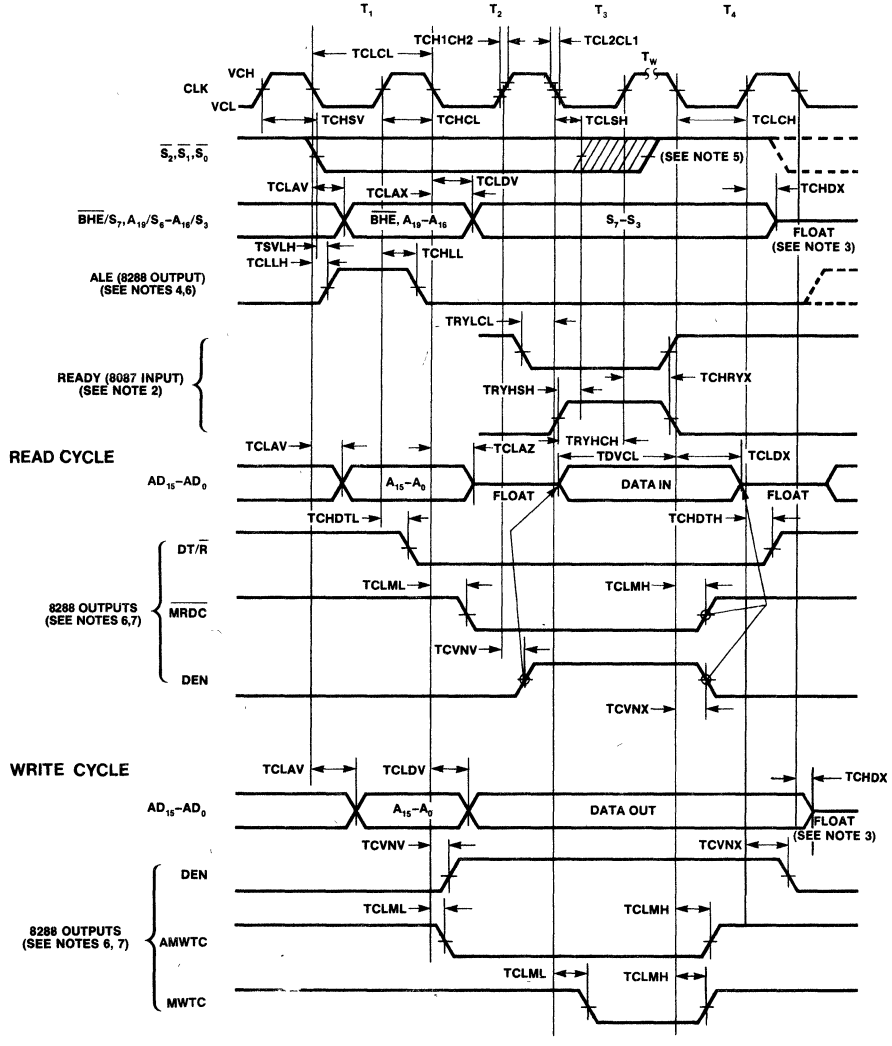
A.C. TESTING LOAD CIRCUIT



C_L INCLUDES JIG CAPACITANCE

WAVEFORMS

MASTER MODE (with 8288 references)

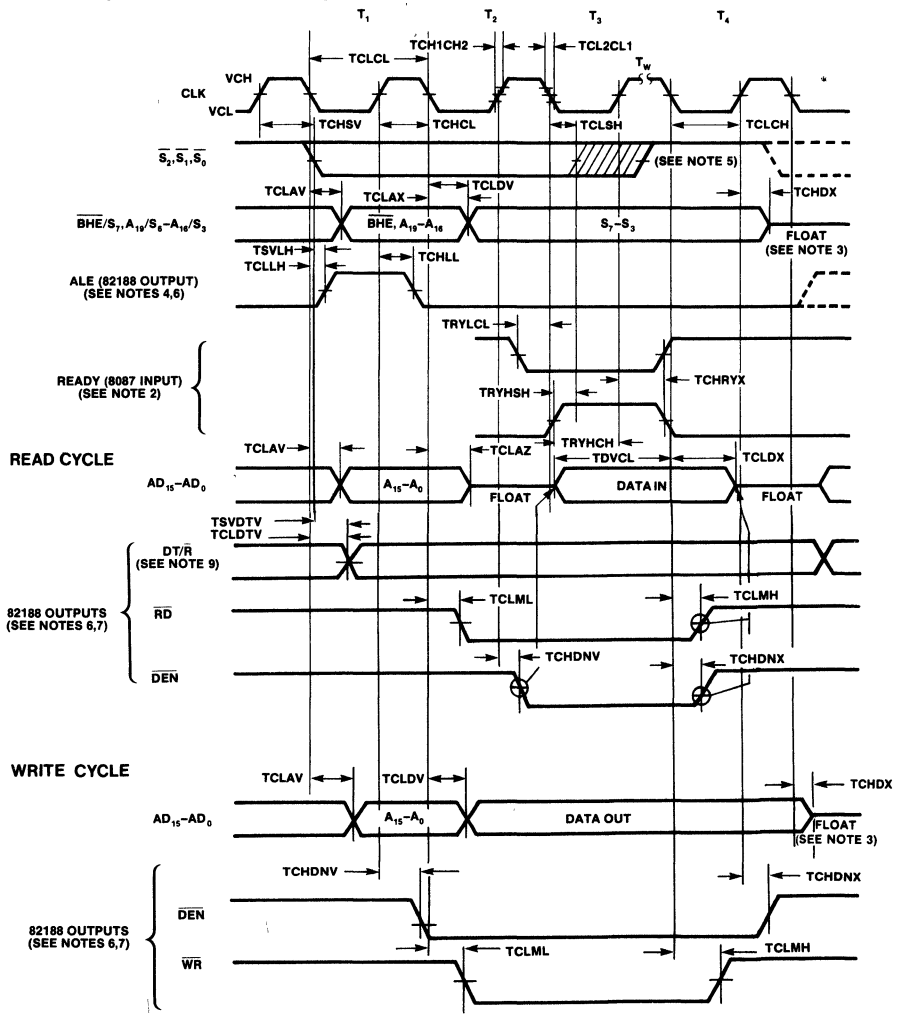


NOTES:

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

WAVEFORMS (Continued)

MASTER MODE (with 82188 references)

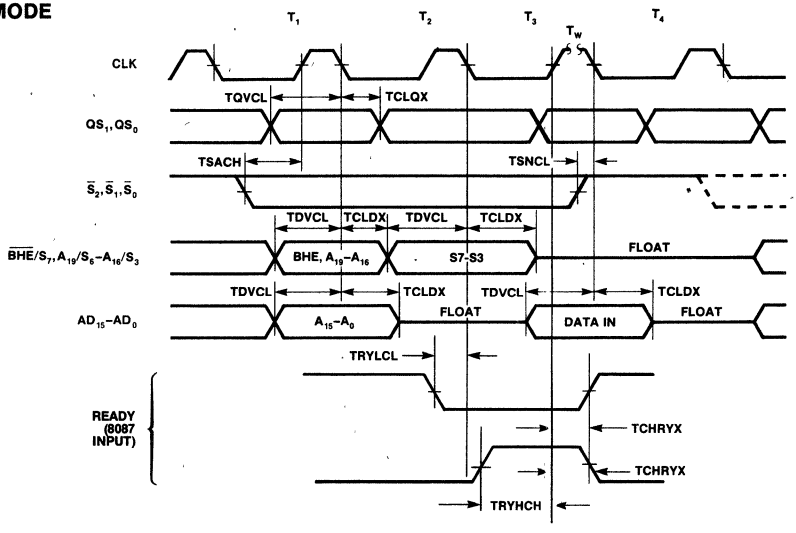


NOTES:

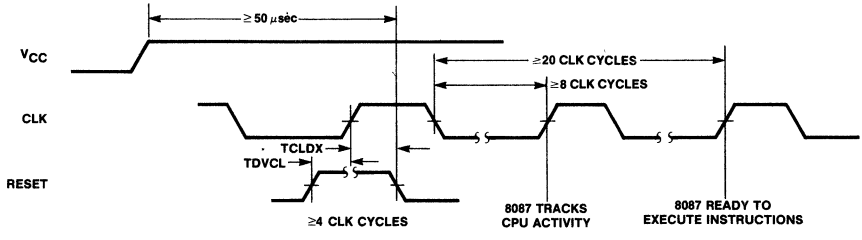
1 ALL SIGNALS SWITCH BETWEEN V_{OL} AND V_{OH} UNLESS OTHERWISE SPECIFIED.
2 READY IS SAMPLED NEAR THE END OF T_2, T_3 AND T_W TO DETERMINE IF T_W MACHINE STATES ARE TO BE INSERTED
3 THE LOCAL BUS FLOATS ONLY IF THE 8087 IS RETURNING CONTROL TO THE 80186/80188
4 ALE RISES AT LATER OF (TSVLH, TCLLH)
5 STATUS INACTIVE IN STATE JUST PRIOR TO T_4
6 SIGNALS AT 8284A OR 82188 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 DT/$\overline{R}$ BECOMES VALID AT THE LATER OF (TSVDTV, TCLDTV)

WAVEFORMS (Continued)

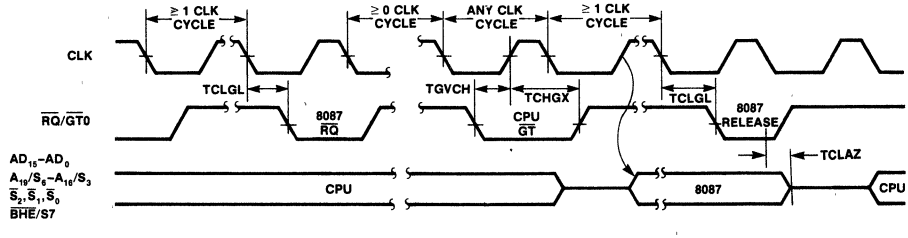
PASSIVE MODE



RESET TIMING



REQUEST/GRANT$_0$ TIMING

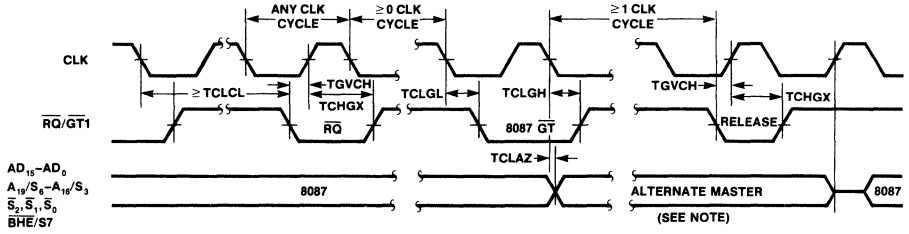


NOTE THE CPU PROVIDES ACTIVE PULLUP OF $\overline{RQ}/\overline{GT}0$, SEE TCLGH SPEC

205835-003

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

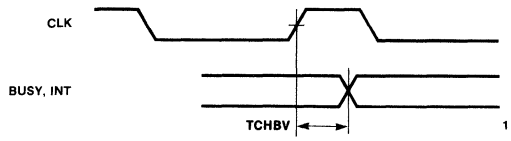


Table 5. 8087 Extensions to the 86/186 Instructions Sets

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:** $0 \leq ST(0) < ST(1) < +\infty$

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.

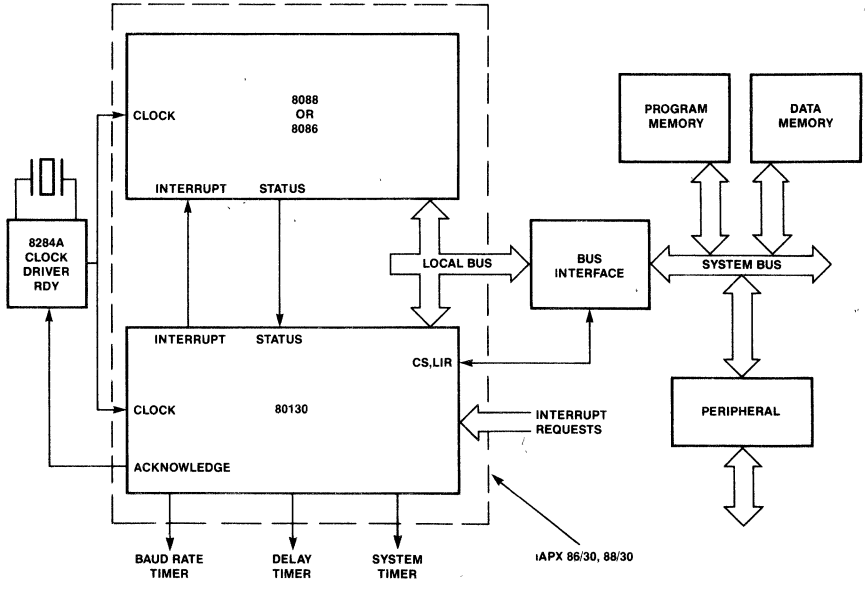


Figure 1. iAPX 86/30, 88/30 Block Diagram

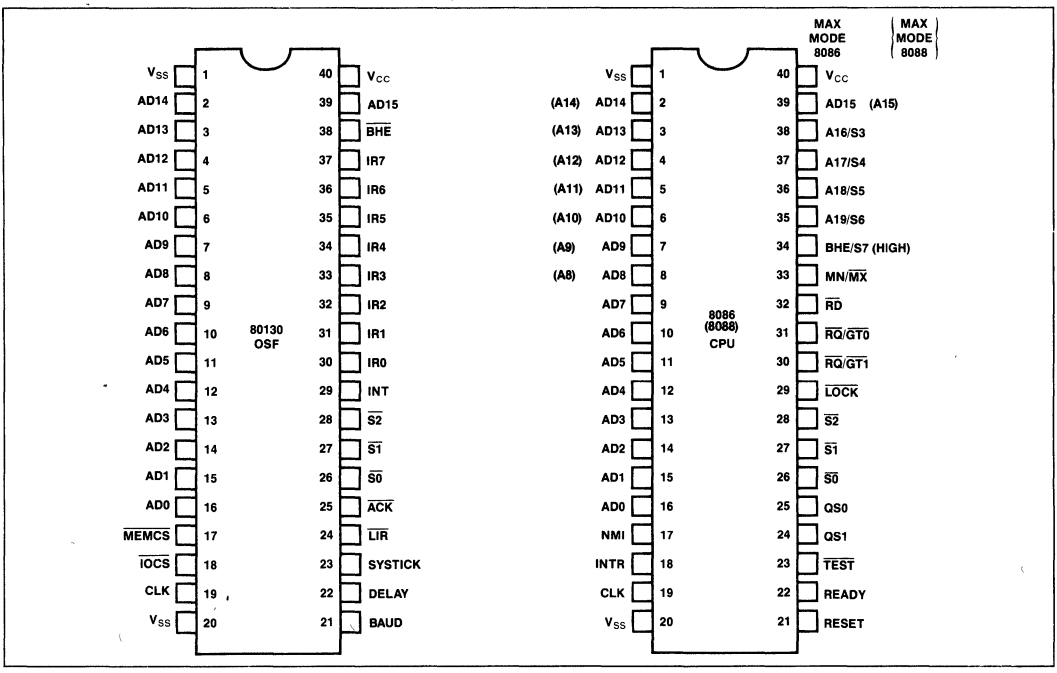


Figure 2. iAPX 86/30, 88/30 Pin Configuration

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

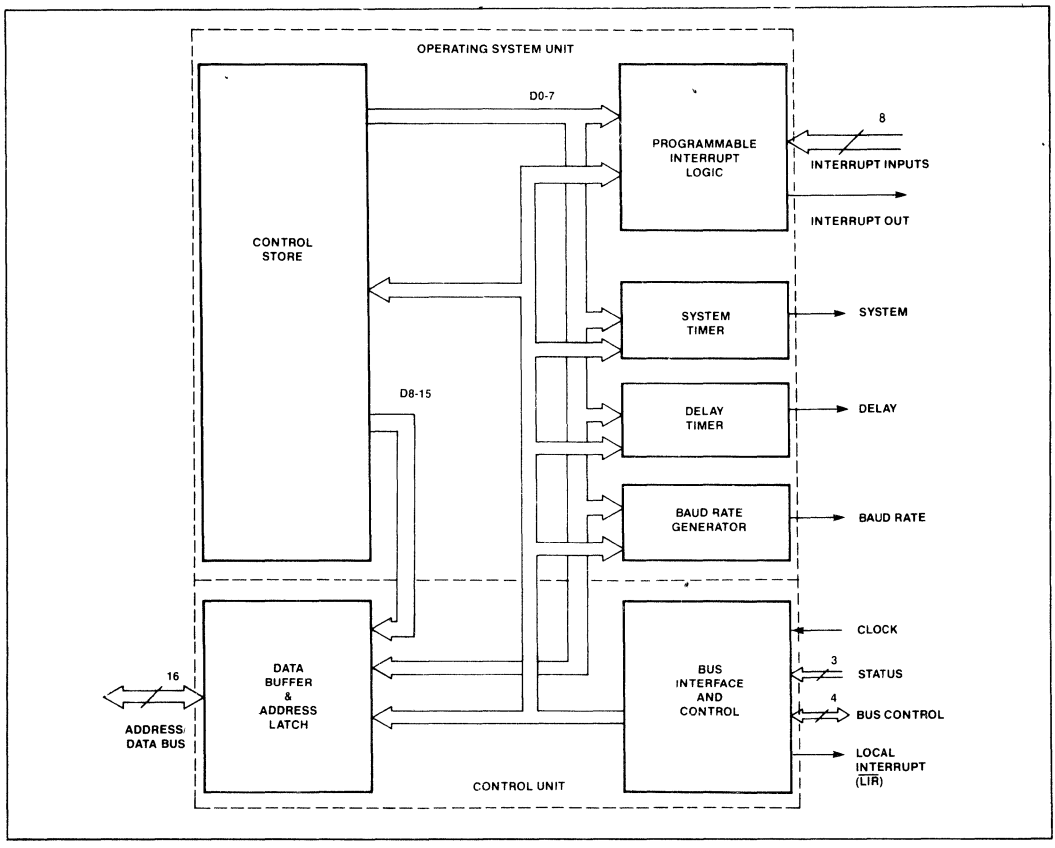


Figure 3. OSF Internal Block Diagram

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

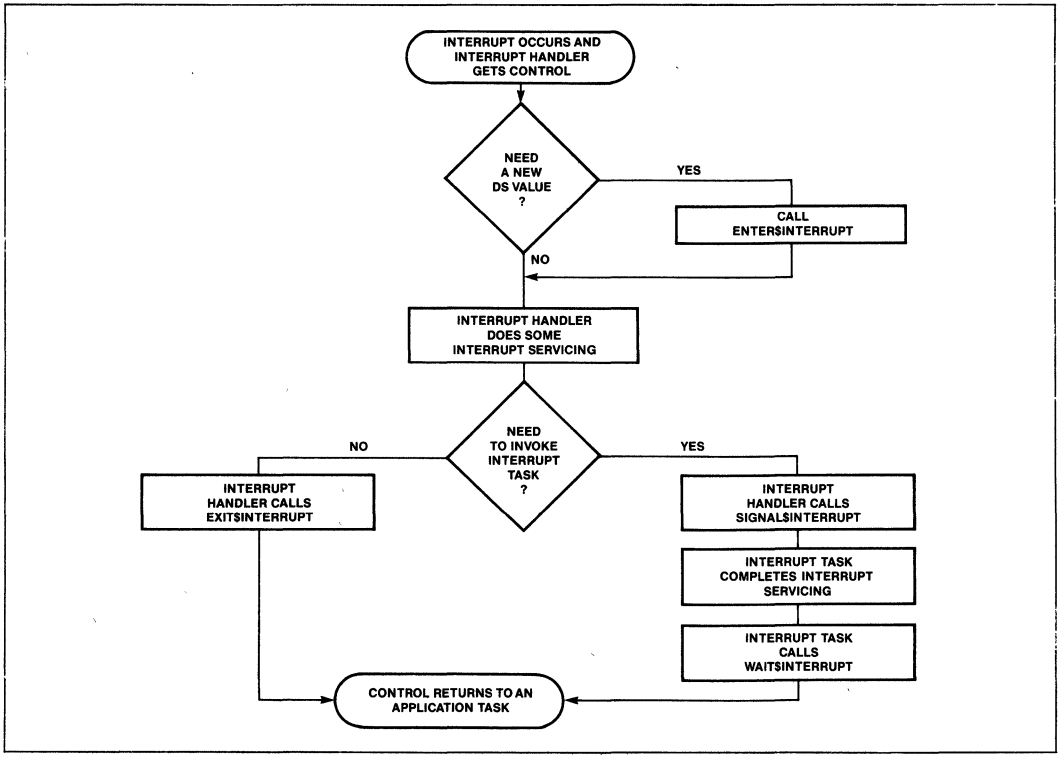


Figure 6. Interrupt Handling Flowchart

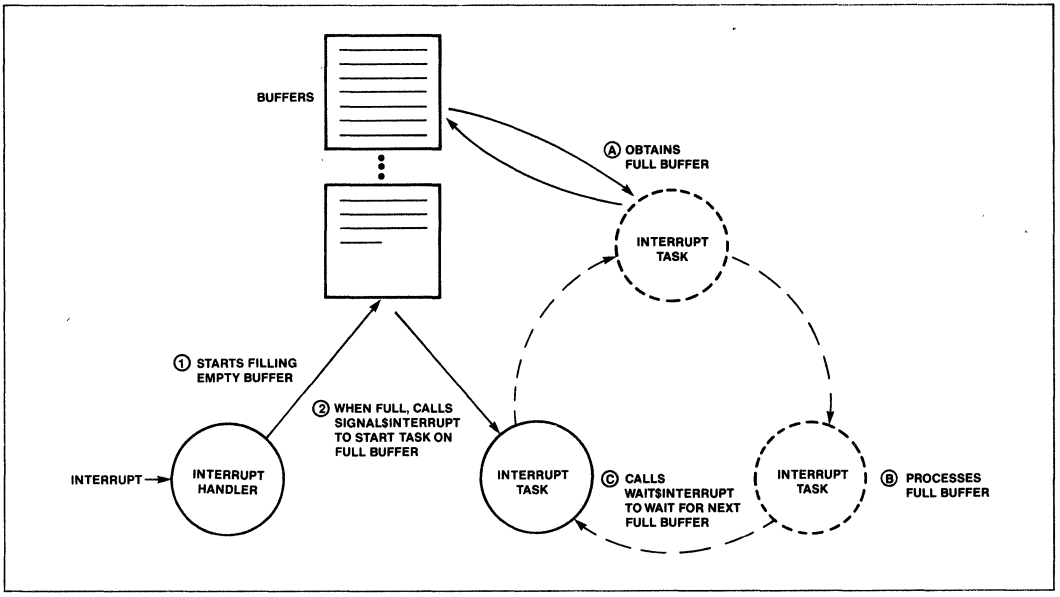


Figure 7. Multiple Buffer Example

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

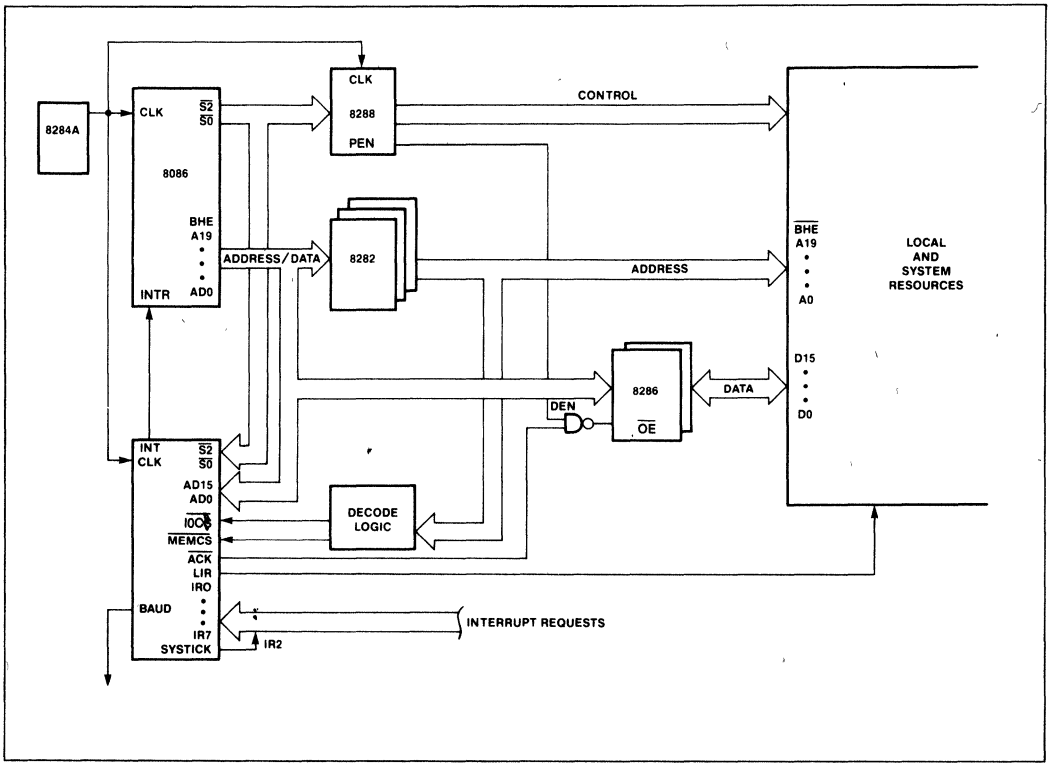


Figure 8. Typical OSP Configuration

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)

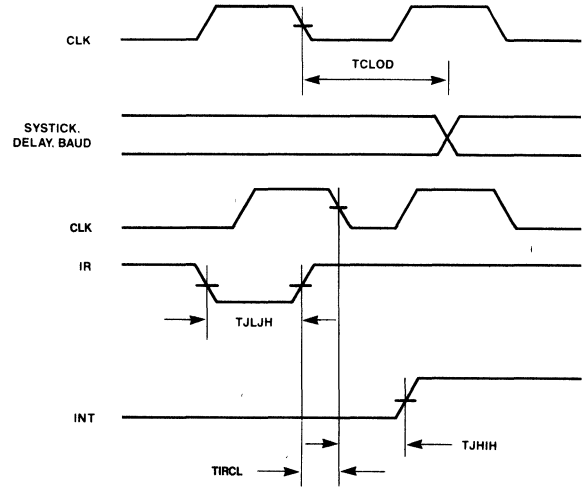
		80130		80130-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
T_{CLCL}	CLK Cycle Period	200	–	125	–	ns	
T_{CLCH}	CLK Low Time	90	–	55	–	ns	
T_{CHCL}	CLK High Time	69	2000	44	2000	ns	
T_{SVCH}	Status Active Setup Time	80	–	65	–	ns	
T_{CHSV}	Status Inactive Hold Time	10	–	10	–	ns	
T_{SHCL}	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		20		ns	

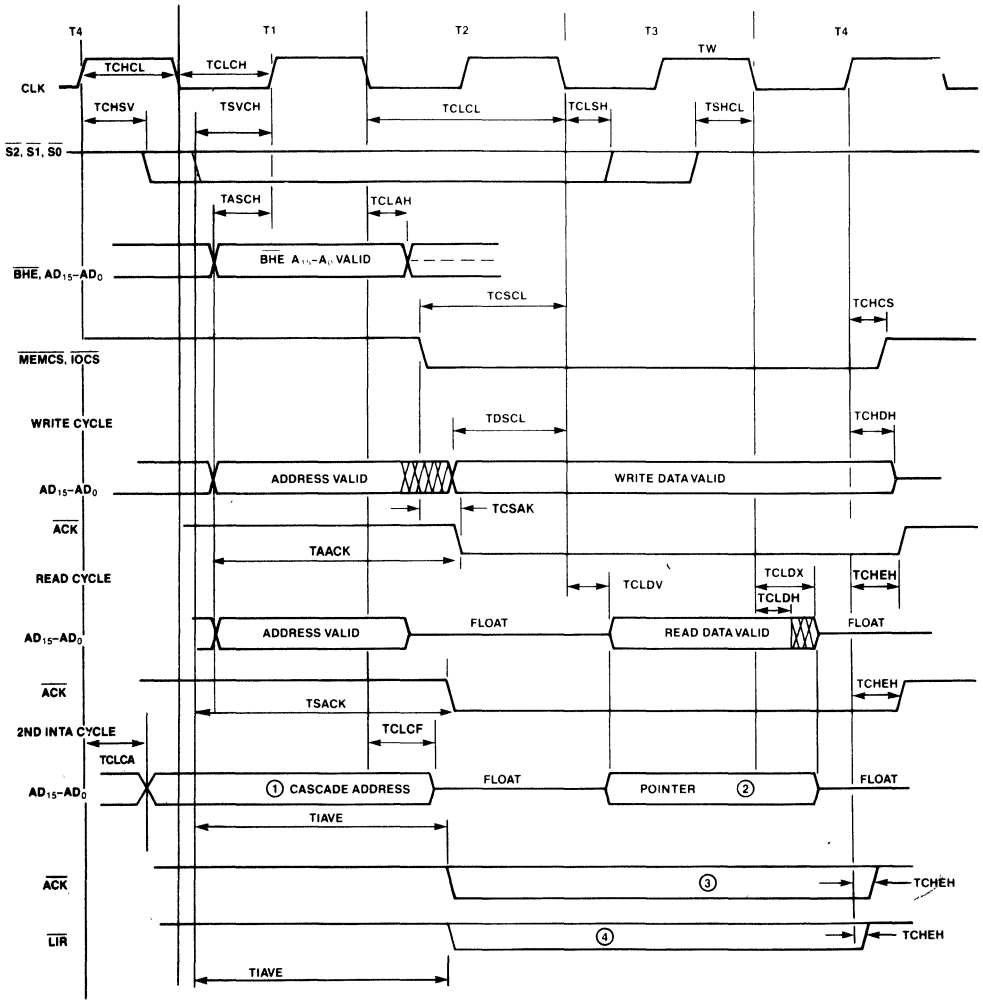
WAVEFORMS

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

ADVANCE INFORMATION

80150/80150-2
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, 7220 Bubble Memory Controller**
- **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

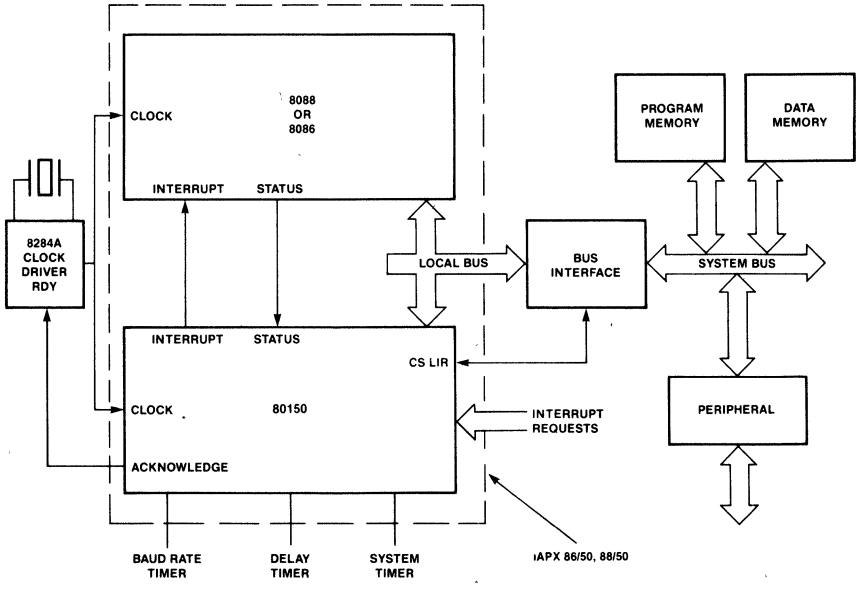


Figure 1. iAPX 86/50, 88/50 Block Diagram

SEPTEMBER 1982
ORDER NUMBER: 210705-002

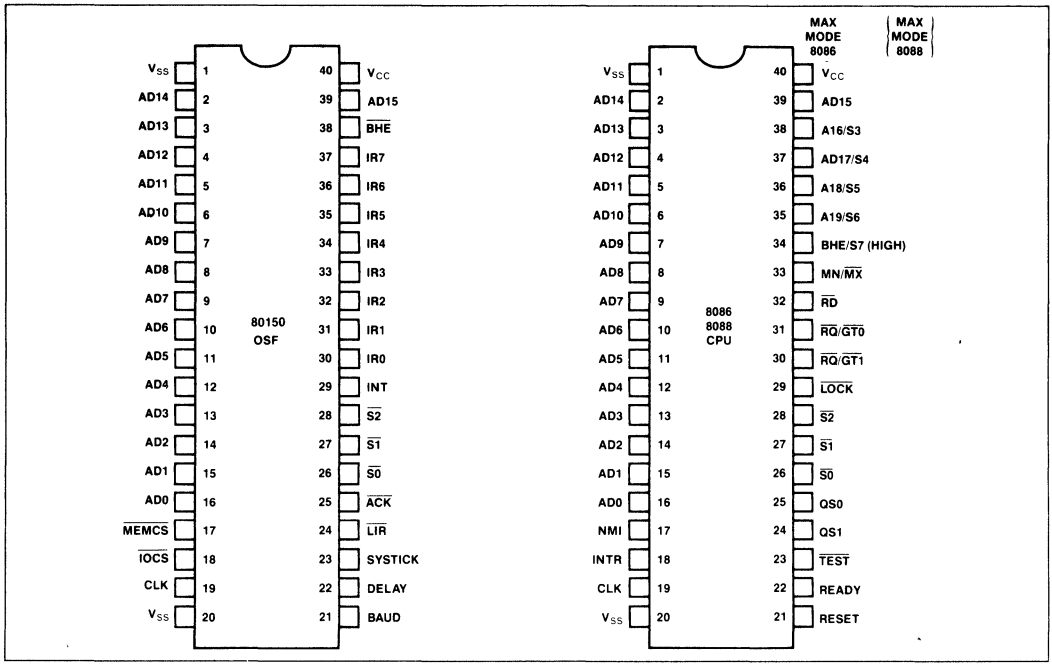


Figure 2. iAPX 86/50, 88/50 Pin Configuration

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

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 80150 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

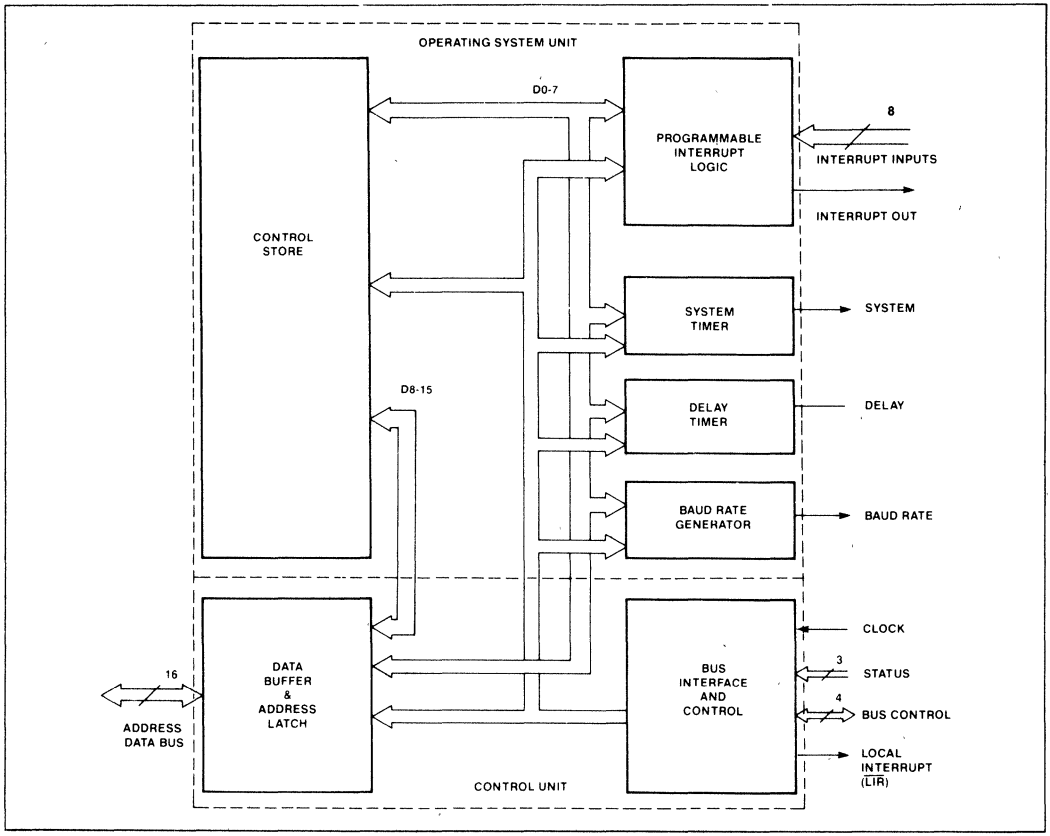


Figure 3. 80150 Internal Block Diagram

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
7220	Bubble Memory Controller

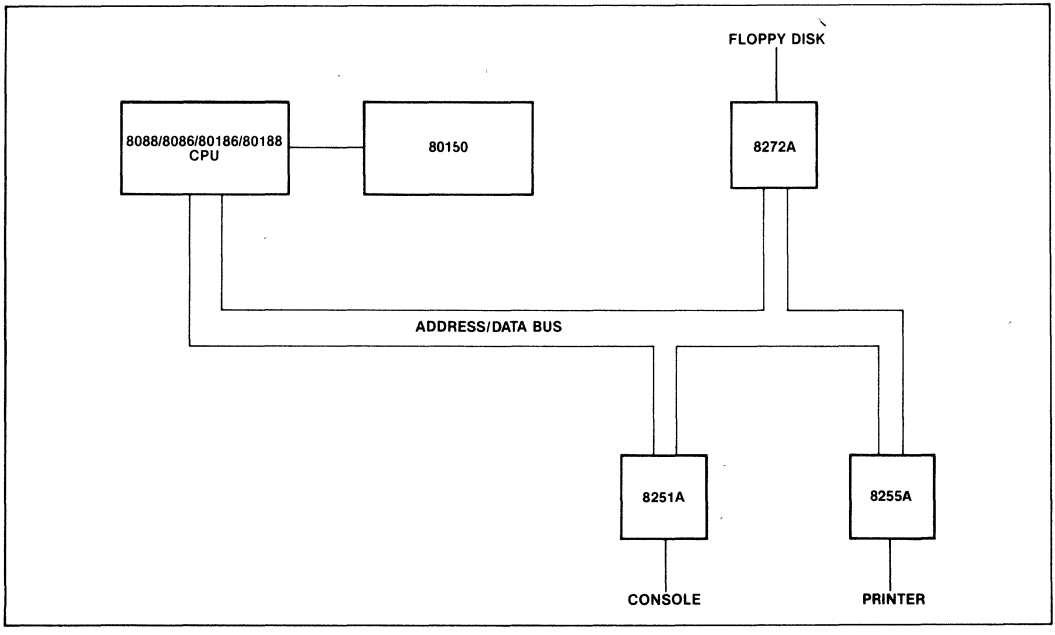


Figure 4. Predefined Configuration

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-

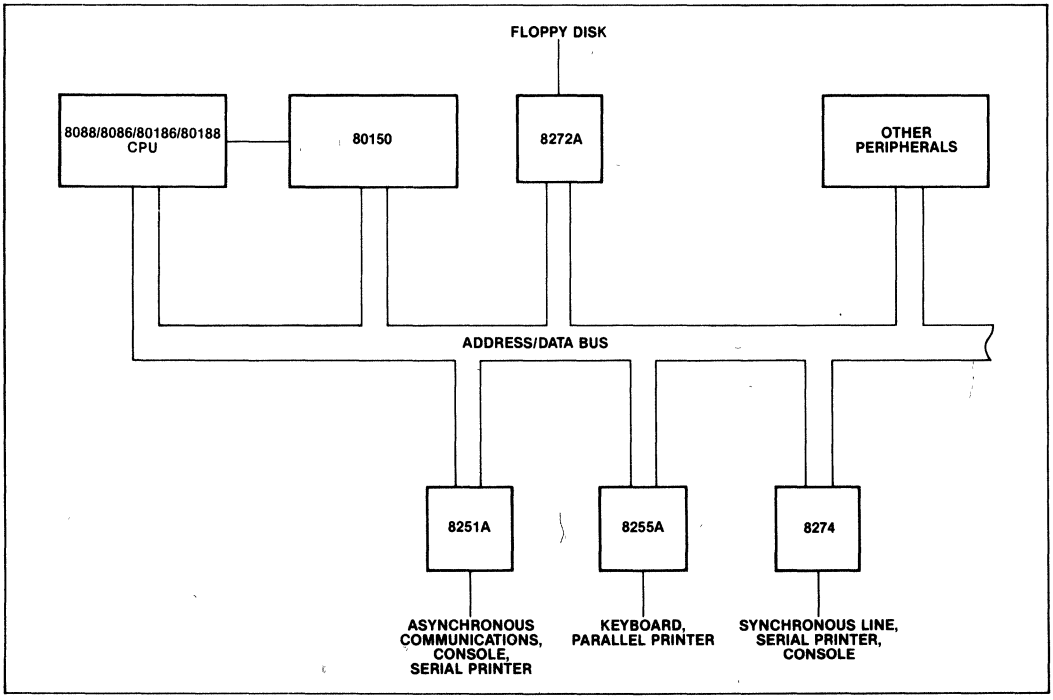


Figure 5. OEM Configurable System

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 and Bubble Memories

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 80150 also contains software drivers for 7220 bubble memory controller. Use of a bubble memory board as a substitute for one floppy disk drive is directly supported.

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,

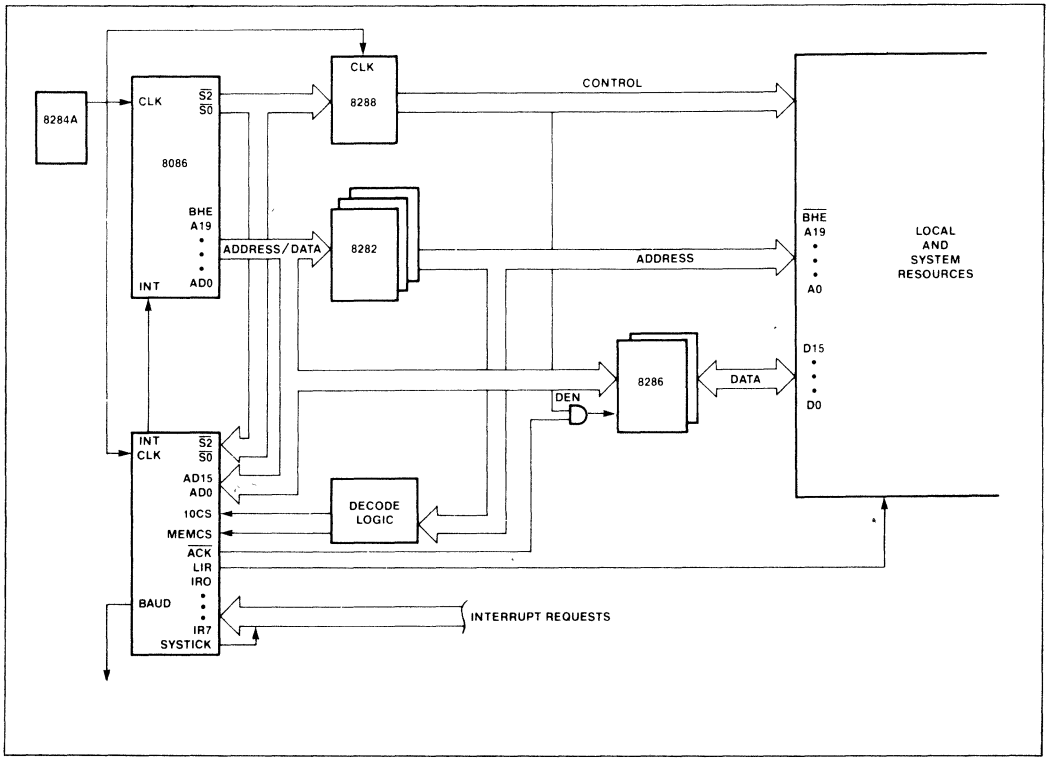


Figure 6. Typical OSP Configuration

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)

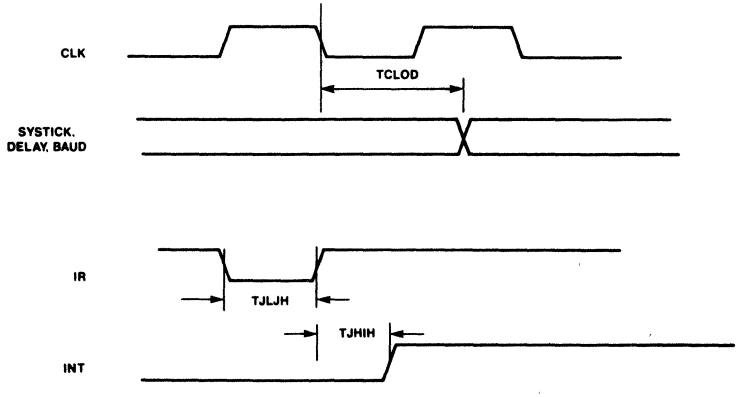
		80150		80150-2			
Symbol	Parameter	Min.	Max.	Min.	Max.	Units	Test Conditions
T_{CLCL}	CLK Cycle Period	200	–	125	–	ns	
T_{CLCH}	CLK Low Time	90	–	55	–	ns	
T_{CHCL}	CLK High Time	69	2000	44	2000	ns	
T_{SVCH}	Status Active Setup Time	80	–	65	–	ns	
T_{CHSV}	Status Inactive Hold Time	10	–	10	–	ns	
T_{SHCL}	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	

A.C. CHARACTERISTIC (Continued)

		80150		80150-2			
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		20		ns	

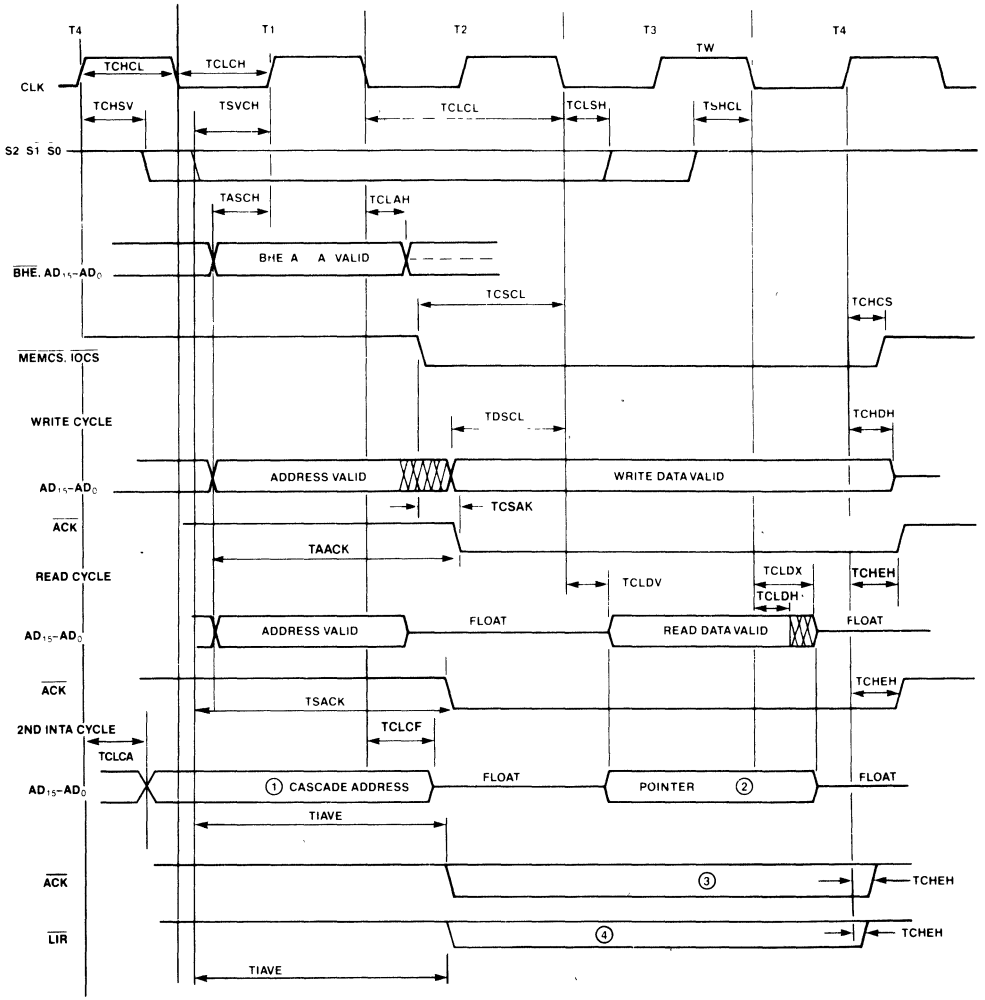
WAVEFORMS

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 THF 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.

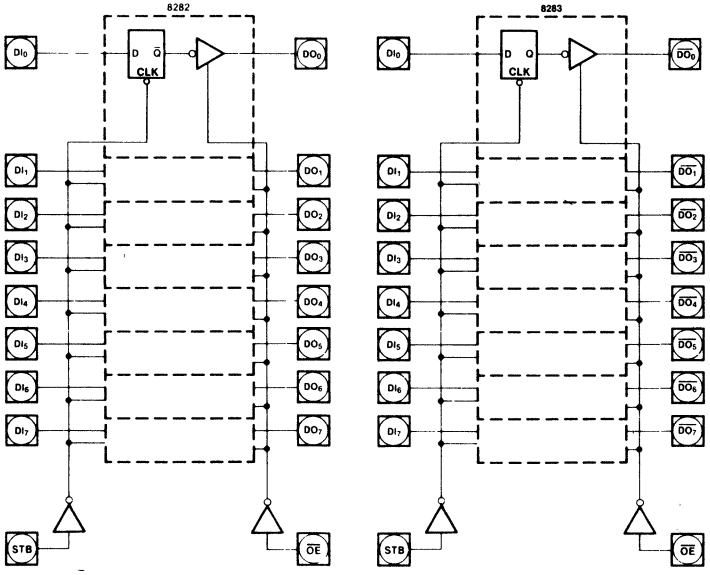


Figure 1. Logic Diagrams

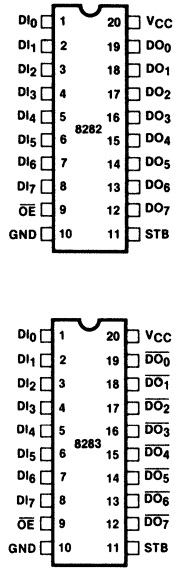


Figure 2. Pin Configurations

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

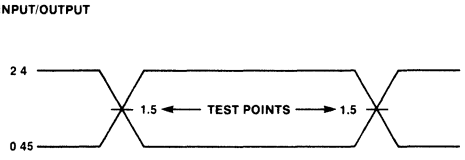
NOTE:
1. Output Loading I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF.*

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

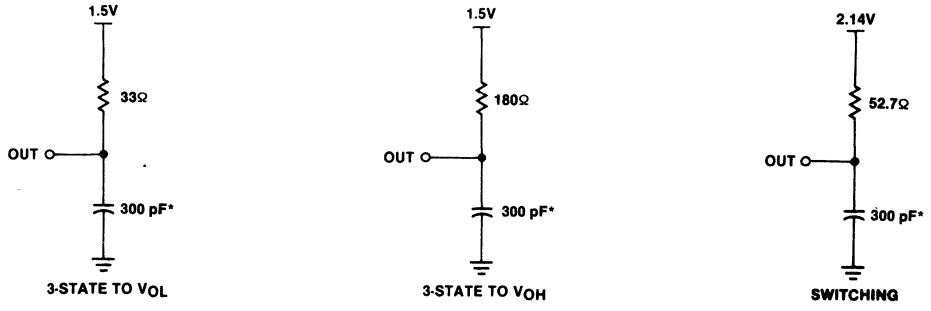
NOTE:
*C_L = 200 pF for plastic 8282/8283.
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TSHOV = 45, 55; TEHOZ = 25; TELOV = 50.

A.C. TESTING INPUT, OUTPUT WAVEFORM



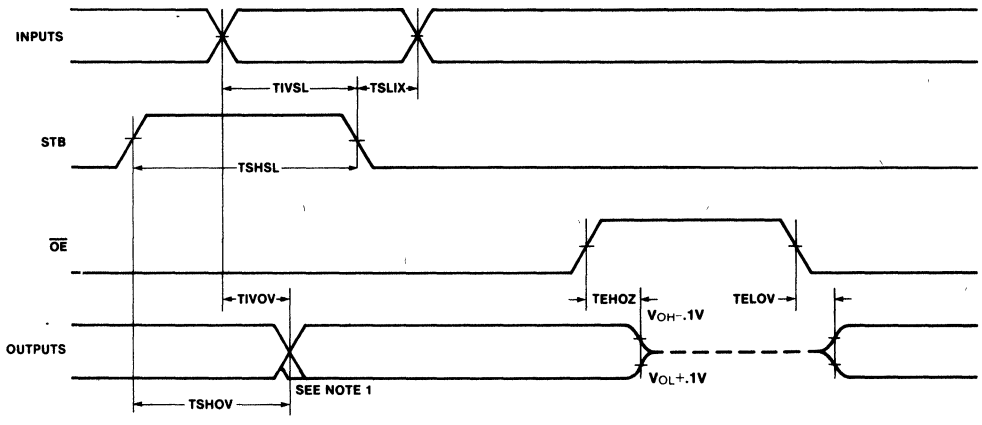
A C TESTING: INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0" TIMING MEASUREMENTS ARE MADE AT 1 5V FOR BOTH A LOGIC "1" AND "0" INPUT RISE AND FALL TIMES ARE MEASURED FROM 0 8V TO 2 0V AND ARE DRIVEN AT 5ns ± 2ns

OUTPUT TEST LOAD CIRCUITS

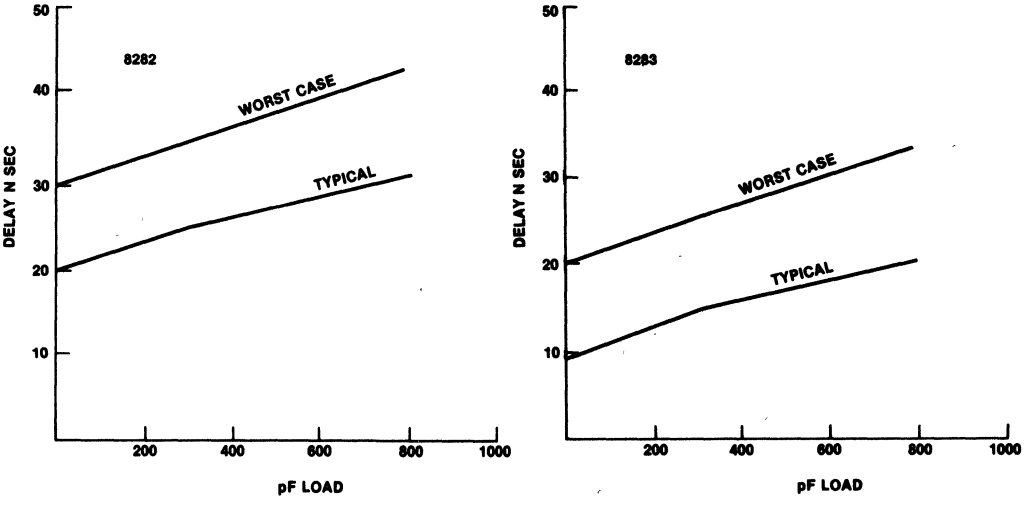


*200 pF for plastic 8282/8283.

WAVEFORMS



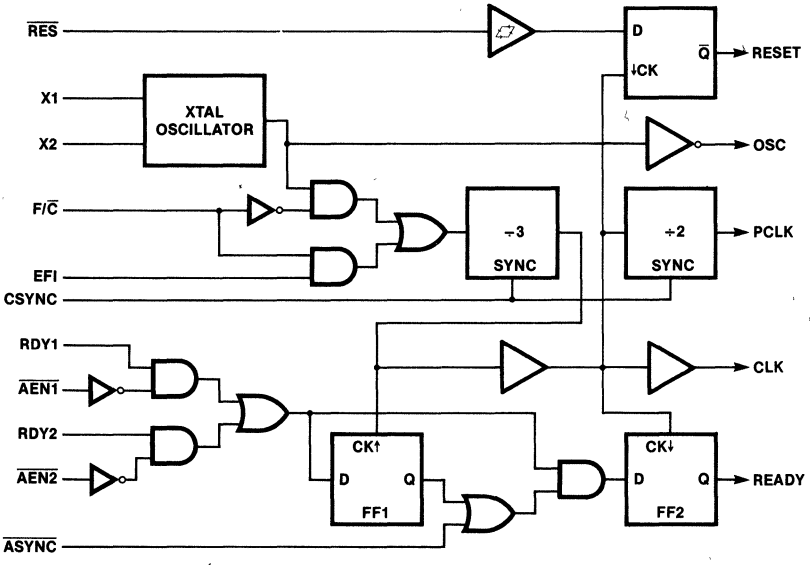
NOTE: 1. OUTPUT MAY BE MOMENTARILY INVALID FOLLOWING THE HIGH GOING STB TRANSITION.
2. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.



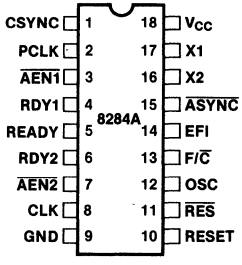
Output Delay vs. Capacitance

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 1/3 of the crystal or EFI input frequency and a 1/3 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 1/2 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 ($R_1 = R_2 = 510\ \Omega$) 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 $\geq$ 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.

If EFI is used and no crystal is connected, it is recommended that X_1 or X_2 should be tied to V_{CC} through a 510Ω resistor to prevent the oscillator from free running which might produce HF noise and additional I_{CC} current.

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.

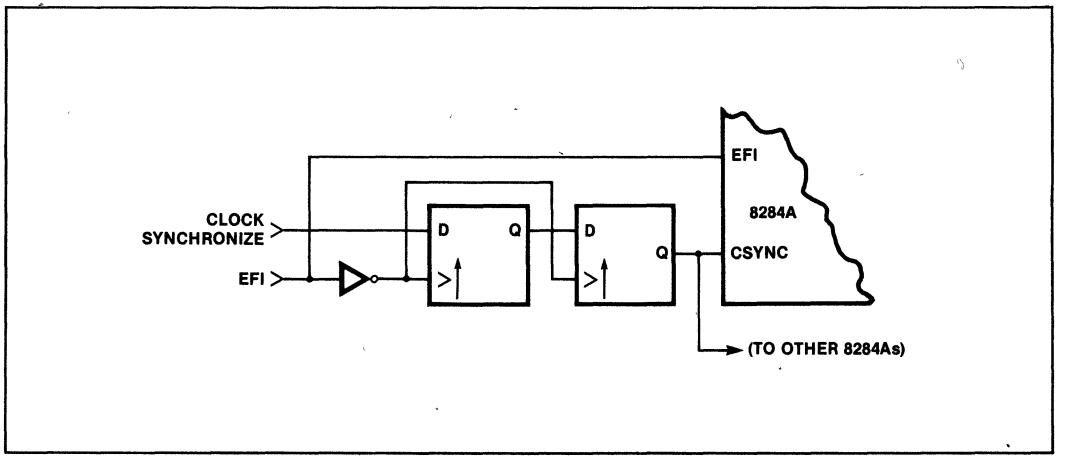


Figure 3. CSYNC Synchronization

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 ± 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 ± 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)

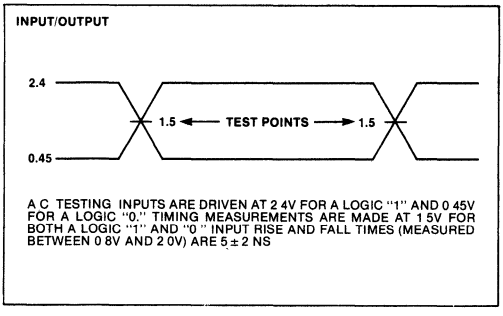
TIMING RESPONSES

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:

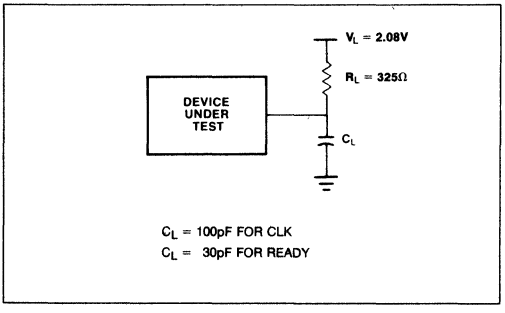
1. Setup and hold necessary only to guarantee recognition at next clock.
2. Applies only to T3 and TW states.
3. Applies only to T2 states.

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

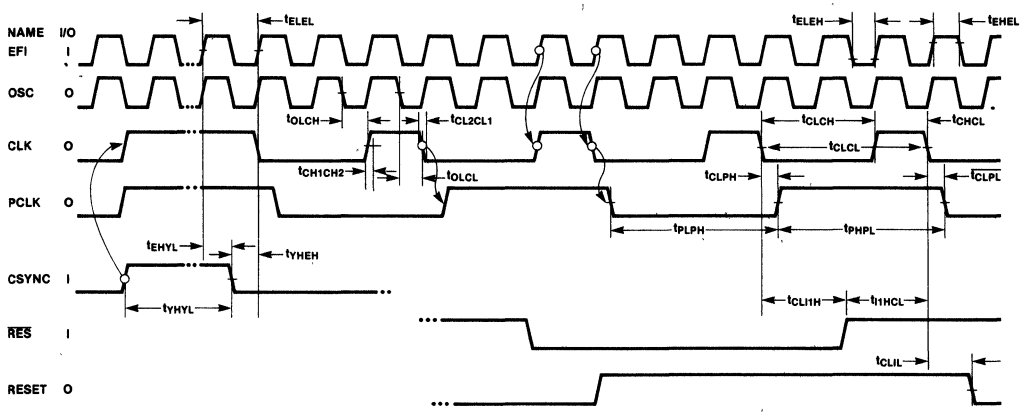
A.C. TESTING LOAD CIRCUIT



C_L = 100pF FOR CLK
C_L = 30pF FOR READY

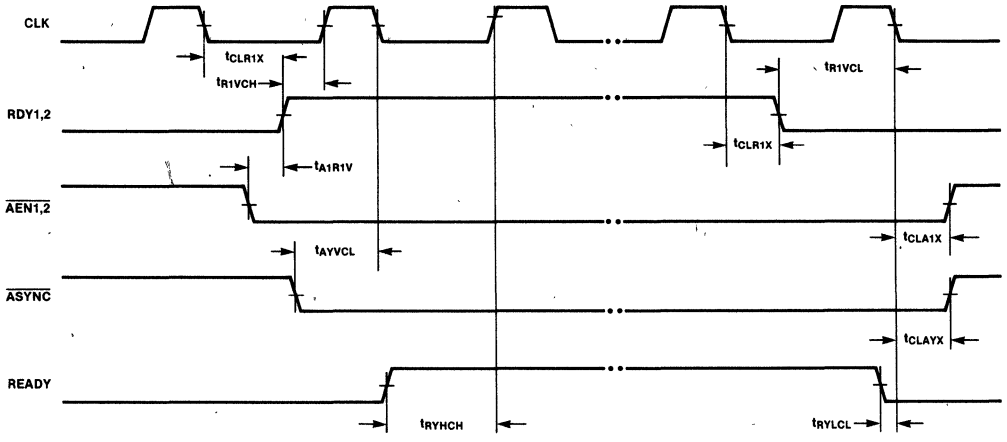
WAVEFORMS

CLOCKS AND RESET SIGNALS



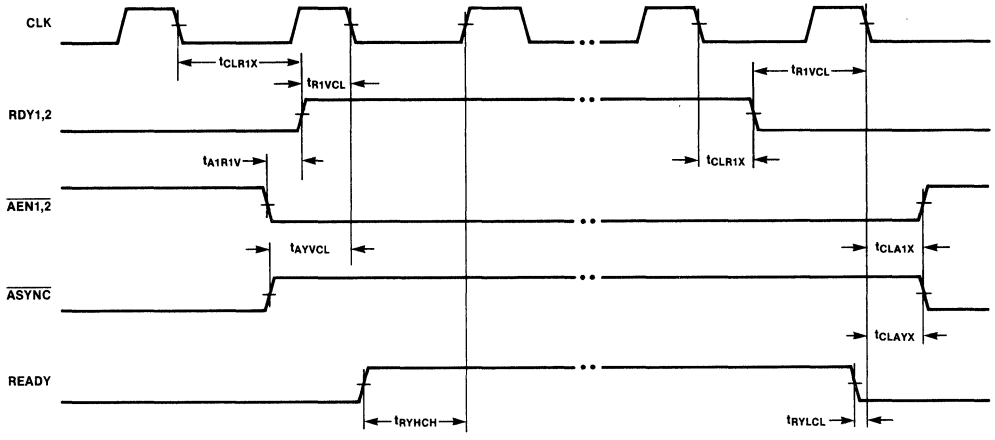
NOTE: ALL TIMING MEASUREMENTS ARE MADE AT 1.5 VOLTS, UNLESS OTHERWISE NOTED.

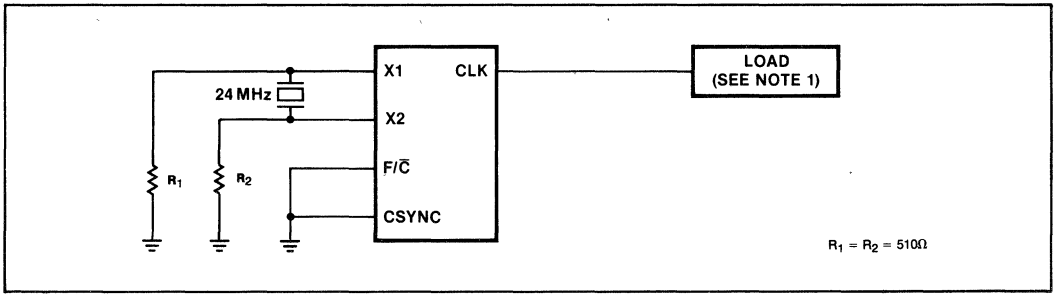
READY SIGNALS (FOR ASYNCHRONOUS DEVICES)



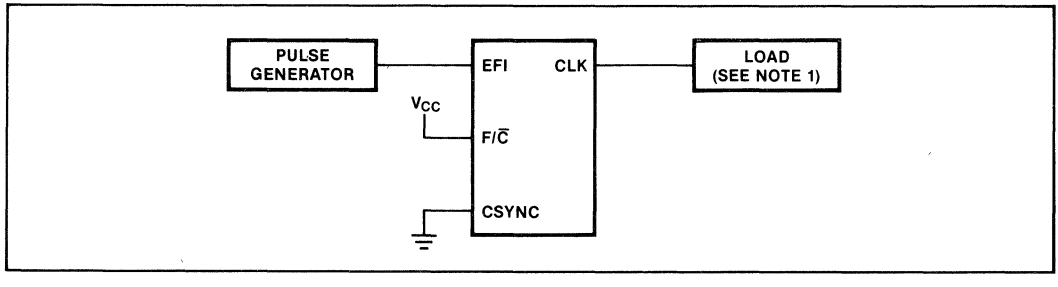
WAVEFORMS (Continued)

READY SIGNALS (FOR SYNCHRONOUS DEVICES)

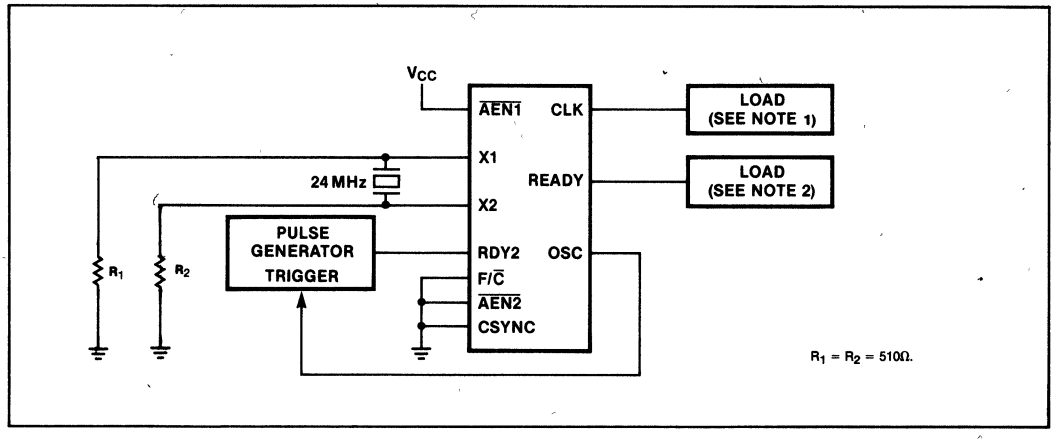




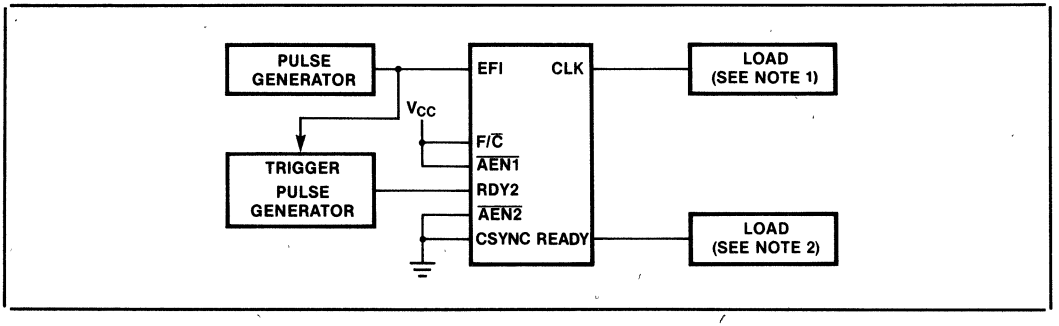
Clock High and Low Time (Using X1, X2)



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.

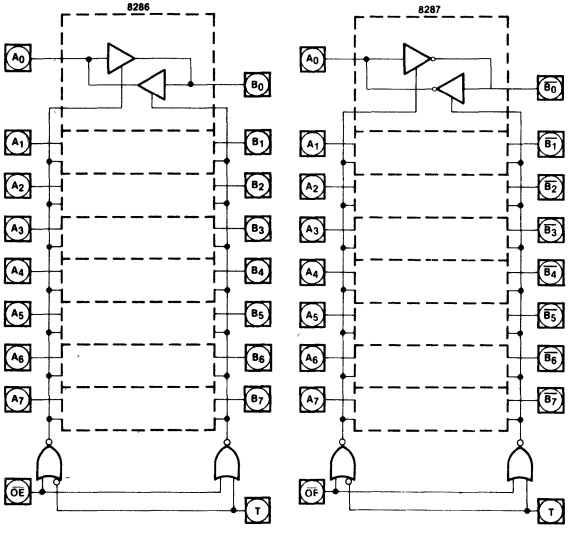


Figure 1. Logic Diagrams

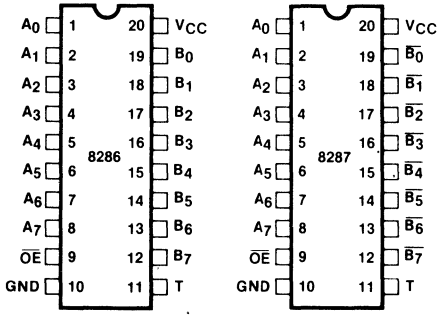


Figure 2. Pin Configurations

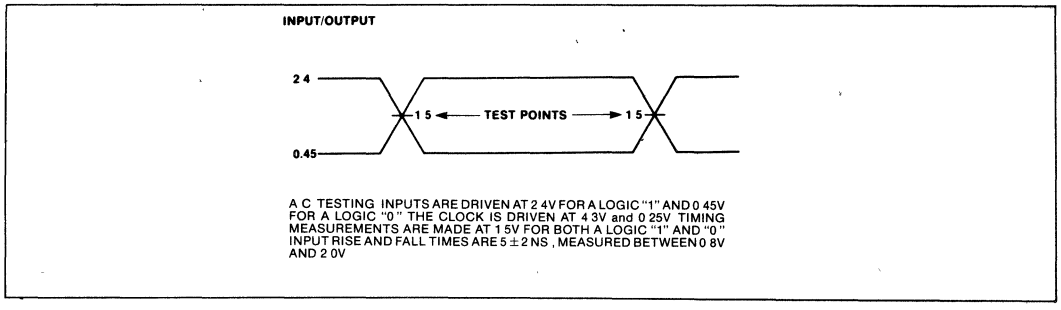
Table 1. Pin Description

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

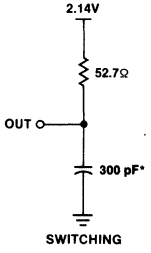
The 8286 and 8287 transceivers are 8-bit transceivers with high impedance outputs. With T active HIGH and $\overline{OE}$ active LOW, data at the A_0–A_7 pins is driven onto the B_0–B_7 pins. With T inactive LOW and $\overline{OE}$ active LOW, data at the B_0–B_7 pins is driven onto the A_0–A_7 pins. No output low glitching will occur whenever the transceivers are entering or leaving the high impedance state.

A.C. TESTING INPUT, OUTPUT WAVEFORM

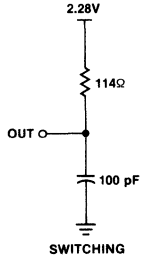


A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". THE CLOCK IS DRIVEN AT 4.3V and 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0". INPUT RISE AND FALL TIMES ARE 5 ± 2 NS, MEASURED BETWEEN 0.8V AND 2.0V.

TEST LOAD CIRCUITS



B 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)

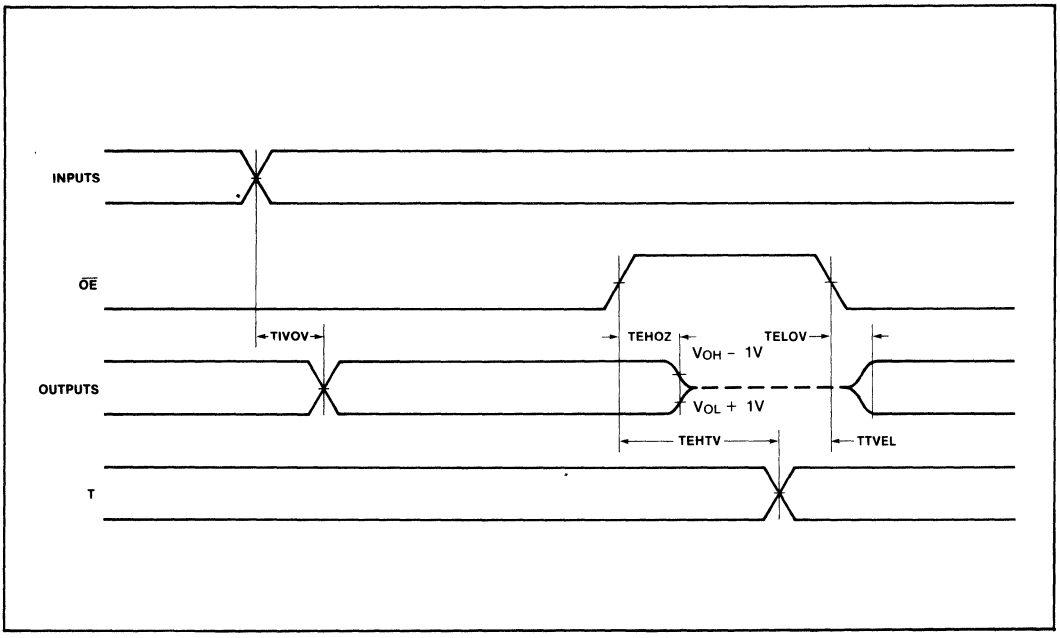
Loading: B Outputs—I_{OL} = 32 mA, I_{OH} = −5 mA, C_L = 300 pF*
A Outputs—I_{OL} = 16 mA, I_{OH} = −1 mA, C_L = 100 pF

Symbol	Parameter	Min	Max	Units	Test Conditions
TIVOV	Input to Output Delay Inverting Non-Inverting	 5 5	 22 30	 ns ns	 (See Note 1)
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

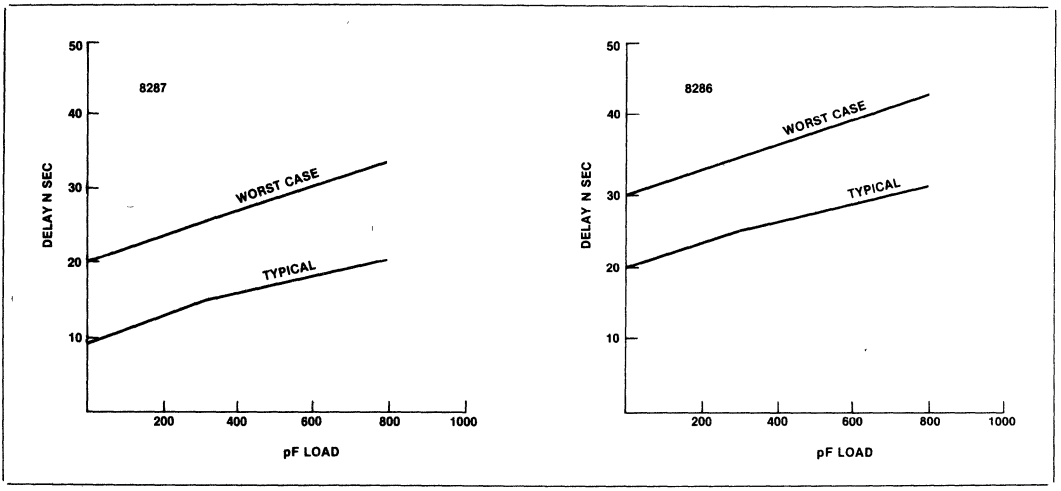
NOTE:
1. See waveforms and test load circuit on following page.
2. For Extended Temperature EXPRESS the Preliminary Maximum Values are TIVOV = 25 vs 22, 35 vs 30; TEHOZ = 25; TELOV = 50.

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**
- **Compatible with 10 MHz iAPX 86 and 8 MHz iAPX 186 based systems.**
- **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.

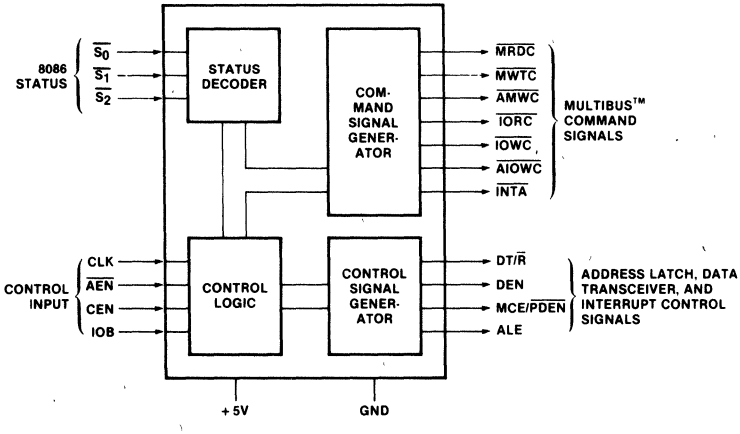


Figure 1. Block Diagram

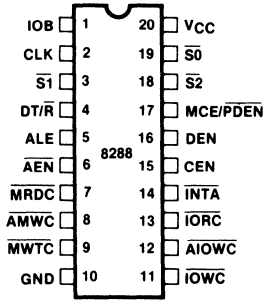


Figure 2. Pin Configuration

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	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.

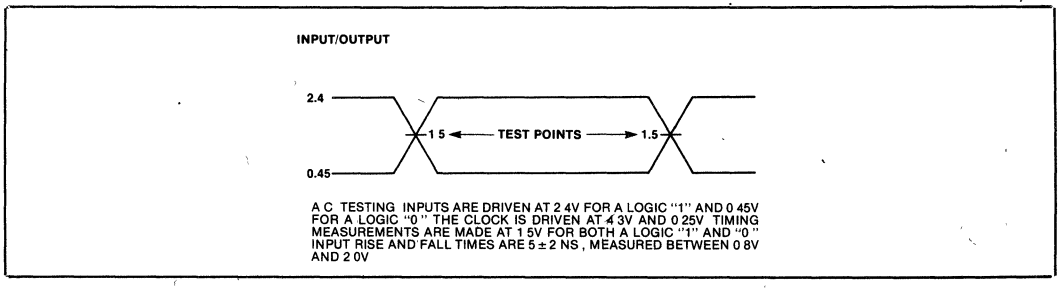
A.C. CHARACTERISTICS (Continued)

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	
TMHNL	Command to DEN Delay (NOTE 1)	TCLCH-5		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

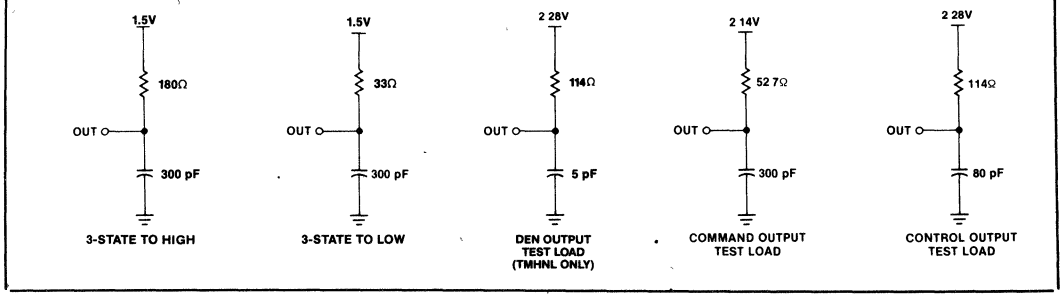
Note 1. TMHNL is tested with DEN C_L = 5 pF

A.C. TESTING INPUT, OUTPUT WAVEFORM

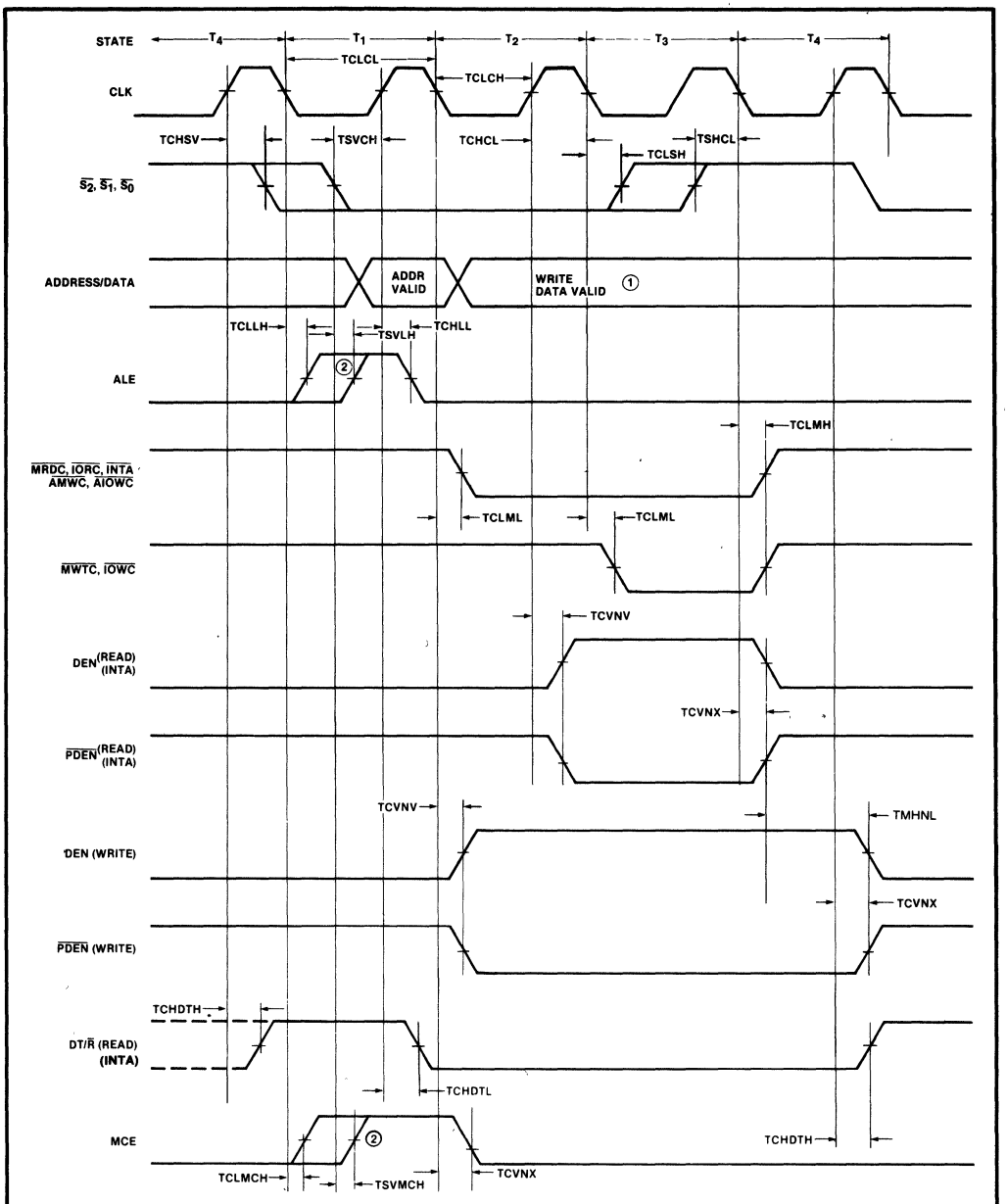


A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". THE CLOCK IS DRIVEN AT 4.3V AND 0.25V. TIMING MEASUREMENTS ARE MADE AT 1.5V FOR BOTH A LOGIC "1" AND "0". INPUT RISE AND FALL TIMES ARE 5 ± 2 NS, MEASURED BETWEEN 0.8V AND 2.0V.

TEST LOAD CIRCUITS—3-STATE COMMAND OUTPUT TEST LOAD



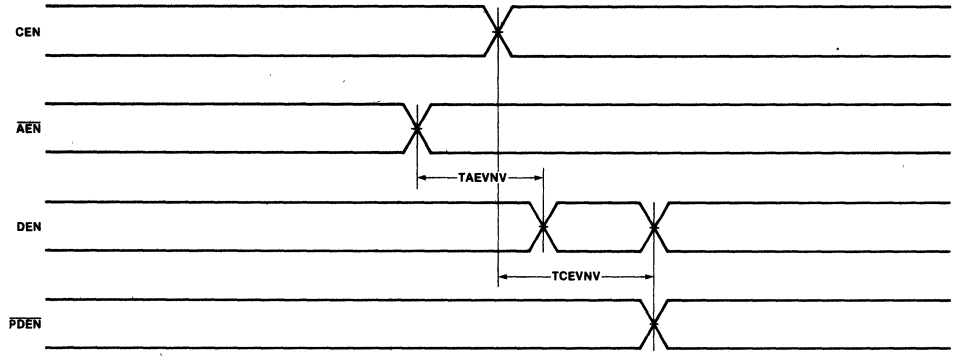
WAVEFORMS



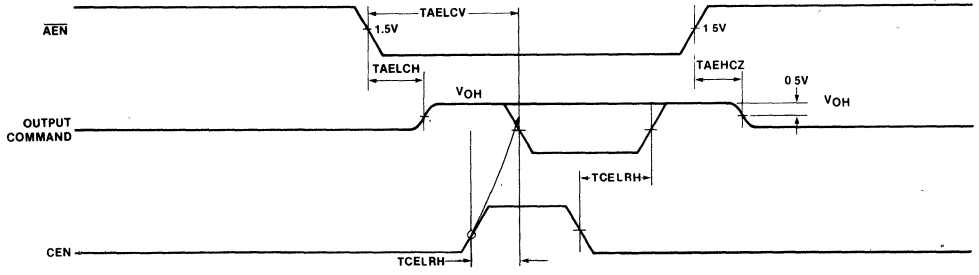
NOTES:
1. ADDRESS/DATA BUS IS SHOWN ONLY FOR REFERENCE PURPOSES
2. LEADING EDGE OF ALE AND MCE IS DETERMINED BY THE FALLING EDGE OF CLK OR STATUS GOING ACTIVE, WHICHEVER OCCURS LAST
3. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS SPECIFIED OTHERWISE.

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.

PRELIMINARY

82188
INTEGRATED BUS CONTROLLER FOR iAPX 86, 88, 186, 188 PROCESSORS

- **Provides Flexibility in System Configurations**
  - **— Supports 8087 Numerics Coprocessor in 80186 and 80188 Systems**
  - **— Provides a Low-cost Interface for 8086, 8088 Systems to an 82586 LAN Coprocessor or 82730 Text Coprocessor**
- **Facilitates Interface to one or more Multimaster Busses**
- **Supports Multiprocessor, Local Bus Systems**
- **Allows use of 80186, 80188 High-Integration Features**
- **3-State, Command Output Drivers**
- **Available in EXPRESS**
  - **— Standard Temperature Range**
  - **— Extended Temperature Range**

The 82188 Integrated Bus Controller (IBC) is a 28-pin HMOS III component for use with 80186, 80188, 8086 and 8088 systems. The IBC provides command and control timing signals plus a configurable $\overline{RQ}/\overline{GT} \longleftrightarrow$ HOLD-HLDA converter. The device may be used to interface an 8087 Numerics Coprocessor with an 80186 or 80188 Processor. Also, an 82586 Local Area Network (LAN) Coprocessor or 82730 Text Coprocessor may be interfaced to an 8086 or 8088 with the IBC.

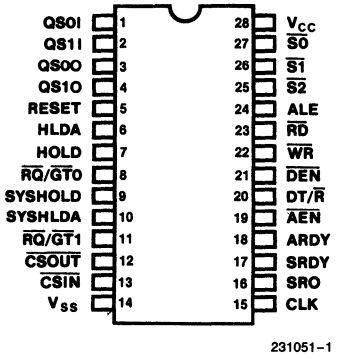


Figure 1.
82188 Pin Configuration

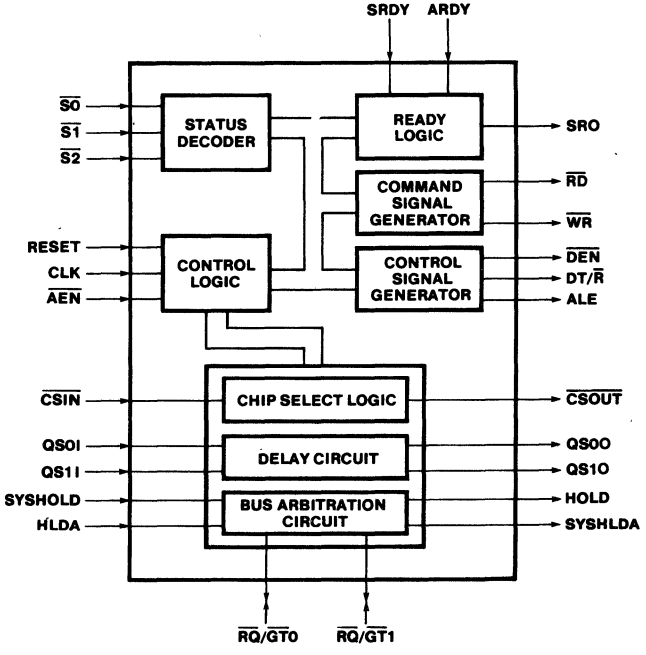


Figure 2.
82188 Block Diagram

PIN DESCRIPTIONS

Symbol	Pin No.	Type	Name and Function
$\overline{S0}$ $\overline{S1}$ $\overline{S2}$	27 26 25	I	**Status Input Pins** $\overline{S0}$–$\overline{S2}$ correspond to the status pins of the CPU. The 82188 uses the status lines to detect and identify the processor bus cycles. The 82188 decodes $\overline{S0}$–$\overline{S2}$ to generate the command and control signals. $\overline{S0}$–$\overline{S2}$ are also used to insert 3 wait states into the SRO line during the first 256 80186 bus cycles after RESET. A HIGH input on all three lines indicates that no bus activity is taking place. The status input lines contain weak internal pull-up devices.

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	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)

CLK	15	I	**CLOCK** CLK is the clock signal generated by the CPU or clock generator device. CLK edges establish when signals are sampled and generated.
RESET	5	I	**RESET** RESET is a level triggered signal that corresponds to the system reset signal. The signal initializes an internal bus cycle counter, thus enabling the 82188 to insert internally generated wait states into the SRO signal during system initialization. The 82188 mode is also determined during RESET. $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ are driven HIGH during RESET regardless of $\overline{AEN}$. RESET is active HIGH.
$\overline{AEN}$	19	I	**Address Enable** This signal enables the system command lines when active. If $\overline{AEN}$ is inactive (HIGH), $\overline{RD}$, $\overline{WR}$, and $\overline{DEN}$ will be tri-stated and ALE will be driven LOW (DT/$\overline{R}$ will not be effected). $\overline{AEN}$ is an asynchronous signal and is active LOW.
ALE	24	O	**Address Latch Enable** This signal is used to strobe an address into address latches. ALE is active HIGH and latch should occur on the HIGH to LOW transition. ALE is intended for use with transparent D-type latches.
$\overline{DEN}$	21	O	**Data Enable** This signal is used to enable data transceivers located on either the local or system data bus. The signal is active LOW. $\overline{DEN}$ is tri-stated when $\overline{AEN}$ is inactive.
DT/$\overline{R}$	20	O	**Data TRANSMIT/RECEIVE** This signal establishes the direction of data flow through the data transceivers. A HIGH on this line indicates TRANSMIT (write to I/O or memory) and a LOW indicates RECEIVE (Read from I/O or memory).

231051-002

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{RD}$	23	O	**READ** This signal instructs an I/O or memory device to drive its data onto the data bus. The $\overline{RD}$ signal is similiar to the $\overline{RD}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{RD}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
$\overline{WR}$	22	O	**WRITE** This signal instructs an I/O or memory device to record the data presented on the data bus. The $\overline{WR}$ signal is similiar to the $\overline{WR}$ signal of the 80186(80188) in Non-Queue-Status Mode. $\overline{WR}$ is active LOW and is tri-stated when $\overline{AEN}$ is inactive.
HOLD	7	O	**HOLD** The HOLD signal is used to request bus control from the 80186 or 80188. The request can come from either the 8087 ($\overline{RQ}/\overline{GT}O$) or from the third processor (SYSHOLD). The signal is active HIGH.
HLDA	6	I	**HOLD Acknowledge** 80186 MODE–This line serves to translate the HLDA output of the 80186(80188) to the appropriate signal of the device requesting the bus. HLDA going active (HIGH) indicates that the 80186 has relinquished the bus. If the requesting device is the 8087, HLDA will be translated into the grant pulse of the $\overline{RQ}/\overline{GT}O$ line. If the requesting device is the optional third processor, HLDA will be routed into the SYSHLDA line. This pin also determines the mode in which the 82188 will operate. If this line is HIGH during the falling edge of RESET, the 82188 will enter the 8086 mode. If LOW, the 82188 will enter the 80186 mode. For 8086 mode, this pin should be strapped to V_{CC}.
$\overline{RQ}/\overline{GT}O$	8	I/O	**Request/Grant O** $\overline{RQ}/\overline{GT}O$ is connected to $\overline{RQ}/\overline{GT}O$ of the 8087 Numeric Coprocessor. When initiated by the 8087, $\overline{RQ}/\overline{GT}O$ will be translated to HOLD-HLDA to acquire the bus from the 80186(80188). This line is bidirectional, and is active LOW. $\overline{RQ}/\overline{GT}O$ has a weak internal pull-up device to prevent erroneous request/grant signals.
$\overline{RQ}/\overline{GT}1$	11	I/O	**Request/Grant 1** 80186 Mode–In 80186 Mode, $\overline{RQ}/\overline{GT}1$ allows a third processor to take control of the local bus when the 8087 has bus control. For a HOLD-HLDA type third processor, the 82188's $\overline{RQ}/\overline{GT}1$ line should be connected to the $\overline{RQ}/\overline{GT}1$ line of the 8087. 8086 MODE–In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is connected to either $\overline{RQ}/\overline{GT}O$ or $\overline{RQ}/\overline{GT}1$ of the 8086. $\overline{RQ}/\overline{GT}1$ will start its request/grant sequence when the SYSHOLD line goes active. In 8086 Mode, $\overline{RQ}/\overline{GT}1$ is used to gain bus control from the 8086 or 8088. $\overline{RQ}/\overline{GT}1$ is a bidirectional line and is active LOW. This line has a weak internal pull-up device to prevent erroneous request/grant signals.

231051-002

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
SYSHOLD	9	I	**System Hold** 80186 MODE–SYSHOLD serves as a hold input for an optional third processor in an 80186(80188)-8087 system. If the 80186(80188) has bus control, SYSHOLD will be routed to HOLD to gain control of the bus. If the 8087 has bus control, SYSHOLD will be translated to $\overline{RQ}/\overline{GT}1$ to gain control of the bus. 8086 MODE–SYSHOLD serves as a hold input for a coprocessor in an 8086 or 8088 system. SYSHOLD is translated to $\overline{RQ}/\overline{GT}1$ of the 82188 to allow the coprocessor to take control of the bus. SYSHOLD may be an asynchronous signal.
SYSHLDA	10	O	**System Hold Acknowledge** SYSHLDA serves as a hold acknowledge line to the processor or coprocessor connected to it. The device connected to the SYSHOLD-SYSHLDA lines is allowed the bus when SYSHLDA goes active (HIGH).
SRDY	17	I	**Synchronous Ready** The SRDY input serves the same function as SRDY of the 80186(80188). The 82188 combines SRDY with ARDY to form a synchronized ready output signal (SRO). SRDY must be synchronized external to the 82188 and is active HIGH. If tied to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 cycles following RESET. If only ARDY is to be used, SRDY should be tied LOW.
ARDY	18	I	**Asynchronous Ready** The ARDY input serves the same function as ARDY of the 80186(80188). ARDY may be an asynchronous input, and is active HIGH. Only the rising edge of ARDY is synchronized by the 82188. The falling edge must be synchronized external to the 82188. If connected to V_{CC}, SRO will remain active (HIGH) after the first 256 80186 bus cycles following RESET. If only SRDY is to be used, ARDY should be connected LOW.
SRO	16	O	**Synchronous READY Output** SRO provides a synchronized READY signal which may be interfaced directly with the SRDY of the 80186(80188) and READY of the 8087. The SRO signal is an accumulation of the synchronized ARDY signal, the SRDY signal, and the internally generated wait state signal.
QS0I QS1I	1 2	I	**Queue-Status Inputs** QS0I, QS1I are connected to the Queue-Status lines of the 80186(80188) to allow synchronization of the queue-status signals to 8087 timing requirements.
QS0O QS1O	3 4	O	**Queue-Status Outputs** QS0O, QS1O are connected to the queue-status pins of the 8087. The signals produced meet 8087 Queue-Status input requirements.

231051-002

PIN DESCRIPTIONS (Continued)

Symbol	Pin No.	Type	Name and Function
$\overline{CSIN}$	13	I	**Chip-Select Input** $\overline{CSIN}$ is connected to one of the chip-select lines of the 80186(80188). $\overline{CSIN}$ informs the 82188 that a bank select is taking place. The 82188 routes this signal to the chip-select output ($\overline{CSOUT}$). $\overline{CSIN}$ is active LOW. This line is not used when memory and I/O device addresses are decoded external to the 80186(80188).
$\overline{CSOUT}$	12	O	**Chip-Select Output** This signal is used as a chip-select line for a bank of memory devices. It is active when $\overline{CSIN}$ is active or when the 8087 has bus control. $\overline{CSOUT}$ is active LOW.

FUNCTIONAL DESCRIPTION

BUS CONTROLLER

The 82188 Integrated Bus Controller (IBC) generates system control and command signals. The signals generated are determined by the Status Decoding Logic. The bus controller logic interprets status lines $\overline{S0}$–$\overline{S2}$ to determine what type of bus cycle is taking place. The appropriate signals are then generated by the Command and Control Signal Generators.

The Address Enable ($\overline{AEN}$) line allows the command and control signals to be disabled. When $\overline{AEN}$ is inactive (HIGH), the command signals and $\overline{DEN}$ will be tri-stated, and ALE will be held low (DT/$\overline{R}$ will be uneffected). $\overline{AEN}$ inactive will allow other systems to take control of the bus. Control and command signals respond to a change in the $\overline{AEN}$ signal within 40 ns.

The command signals consist of $\overline{RD}$ and $\overline{WR}$. The 82188's $\overline{RD}$ and $\overline{WR}$ signals are similiar to $\overline{RD}$ and $\overline{WR}$ of the 80186(80188) in the non-Queue-Status Mode. These command signals do not differentiate between memory and I/O devices. $\overline{RD}$ and $\overline{WR}$ can be conditioned by $\overline{S2}$ of the 80186(80188) to obtain separate signals for I/O and memory devices.

The control commands consist of Data Enable ($\overline{DEN}$), Data Transmit/Receive (DT/$\overline{R}$), and Address Latch Enable (ALE). The control commands are similiar to those generated by the 80186(80188). $\overline{DEN}$ determines when the external bus should be enabled onto the local bus. DT/$\overline{R}$ determines the direction of the data transfer, and ALE determines when the address should be strobed into the latches (used for demultiplexing the address bus).

MODE SELECT

The 82188 Integrated Bus Controller (IBC) is configurable. The device has two modes: 80186 Mode and 8086 Mode. Selecting the mode of the device configures the Bus Arbitration Logic (see BUS ARBITRATION section for details). In 80186 Mode, the 82188 IBC may be used as a bus controller/interface device for an 80186(81088), 8087, and optional third processor system. In 8086 Mode, the 82188 IBC may be used as an interface device allowing a maximum mode 8086(8088) to interface with a coprocessor that uses a HOLD-HLDA bus exchange protocol.

The mode of the 82188 is determined during RESET. If the HLDA line is LOW at the falling edge of RESET (as in the case when tied to the HLDA line of the 80186 or 80188), the 82188 will enter into 80186 Mode. If the HLDA line is HIGH at the falling edge of RESET, the 82188 will enter 8086 Mode. In 8086 Mode, only the Bus Arbitration Logic is used. The eight pins used in 8086 Mode are: SYSHOLD, SYSHLDA, HLDA, CLK, RESET, $\overline{RQ}/\overline{GT1}$, V_{CC}, and V_{SS}. The other pins may be left unconnected.

BUS ARBITRATION

The Bus Exchange Logic interfaces up to three sets of bus exchange signals:

- HOLD-HLDA
- SYSHOLD-SYSHLDA
- $\overline{RQ}/\overline{GT0}$ ($\overline{RQ}/\overline{GT1}$)

This logic executes translating, routing, and arbitrating functions. The logic translates HOLD-HLDA signals to $\overline{RQ}/\overline{GT}$ signals and $\overline{RQ}/\overline{GT}$ signals to HOLD-HLDA signals. The logic also determines which set of bus exchange signals are to be interfaced. The mode of the 82188 and the priority of the devices requesting the bus determine the routing of the bus exchange signals.

231051-002

80186 MODE

In 80186 Mode, a system may have three potential bus masters: the 80186 or 80188 CPU, the 8087 Numerics Coprocessor, and a third processor (such as the 82586 LAN or 82730 Text Coprocessor). The third processor may have either a HOLD-HLDA or $\overline{RQ}/\overline{GT}$ bus exchange protocol. The possible bus exchange signal connections and paths for 80186 Mode are shown in Figures 3 & 4 and Tables 1 & 2, respectively. If no HOLD-HLDA type third processor is used, SYSHOLD should be tied LOW to prevent an erroneous SYSHOLD signal. In 80186 mode, the bus priorities are:

Highest Priority.....Third Processor
Second Highest Priority.....8087
Default Priority 80186

— THREE-PROCESSOR SYSTEM OPERATION (HOLD-HLDA TYPE THIRD PROCESSOR)

In the configuration shown in Figure 3, the third processor requests the bus by sending SYSHOLD HIGH. The 82188 will route (and translate if necessary) the request to the current bus master. This includes routing the request to HOLD if the 80186(80188) is the current bus master or routing and translating the request to $\overline{RQ}/\overline{GT}1$ if the 8087 is in control of the bus. The third processor's request is not passed through the 8087 if the 80186 is the bus master (see Table 1).

The 8087 requests the bus using $\overline{RQ}/\overline{GT}0$. The request pulse from the 8087 will be translated and routed to HOLD if the 80186 is the bus master. If the third processor has control of the bus, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus (sending SYSHOLD LOW). In this case, HOLD will remain HIGH during the third processor-to-8087 bus control transfer. The 80186 will not be granted the bus until both coprocessors have released it.

Table 1. Bus Exchange Paths (80186 Mode) (HOLD-HLDA Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \frac{HOLD}{HLDA}$	$\frac{SYSHOLD}{SYSHLDA} \longleftrightarrow \overline{RQ}/\overline{GT}1$	n/a

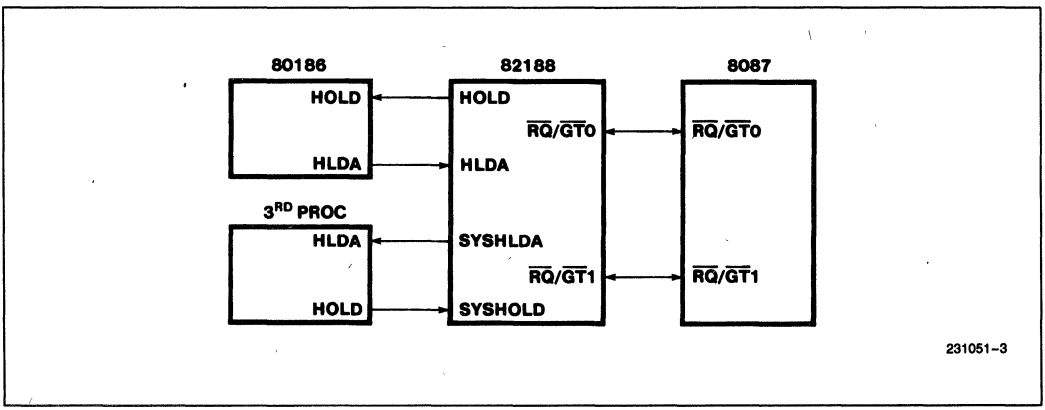


Figure 3.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System (HOLD-HLDA Type 3rd Proc)

231051-002

Table 2. Bus Exchange Paths (80186 Mode) ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

Requesting Device	Current Bus Master		
	80186	8087	3rd Proc
80186	n/a	n/a	n/a
8087	$\overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	n/a	n/a
3rd Proc	$\overline{RQ}/\overline{GT}1 \longleftrightarrow \overline{RQ}/\overline{GT}0 \longleftrightarrow \frac{HOLD}{HLDA}$	$\overline{RQ}/\overline{GT}1$	n/a

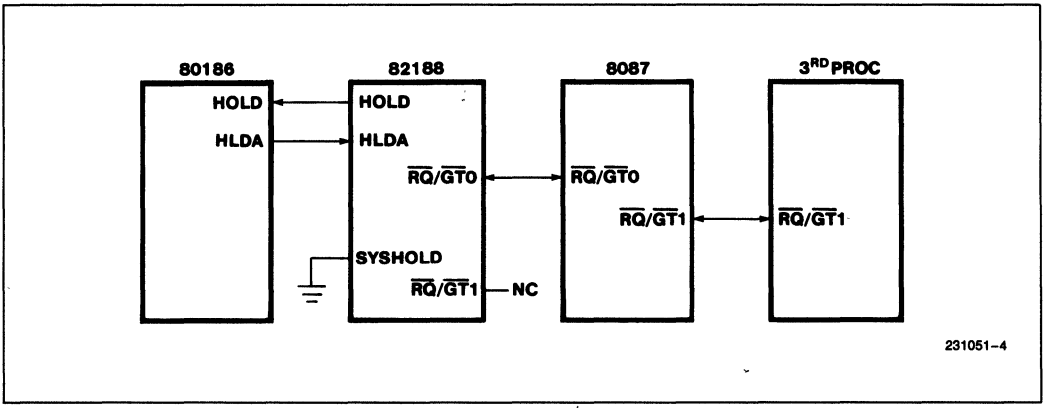


Figure 4.
Bus Exchange Signal Connections (80186 Mode) for a Three Local Processor System ($\overline{RQ}/\overline{GT}$ Type 3rd Proc)

When the bus is requested from the 80186(80188), a bus priority decision is made. This decision is made when the HLDA line goes active. Upon receipt of the HLDA signal, the highest-priority requesting device will be acknowledged the bus. For example, if the 8087 initially requested the bus, the bus will be granted to the third processor if SYSHOLD became active before HLDA was received by the 82188. In this case, the grant pulse to the 8087 will be delayed until the third processor relinquishes the bus.

— THREE-PROCESSOR SYSTEM OPERATION ($\overline{RQ}/\overline{GT}$ TYPE THIRD PROCESSOR)

In the configuration shown in Figure 4, the third processor requests the bus by initiating a request/grant sequence with the 8087's $\overline{RQ}/\overline{GT}1$ line. The 8087 will grant the bus if it is the current bus master or will pass the request on if the 80186 is the current bus master (see Table 2). In this configuration, the 82188's Bus Arbitration Logic translates $\overline{RQ}/\overline{GT}0$ to HOLD-HLDA. The 8087 provides the bus arbitration in this configuration.

8086 MODE

The 8086 Mode allows an 8086, 8088 system to contain both $\overline{RQ}/\overline{GT}$ and HOLD-HLDA type coprocessors simultaneously. In 8086 Mode, two possible bus masters may be interfaced by the 82188; an 8086 or 8088 CPU and a coprocessor which uses a HOLD-HLDA bus exchange protocol (typically an 82586 LAN Coprocessor or an 82730 Text Coprocessor). The bus exchange signal connections for 8086 Mode are shown in Figure 5. Bus arbitration signals used in the 8086 Mode are:

- $\overline{RQ}/\overline{GT}1$
- SYSHOLD
- SYSHLDA

In 8086 Mode, no arbitration is necessary since only two devices are interfaced. The coprocessor has bus priority over the 8086(8088). SYSHOLD-SYSHLDA are routed and translated directly to $\overline{RQ}/\overline{GT}1$. $\overline{RQ}/\overline{GT}1$ of the 82188 may be tied to either $\overline{RQ}/\overline{GT}0$ or $\overline{RQ}/\overline{GT}1$ of the 8086(8088).

231051-002

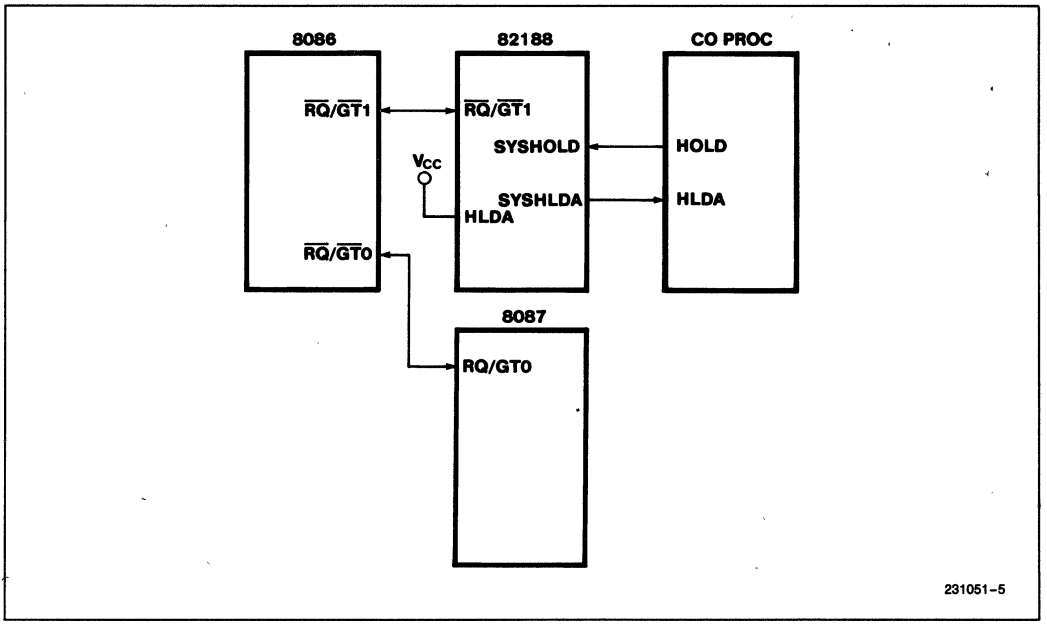


Figure 5. Bus Exchange Signal Connections (8086 Mode)

QUEUE-STATUS DELAY

The Queue-Status Delay logic is used to delay the queue-status signals from the 80186(80188) to meet 8087 queue-status timing requirements. QS0I, QS1I correspond to the queue-status lines of the 80186(80188). The 82188 delays these signals by one clock phase. The delayed signals are interfaced to the 8087 queue-status lines by QS0O, QS1O.

CHIP-SELECT

The Chip-Select Logic allows the utilization of the chip select circuitry of the 80186(80188). Normally, this circuitry could not be used in an 80186(80188)-8087 system since the 8087 contains no chip select circuitry. The Chip-Select Logic contains two external connections: Chip-Select Input ($\overline{CSIN}$) and Chip-Select Output ($\overline{CSOUT}$). $\overline{CSOUT}$ is active when either $\overline{CSIN}$ is active or when the 8087 has control of the bus.

By using $\overline{CSOUT}$ to select memory containing data structures, no external decoding is necessary. The 80186 may gain access to this memory bank through the $\overline{CSIN}$ line while the 8087 will automatically obtain access when it becomes the bus master. Note that this configuration limits the amount of memory accessible by the 8087 to the physical memory bank selected by $\overline{CSOUT}$. Systems where the 8087 must access the full 1 Megabyte address space must use an external decoding scheme.

READY

The Ready logic allows two types of Ready signals: a Synchronous Ready Signal (SRDY) and an Asynchronous Ready Signal (ARDY). These signals are similiar to SRDY and ARDY of the 80186. Wait states will be inserted when both SRDY and ARDY are LOW. Inserting wait states allows slower memory and I/O devices to be interfaced to the 80186(80188)-8087 system.

ARDY's LOW-to-HIGH transition is synchronized to the CPU clock by the 82188. The 82188 samples ARDY at the beginning of T2, T3 and Tw until sampled HIGH. Note that ARDY of the 82188 is sampled one phase earlier than ARDY of the 80186. ARDY's falling edge must be synchronous to the CPU clock. ARDY allows an easy interface with devices that emit an asynchronous ready signal.

The SRDY signal allows direct interface to devices that emit a synchronized ready signal. SRDY must be synchronized to the CPU clock for both of its transitions. SRDY is sampled in the middle of T2, T3 and in the middle of each Tw. An 82188-80186(80188)'s SRDY setup time is 30 ns longer than the 80186(80188)'s SRDY setup time. SRDY eliminates the half-clock cycle penalty necessary for ARDY to be internally sychronized.

The sychronized ready output (SRO) is the accumulation of SRDY, ARDY, and the internal wait-state

231051-002

generator. SRO should be connected to SRDY of the 80186(80188) (with 80186(80188)'s ARDY tied LOW), and READY of the 8087.

SRDY	ARDY	SRO
0	0	0
1	X	1
X	1	1

The internal wait state generator allows for synchronization between the 80186(80188) and 8087 in 80186 mode. Upon RESET, the 82188 automatically inserts 3 wait-states per 80186(80188) bus cycle, overlapped with any externally produced wait-states created by ARDY and SRDY.

Since the 8087 has no provision for internal wait-state generation, only externally created wait states will be effective. The 82188, upon RESET, will inject 3 wait states for each of the first 256 80186(80188) bus cycles onto the SRO line. This will allow the 8087 to match the 80186(80188)'s timing.

The internally-generated wait states are overlapped with those produced by the SRDY and ARDY lines. Overlapping the injected wait states insures a minimum of three wait states for the first 256 80186(80188) bus cycles after RESET. Systems with a greater number of wait states will not be effected. Internal wait state generation by the 82188 will stop on the 256th 80186(80188) bus cycle after RESET. To maintain sychronization between the 80186(80188) and 8087, the following conditions are necessary:

- The 80186(80188)'s control block must be mapped in I/O space before it is written to or read from.
- All memory chip-select lines must be set to 0 WAIT STATES, EXTERNAL READY ALSO USED within the first 256 80186(80188) bus cycles after RESET.

An equivelant READY logic diagram is shown in Figure 6.

SYSTEM CONSIDERATIONS

In any 82188 configuration, clock compatibility must be considered. Depending on the device, a 50% or a 33% duty-cycle clock is needed. For example, the 80186 and 80188 (as well as the 82188, 82586, and 82730) requires a 50% duty-cycle clock. The 8086, 8088 and their 'kit' devices' (8087, 8089, 8288, and 8289) clock requirements, on the other hand, require a 33% duty-cycle clock signal. The system designer must make sure clock requirements of all the devices in the system are met.

Figures 7 & 8 show two system configurations using the 82188. Figure 7 demonstrates the usage of the 82188 in 80186 Mode where it is used to interface an 8087 into an 80186 system. Figure 8 demonstrates the usage of the 82188 in 8086 Mode where it is used to convert the HOLD-HLDA bus exchange protocol of the 82586 to the RQ/GT bus exchange protocol of the 8086.

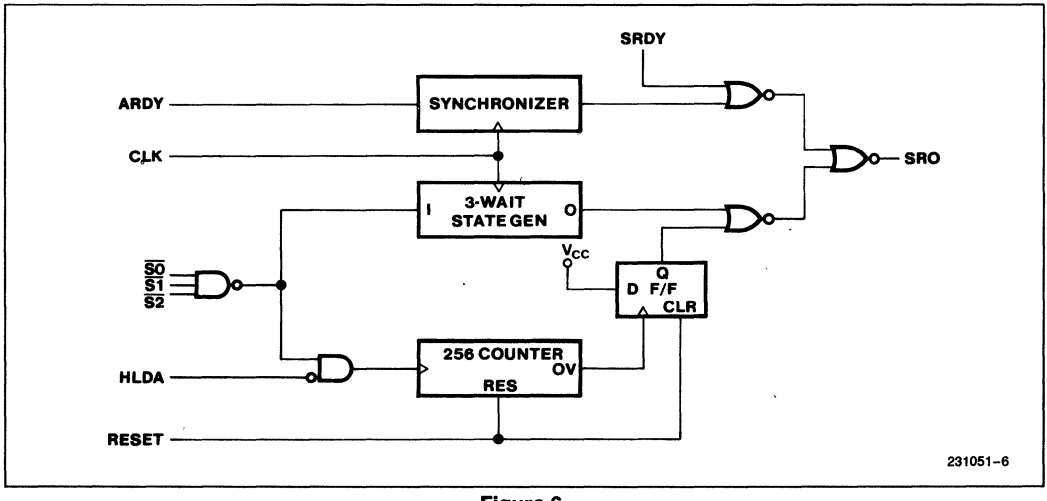


Figure 6.
Equivalent 82188 READY circuit

231051-002

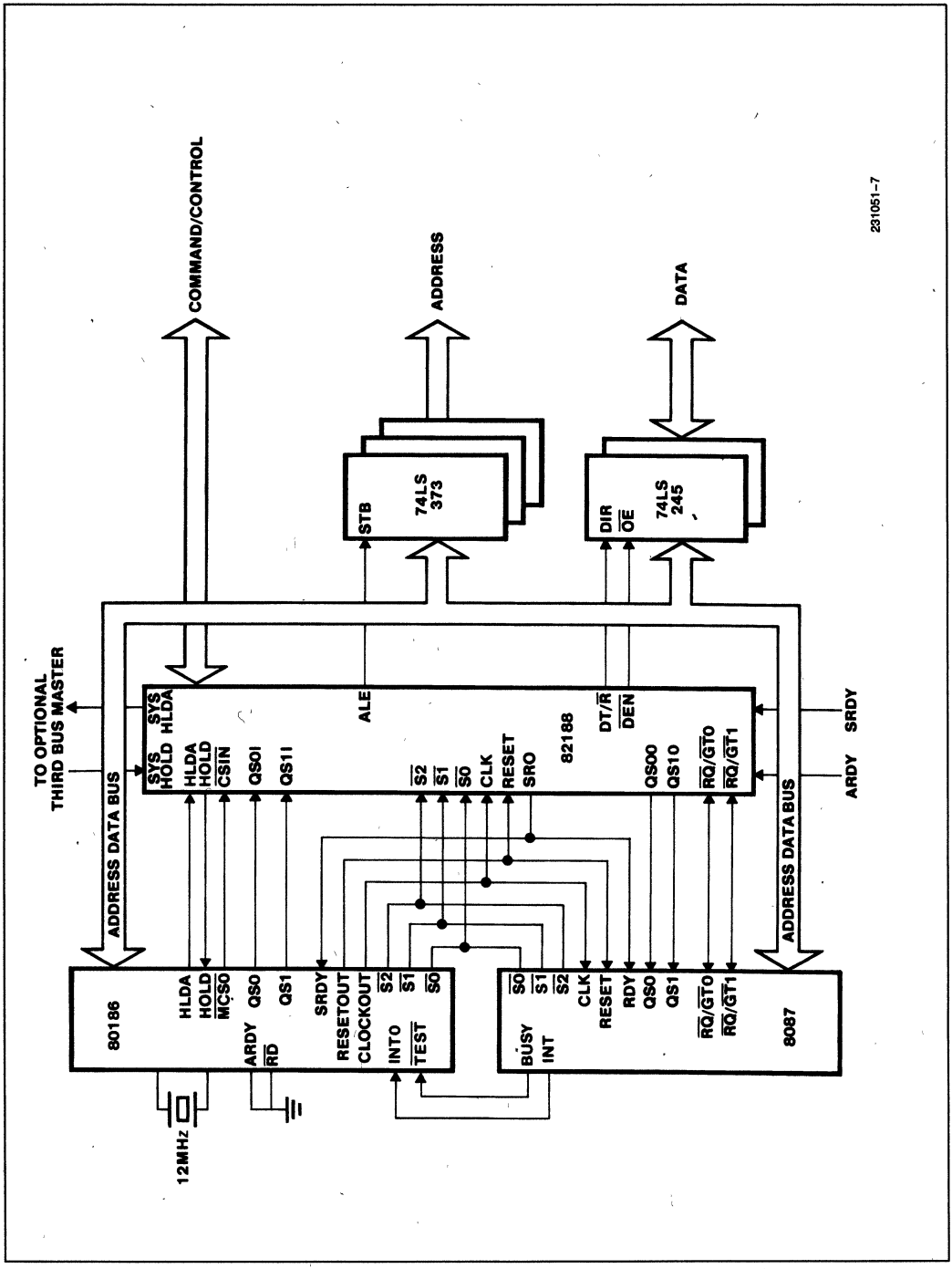


Figure 7.
80186-6/8087-2 System Using the 82188 in 80186 Mode

231051-002

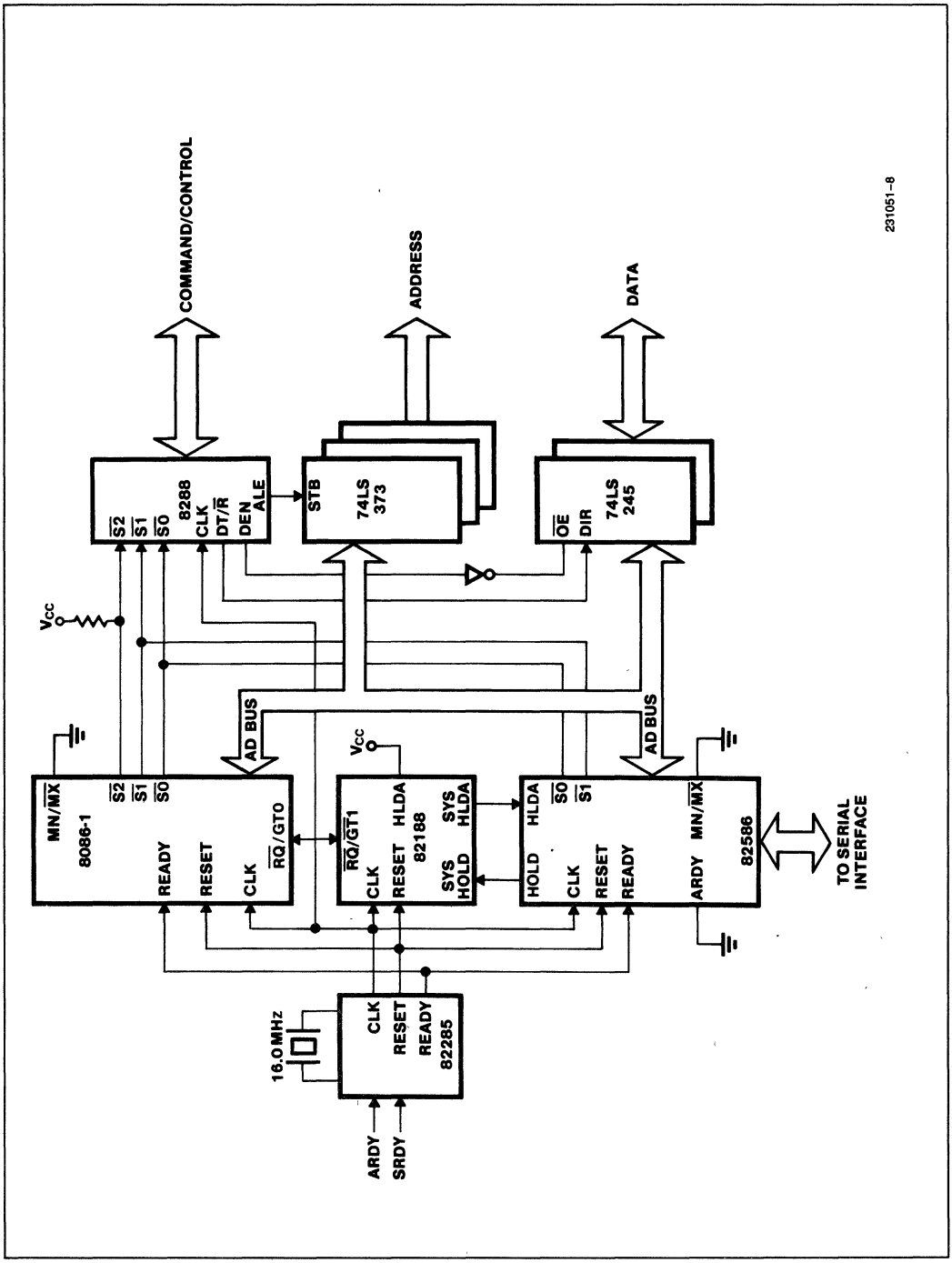


Figure 8.
8086-1/82586 System using the 82188 in 8086 Mode

231051-002

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 GND	−1.0V to 7.0V
Power Dissipation	0.7 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 periods may affect device reliability.

DC CHARACTERISTICS

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

Symbol	Parameter	Min	Max	Units	Test Cond.
V_{IL}	Input Low Voltage	−0.5	+0.8	volts	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	volts	
V_{OL}	Output Low Voltage		0.45	volts	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		volts	I_{OH} = −400 μA
I_{CC}	Power Supply Current		100	mA	T_A = 25°C
I_{LI}	Input Leakage Current		±10	μA	$0V < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45 < V_{OUT} < V_{CC}$
V_{CLI}	CLK Input Low Voltage	−0.5	+0.6	volts	
V_{CHI}	CLK Input High Voltage	3.9	V_{CC} +1.0	volts	
C_{IN}	Input Capacitance		10	pF	
C_{IO}	I/O Capacitance		20	pF	

AC CHARACTERISTICS

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

TIMING REQUIREMENTS

Symbol	Parameter	Min	Max	Units	Notes
TCLCL	Clock Period	125	500	ns	
TCLCH	Clock LOW Time	½TCLCL-7.5		ns	
TCHCL	Clock HIGH Time	½TCLCL-7.5		ns	
TARYHCL	ARDY Active Setup Time	20		ns	9,2
TCHARYL	ARDY Hold Time	15		ns	8
TARYLCH	ARDY Inactive Setup Time	35		ns	
TSRYHCL	SRDY Input Setup Time	65,50		ns	1
TSVCH	STATUS Active Setup Time	55		ns	
TSXCL	STATUS Inactive Setup Time	50		ns	
TQIVCL	QS0I, QS1I Setup Time	10		ns	
THAVGV	HLDA Setup Time	50		ns	
TSHVCL	SYSHOLD Asynchronous Setup Time	25		ns	9,2
TGVCH	$\overline{RQ}/\overline{GT}$ Input Setup Time	0		ns	6

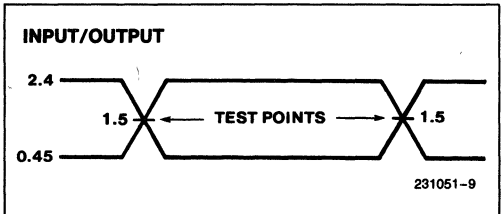
231051-002

TIMING RESPONSES

Symbol	Parameter	Min	Max	Units	Notes
TSVLH	STATUS Valid to ALE Delay		30	ns	4
TCHLL	ALE Inactive Delay		30	ns	
TCLML	$\overline{RD}$, $\overline{WR}$ Active Delay	0	70	ns	
TCLMH	$\overline{RD}$, $\overline{WR}$ Inactive Delay	0	55	ns	
TSVDTV	STATUS to DT/$\overline{R}$ Delay	0	30	ns	3
TCLDTV	DT/$\overline{R}$ Active Delay	0	55	ns	3
TCHDNV	$\overline{DEN}$ Active Delay	0	55	ns	
TCHDNX	$\overline{DEN}$ Inactive Delay	5	55	ns	
TCLQOV	QS0O, QS1O Delay	5	50	ns	
TCHHV	HOLD Delay		50	ns	2,6
TCLSAV	SYSHLDA Delay		50	ns	6
TCLGV	$\overline{RQ}$/$\overline{GT}$ Output Delay		40	ns	6
TGVHV	$\overline{RQ}$/$\overline{GT}$0 To HOLD Delay		50	ns	2,6
TCLLH	ALE Active Delay	0	30	ns	4
TAELCV	Command Enable Delay		40	ns	
TAEHCX	Command Disable Delay		40	ns	
TCHRO	SRO Output Delay	5	30	ns	5,6
TSRYHRO	SRDY To SRO Delay		30	ns	5
TSCICSO	$\overline{CSIN}$ To $\overline{CSOUT}$ Delay		30	ns	

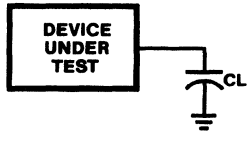
NOTES (applicable to both spec listing and timing diagrams):
1. TSRYHOL = (80186's) TSRYCL + 30 ns=65 ns for 6 MHz operation and 50 ns for 8 MHz operation.
2. Timing not tested.
3. DT/$\overline{R}$ will be asserted to the latest of TSVDTV & TCLDTV.
4. ALE will be asserted to the latest of TSVLH & TCLLH.
5. SRO will be asserted to the latest of TCHRO & TSRYHRO.
6. CL = 20–100 pF
7. Address/Data bus shown for reference only.
8. The falling edge of ARDY must be synchronized to CLK.
9. To guarantee recognition at next falling clock edge.

A.C. Testing Input, Output Waveform



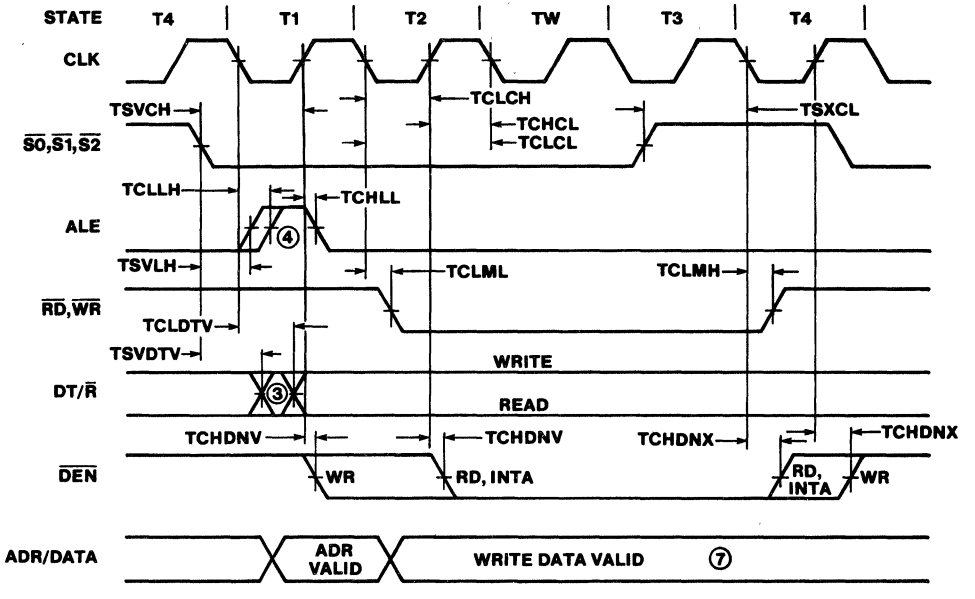
A.C. Testing: Inputs are driven at 2.4V for a Logic '1' and 0.45V for a Logic '0'.

A.C. Testing Load Circuit

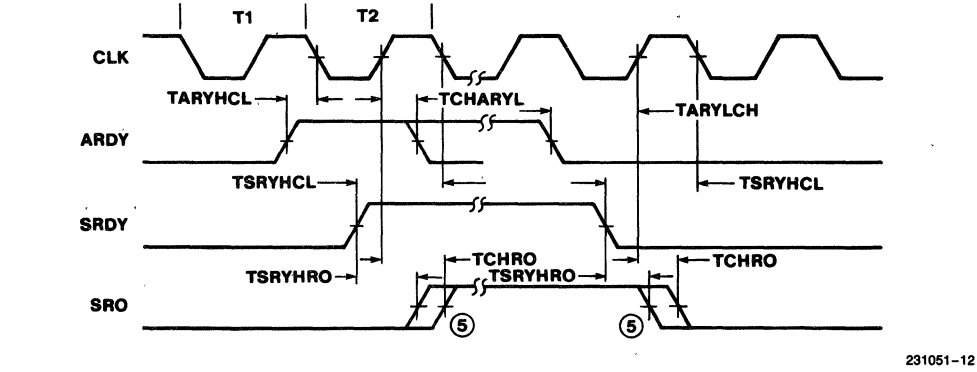


C_L includes Jig Capacitance
C_L = 20–200 pF unless otherwise noted

231051-002

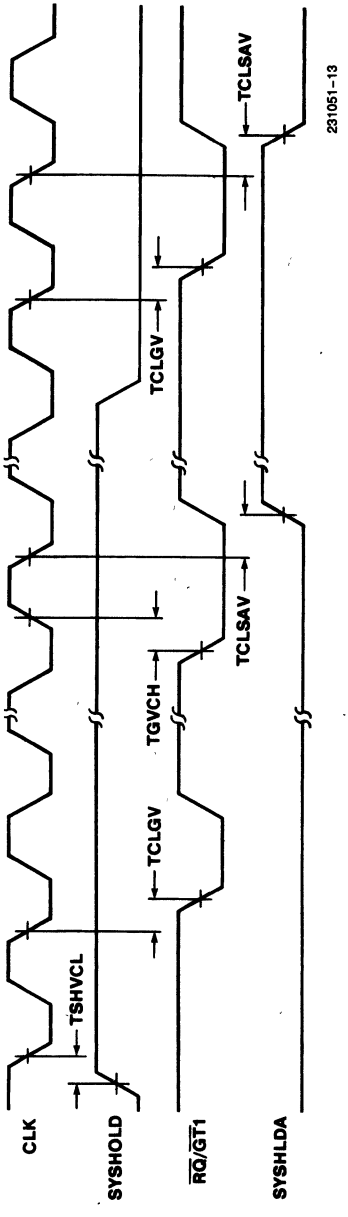


Command and Control Waveforms–80186 Mode



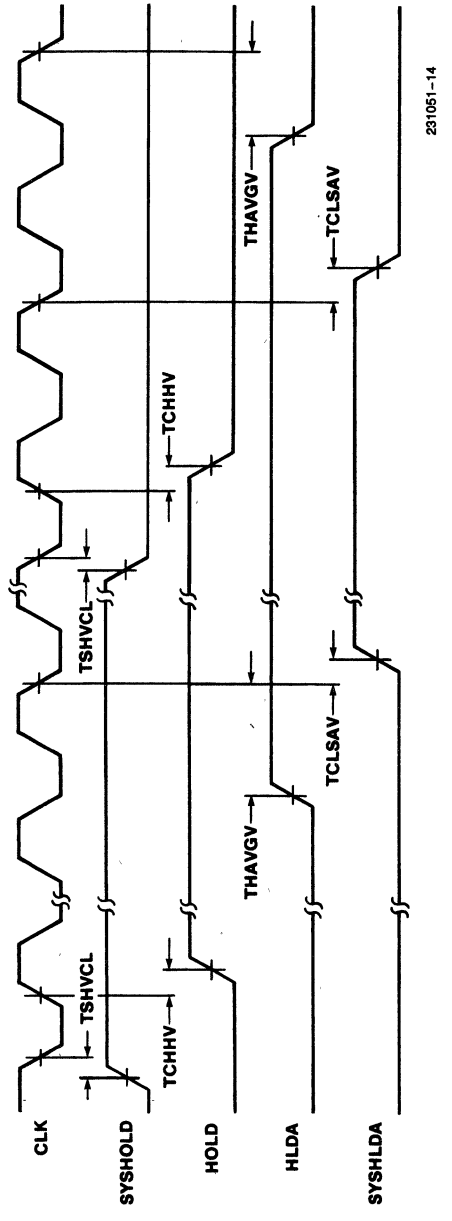
READY Timing–80186 Mode

231051-002



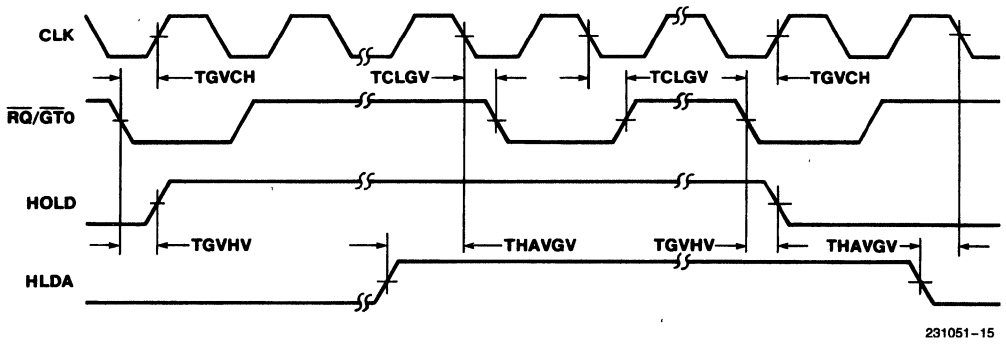
SYSHOLD-SYSHLDA to $\overline{RQ}/\overline{GT}1$ Timing–80186 Mode and 8086 Mode

231051-002

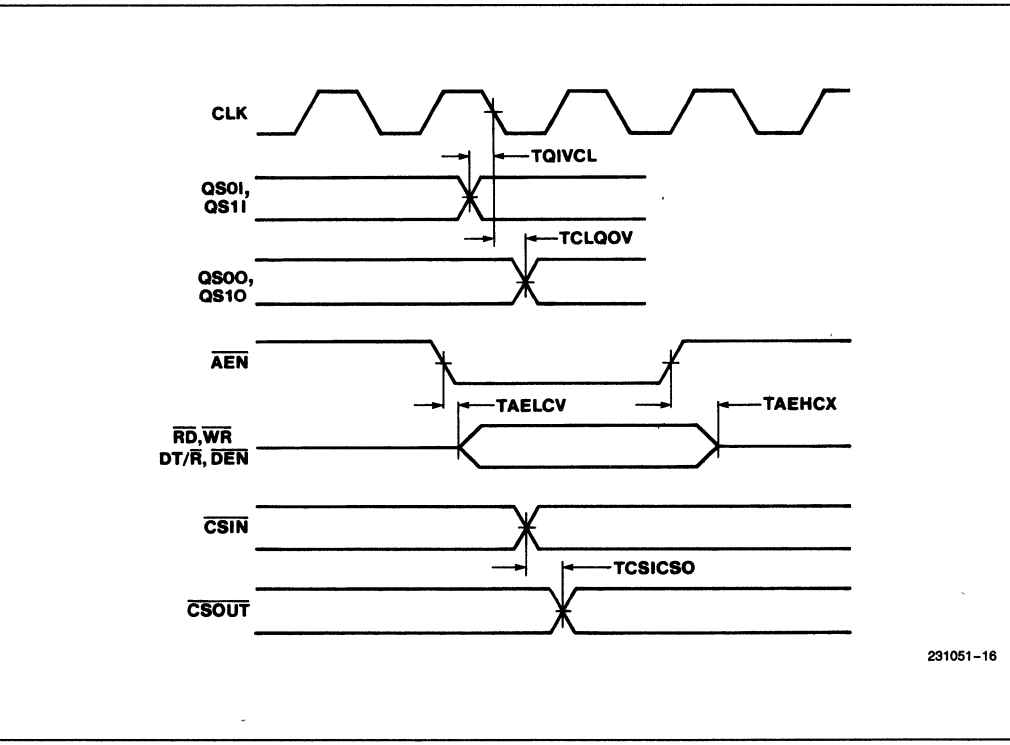


SYSHOLD-SYSHLDA To HOLD-HLDA Timing–80186 Mode

231051-002



RQ/GT0 to HOLD-HLDA Timing–80186 Mode



Queue Status, ALE, Chip Select Delay Timing–80186 Mode

231051-002

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.

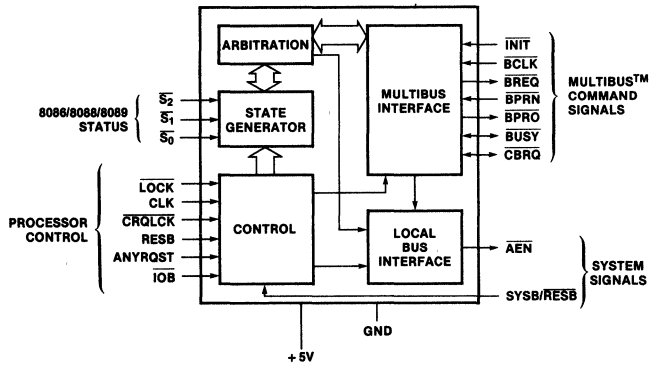


Figure 1. Block Diagram

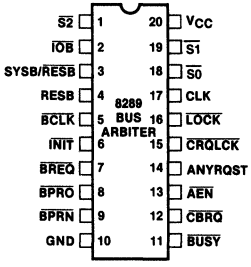


Figure 2. Pin Diagram

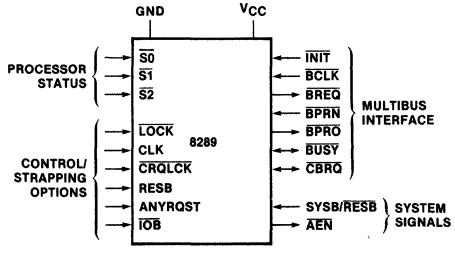


Figure 3. Functional Pinout

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.
$\overline{BPRO}$	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.

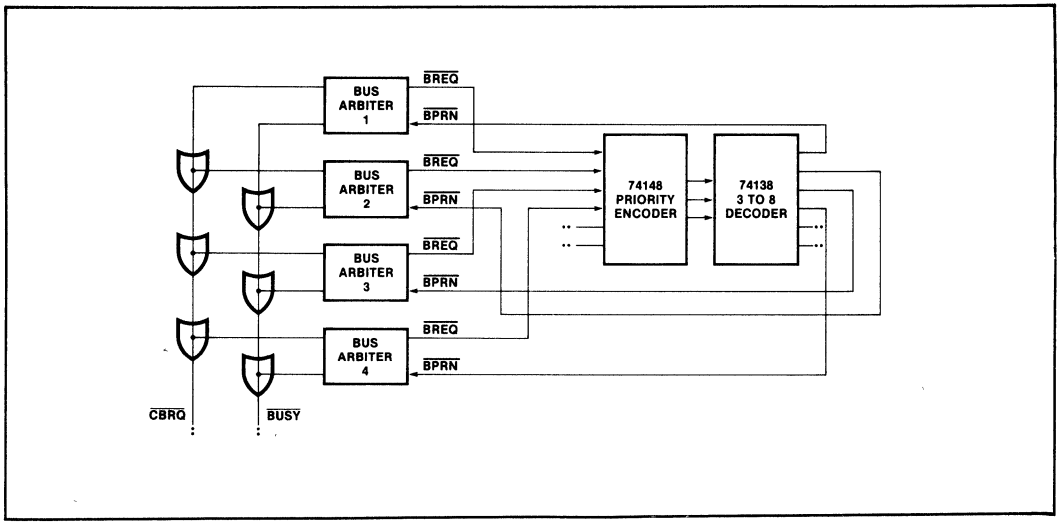


Figure 4. Parallel Priority Resolving Technique

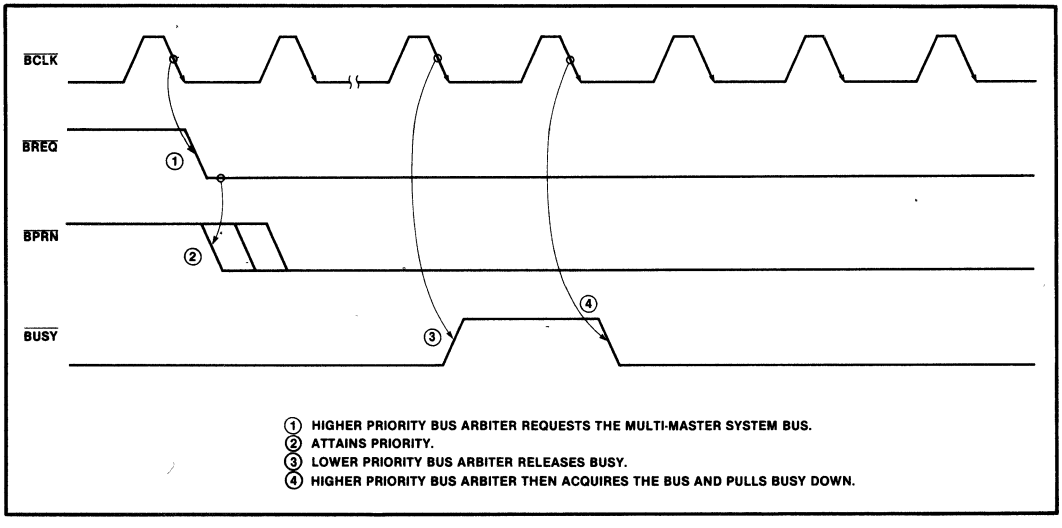


Figure 5. Higher Priority Arbiter obtaining the Bus from a Lower Priority Arbiter

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.

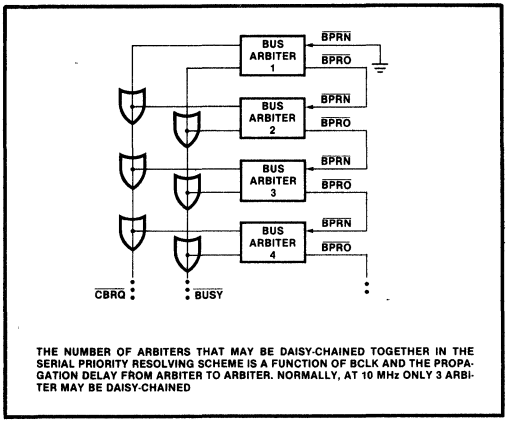


Figure 6. Serial Priority Resolving

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

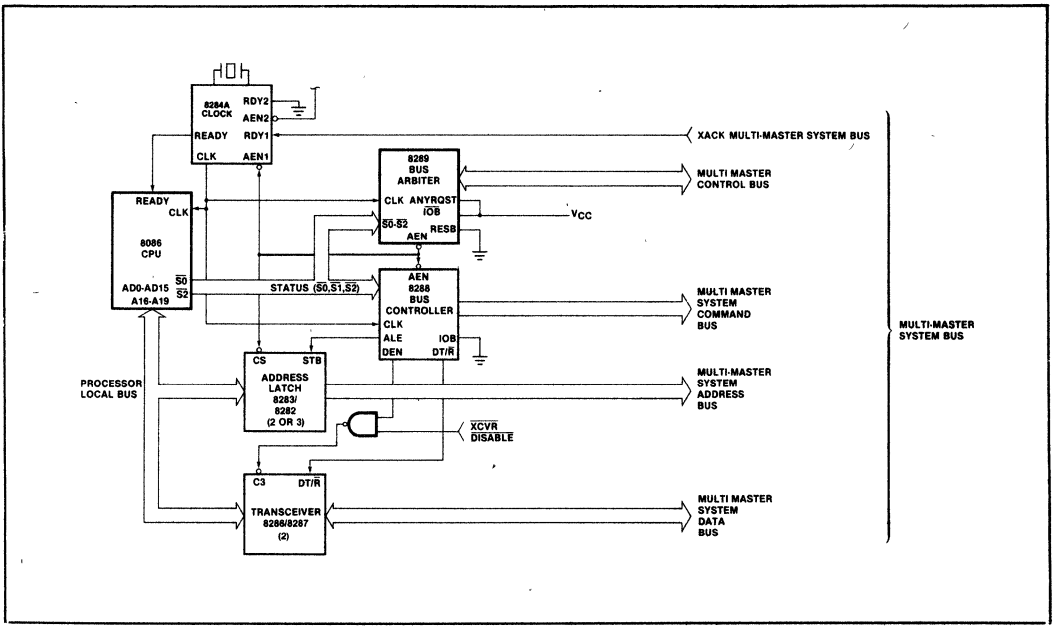


Figure 7. Typical Medium Complexity CPU System

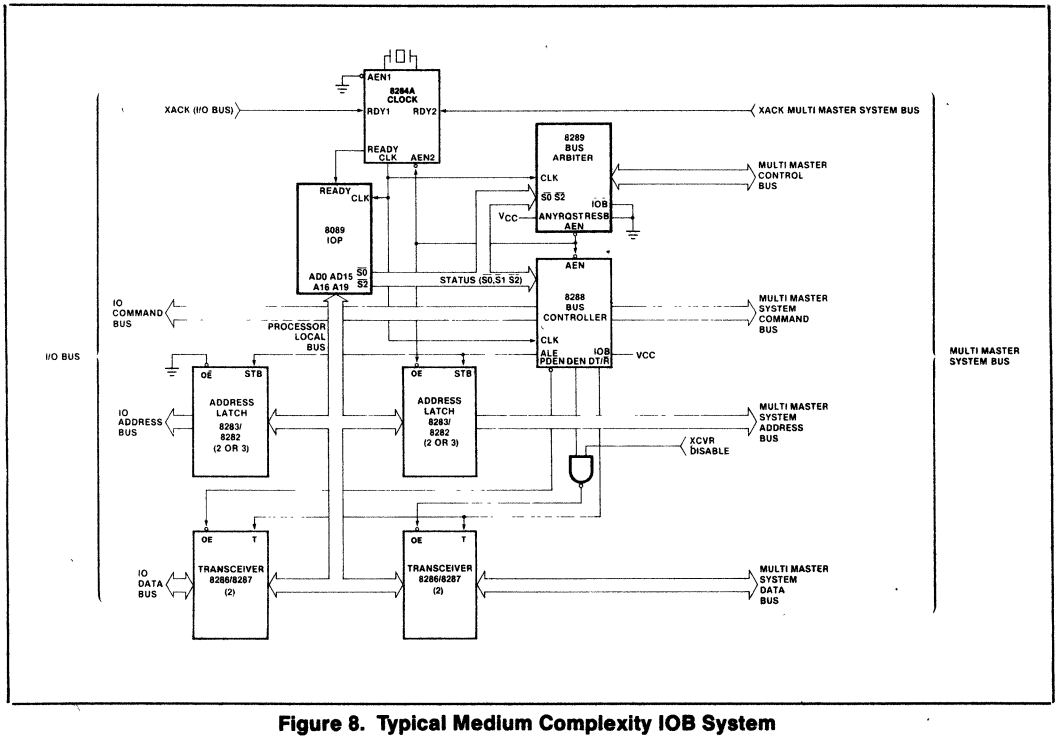


Figure 8. Typical Medium Complexity IOB System

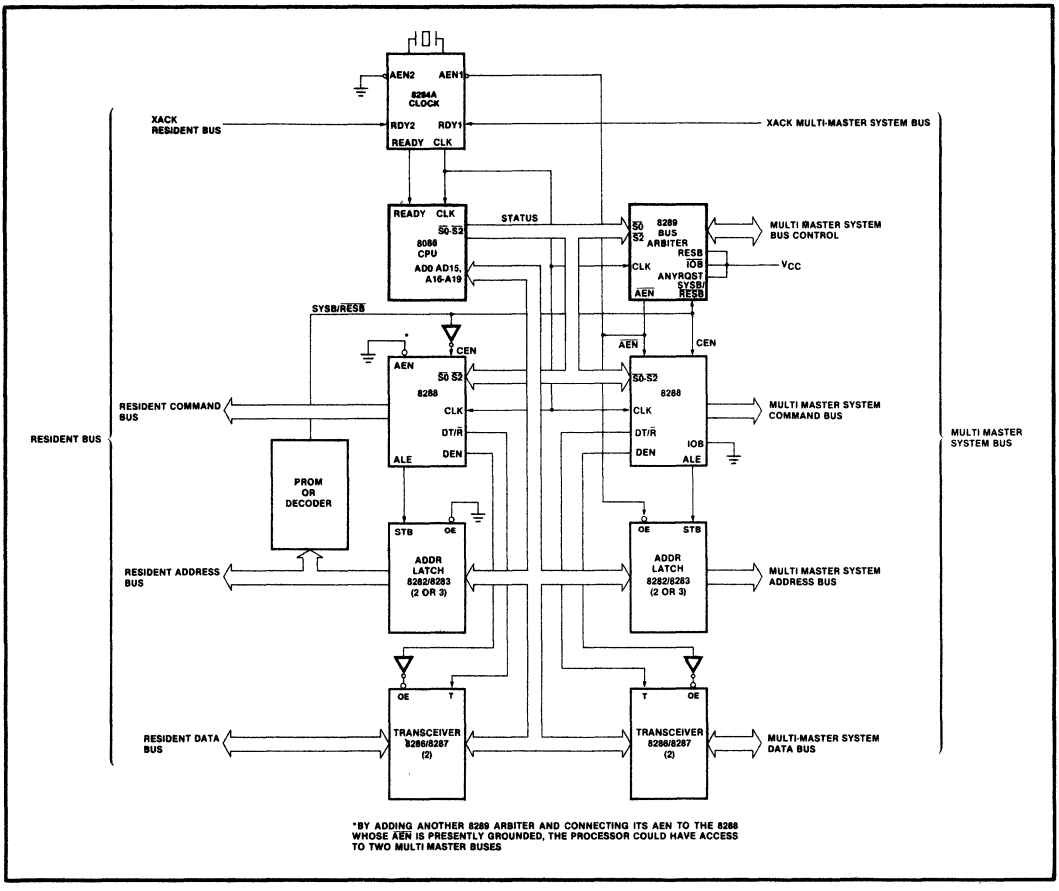


Figure 9. 8289 Bus Arbiter Shown in System-Resident Bus Configuration

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 $\overline{BUSY}$, $\overline{CBRQ}$ $\overline{AEN}$ $\overline{BPRO}$, $\overline{BREQ}$		 0.45 0.45 0.45	 V V V	 I_{OL} = 20 mA I_{OL} = 16 mA 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)

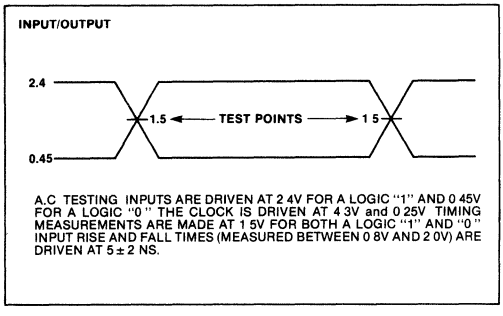
TIMING RESPONSES

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.

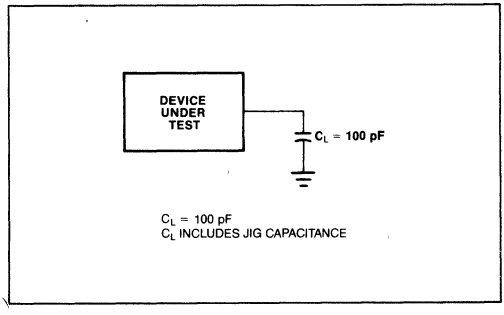
NOTES:
1. BCLK generates the first BPRO wherein subsequent BPRO changes lower in the chain are generated through BPRON.
2. Measured at .5V above GND.

A.C. TESTING INPUT, OUTPUT WAVEFORM



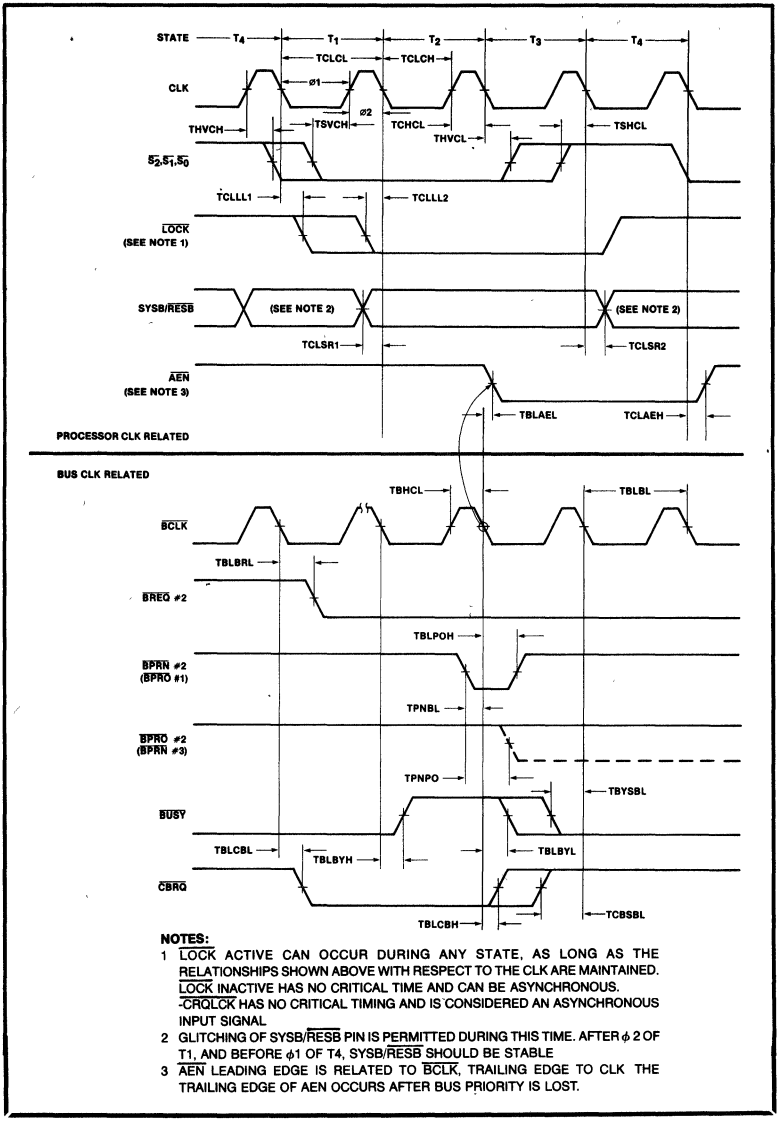
A.C TESTING INPUTS ARE DRIVEN AT 2 4V FOR A LOGIC "1" AND 0 45V FOR A LOGIC "0" THE CLOCK IS DRIVEN AT 4 3V and 0 25V TIMING MEASUREMENTS ARE MADE AT 1 5V FOR BOTH A LOGIC "1" AND "0" INPUT RISE AND FALL TIMES (MEASURED BETWEEN 0 8V AND 2 0V) ARE DRIVEN AT 5 ± 2 NS.

A.C. TESTING LOAD CIRCUIT



C_L = 100 pF
C_L INCLUDES JIG CAPACITANCE

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 ϕ 2 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

intel®

APPLICATION NOTE

AP-67

September 1979

8086 System Design

George Alexy
Microcomputer Applications

ORDER NUMBER: 230792-001

8086 System Design

Contents

1. INTRODUCTION

The 8086 family, Intel's new series of microprocessors and system components, offers the designer an advanced system architecture which can be structured to satisfy a broad range of applications. The variety of speed, configuration and component selections available within the family enables optimization of a specific design to both cost and performance objectives. More important however, the 8086 family concept allows the designer to develop a family of systems providing multiple levels of enhancement within a single design and a growth path for future designs.

This application note is directed toward the implementation of the system hardware and will provide an introduction to a representative sample of the systems configurable with the 8086 CPU member of the family. Application techniques and timing analysis will be given to aid the designer in understanding the system requirements, advantages and limitations. Additional Intel publications the reader may wish to reference are the 8086 User's Manual (9800722A), 8086 Assembly Language Reference Guide (9800749A), AP-28A MULTIBUS™ Interfacing (98005876B), INTEL MULTIBUS® SPECIFICATION (9800683), AP-45 Using the 8202 Dynamic RAM Controller (9800809A), AP-51 Designing 8086, 8088, 8089 Multiprocessor Systems with the 8289 Bus Arbiter and AP-59 Using the 8259A Programmable Interrupt Controller. References to other Intel publications will be made throughout this note.

2. 8086 OVERVIEW AND BASIC SYSTEM CONCEPTS

2A. 8086 Bus Cycle Definition

The 8086 is a true 16-bit microprocessor with 16-bit internal and external data paths, one megabyte of memory address space (2**20) and a separate 64K byte (2**16) I/O address space. The CPU communicates with its external environment via a twenty-bit time multiplexed address, status and data bus and a command bus. To transfer data or fetch instructions, the CPU executes a bus cycle (Fig. 2A1). The minimum bus cycle consists of four CPU clock cycles called T states. During the first T state (T1), the CPU asserts an address on the twenty-bit

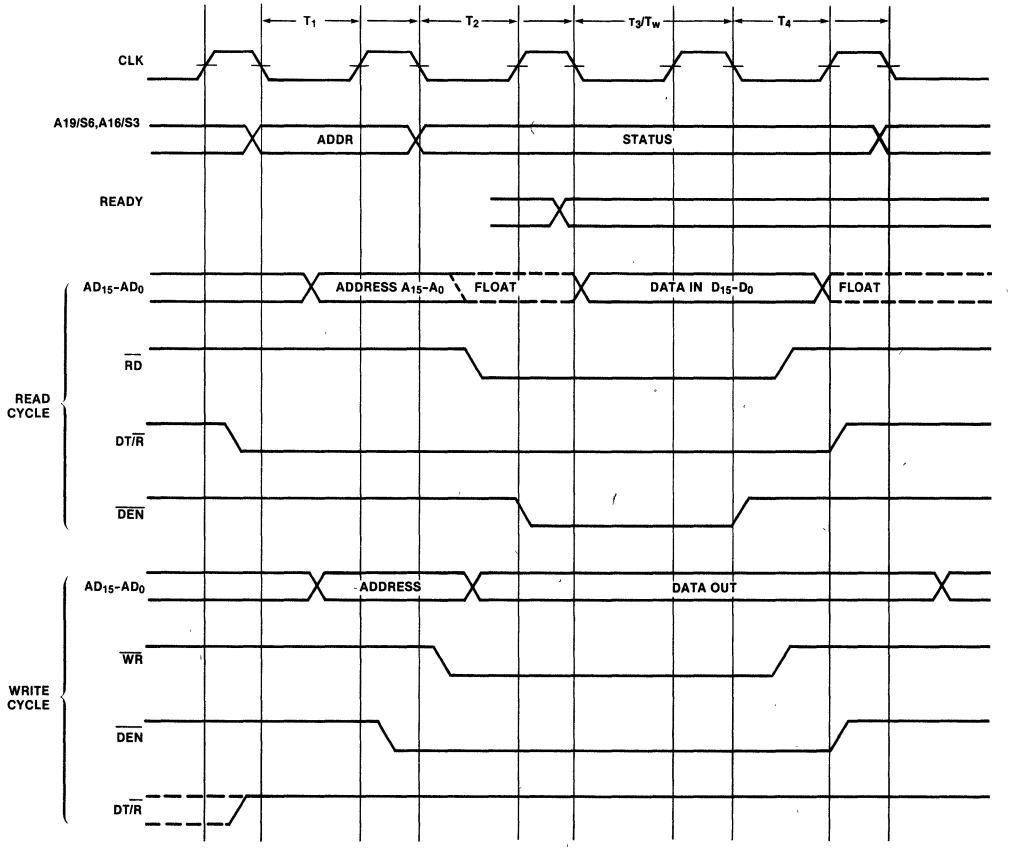


Figure 2A1. Basic 8086 Bus Cycle

multiplexed address/data/status bus. For the second T state (T2), the CPU removes the address from the bus and either three-states its outputs on the lower sixteen bus lines in preparation for a read cycle or asserts write data. Data bus transceivers are enabled in either T1 or T2 depending on the 8086 system configuration and the direction of the transfer (into or out of the CPU). Read, write or interrupt acknowledge commands are always enabled in T2. The maximum mode 8086 configuration (to be discussed later) also provides a write command enabled in T3 to guarantee data setup time prior to command activation.

During T2, the upper four multiplexed bus lines switch from address (A19-A16) to bus cycle status (S6,S5,S4,S3). The status information (Table 2A1) is available primarily for diagnostic monitoring. However, a decode of S3 and S4 could be used to select one of four banks of memory, one assigned to each segment register. This technique allows partitioning the memory by segment to expand the memory addressing beyond one megabyte. It also provides a degree of protection by preventing erroneous write operations to one segment from overlapping into another segment and destroying information in that segment.

The CPU continues to provide status information on the upper four bus lines during T3 and will either continue to assert write data or sample read data on the lower sixteen bus lines. If the selected memory or I/O device is not capable of transferring data at the maximum CPU transfer rate, the device must signal the CPU "not ready" and force the CPU to insert additional clock cycles (Wait states TW) after T3. The 'not ready' indication must be presented to the CPU by the start of T3. Bus activity during TW is the same as T3. When the selected device has had sufficient time to complete the transfer, it asserts "Ready" and allows the CPU to continue from the TW states. The CPU will latch the data on the bus during the last wait state or during T3 if no wait states are requested. The bus cycle is terminated in T4 (command lines are disabled and the selected external device deselects from the bus). The bus cycle appears to devices in the system as an asynchronous event consisting of an address to select the device followed by a read strobe or data and a write strobe. The selected device accepts bus data during a write cycle and drives the desired data onto the bus during a read cycle. On termination of the command, the device latches write data or disables its bus drivers. The only control the device has on the bus cycle is the insertion of wait cycles.

The 8086 CPU only executes a bus cycle when instructions or operands must be transferred to or from memory or I/O devices. When not executing a bus cycle, the bus interface executes idle cycles (TI). During the idle cycles, the CPU continues to drive status information from the previous bus cycle on the upper address lines. If the previous bus cycle was a write, the CPU continues to drive the write data onto the multiplexed bus until the start of the next bus cycle. If the CPU executes idle cycles following a read cycle, the CPU will not drive the lower 16 bus lines until the next bus cycle is required.

Since the CPU prefetches up to six bytes of the instruction stream for storage and execution from an internal instruction queue, the relationship of instruction fetch and associated operand transfers may be skewed in time and separated by additional instruction fetch bus cycles. In general, if an instruction is fetched into the 8086's internal instruction queue, several additional instructions may be fetched before the instruction is removed from the queue and executed. If the instruction being executed from the queue is a jump or other control transfer instruction, any instructions remaining in the queue are not executed and are discarded with no effect on the CPU's operation. The bus activity observed during execution of a specific instruction is dependent on the preceding instructions but is always deterministic within the specific sequence.

Table 2A1

S3	S4	
0	0	Alternate (relative to the ES segment)
1	0	Stack (relative to the SS segment)
0	1	Code/None (relative to the CS segment or a default of zero)
1	1	Data (relative to the DS segment)

S5 = IF (interrupt enable flag)
S6 = 0 (indicates the 8086 is on the bus)

2B. 8086 Address and Data Bus Concepts

Since the majority of system memories and peripherals require a stable address for the duration of the bus cycle, the address on the multiplexed address/data bus during T1 should be latched and the latched address used to select the desired peripheral or memory location. Since the 8086 has a 16-bit data bus, the multiplexed bus components of the 8085 family are not applicable to the 8086 (a device on address/data bus lines 8-15 will not be able to receive the byte selection address on lines 0-7). To demultiplex the bus (Fig. 2B1a), the 8086 system provides an Address Latch Enable signal (ALE) to capture the address in either the 8282 or 8283 8-bit bi-stable latches (Diag. 2B1). The latches are either inverting (8283) or non-inverting (8282) and have outputs driven by three-state buffers that supply 32 mA drive capability and can switch a 300 pF capacitive load in 22 ns (inverting) or 30 ns (non-inverting). They propagate the address through to the outputs while ALE is high and latch the address on the falling edge of ALE. This only delays address access and chip select decoding by the propagation delay of the latch. The outputs are enabled through the low active $\overline{OE}$ input. The demultiplexing of the multiplexed address/data bus (latchings of the address from the multiplexed bus), can be done locally at appropriate points in the system or at the CPU with a separate address bus distributing the address throughout the system (Fig. 2B1b). For optimum system performance and compatibility with multiprocessor and MULTIBUS™ configurations, the latter technique is strongly recommended over the first. The remainder of this note will assume the bus is demultiplexed at the CPU.

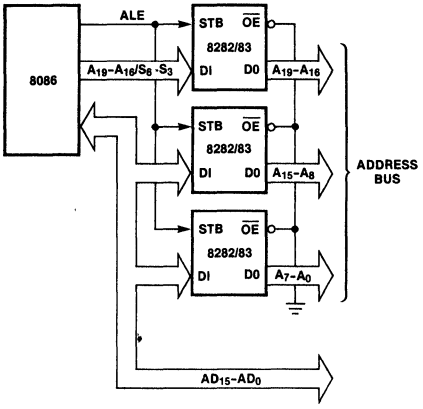


Figure 2B1a. Demultiplexing the 8086 Bus

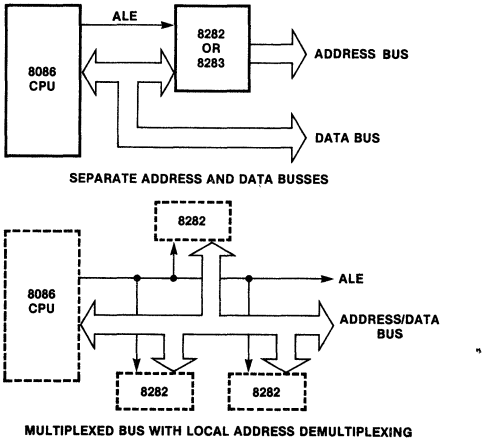


Figure 2B1b.

The programmer views the 8086 memory address space as a sequence of one million bytes in which any byte may contain an eight bit data element and any two consecutive bytes may contain a 16-bit data element. There is no constraint on byte or word addresses (boundaries). The address space is physically implemented on a sixteen bit data bus by dividing the address space into two banks of up to 512K bytes (Fig. 2B2). One bank is connected to the lower half of the sixteen-bit data bus (D7-0) and contains even addressed bytes (A0 = 0). The other bank is connected to the upper half of the data bus (D15-8) and contains odd addressed bytes (A0 = 1). A specific byte within each bank is selected by address lines A19-A1. To perform byte transfers to even addresses (Fig. 2B3a), the information is transferred over the lower half of the data bus (D7-0). A0 (active low) is used to enable the bank connected to the lower half of the data bus to participate in the transfer. Another signal provided by the 8086, Bus High Enable ($\overline{BHE}$), is used to disable the bank on the upper half of the data bus from participating in the transfer. This is necessary to prevent a write operation to the lower bank from destroying data in the upper bank. Since $\overline{BHE}$ is a multiplexed signal with timing identical to the A19-A16 address lines, it also should be latched with ALE to provide a stable signal during the bus cycle. During T2 through T4, the $\overline{BHE}$ output is multiplexed with status line S7 which is equal to $\overline{BHE}$. To perform byte transfers to odd addresses (Fig. 2B3b), the information is transferred over the upper half of the data bus (D15-D8) while $\overline{BHE}$ (active low) enables the upper bank and A0 disables the lower bank. Directing the data transfer to the appropriate half of the data bus and activation of $\overline{BHE}$ and A0 is performed by the 8086, transparent to the programmer. As an example, consider loading a byte of data into the CL register (lower half of the CX register) from an odd addressed memory location (referenced over the upper half of the 16-bit data bus). The data is transferred into the 8086 over the upper 8 bits of the data bus, automatically redirected to the lower half of the 8086 internal 16-bit data path and stored into the CL register. This capability also allows byte I/O transfers with the AL register to be directed to I/O devices connected to either the upper or lower half of the 16-bit data bus.

To access even addressed sixteen bit words (two consecutive bytes with the least significant byte at an even

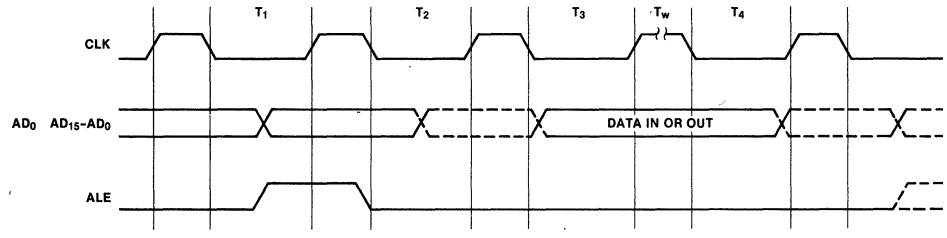


Diagram 2B1. ALE Timing

byte address), A19-A1 select the appropriate byte within each bank and A0 and $\overline{BHE}$ (active low) enable both banks simultaneously (Fig. 2B3c). To access an odd addressed 16-bit word (Fig. 2B3d), the least significant byte (addressed by A19-A1) is first transferred over the upper half of the bus (odd addressed byte, upper bank, $\overline{BHE}$ low active and A0 = 1). The most significant byte is accessed by incrementing the address (A19-A0) which allows A19-A1 to address the next physical word location (remember, A0 was equal to one which indicated a word referenced from an odd byte boundary). A second bus cycle is then executed to perform the transfer of the most significant byte with the lower bank (A0 is now active low and $\overline{BHE}$ is high). The sequence is automatically executed by the 8086 whenever a word transfer is executed to an odd address. Directing the upper and lower bytes of the 8086's internal sixteen-bit registers to the appropriate halves of the data bus is also performed automatically by the 8086 and is transparent to the programmer.

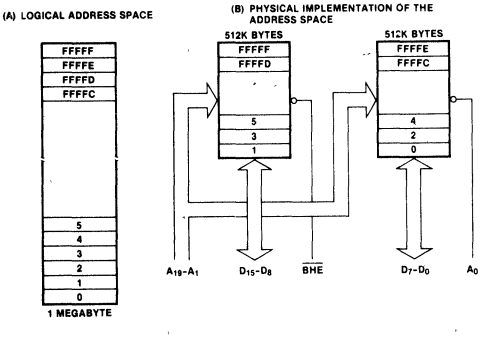


Figure 2B2. 8086 Memory

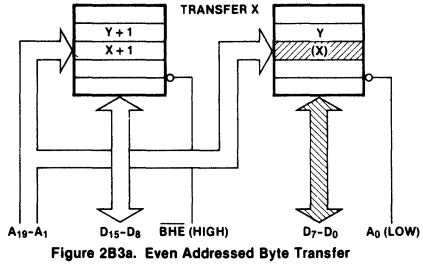


Figure 2B3a. Even Addressed Byte Transfer

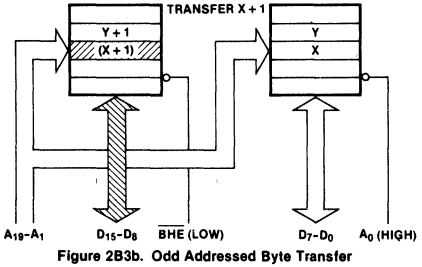


Figure 2B3b. Odd Addressed Byte Transfer

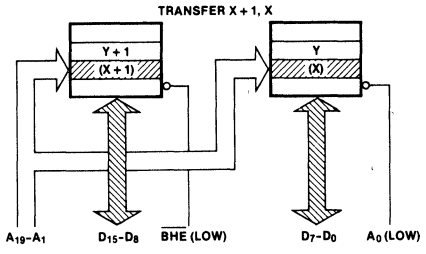


Figure 2B3c. Even Addressed Word Transfer

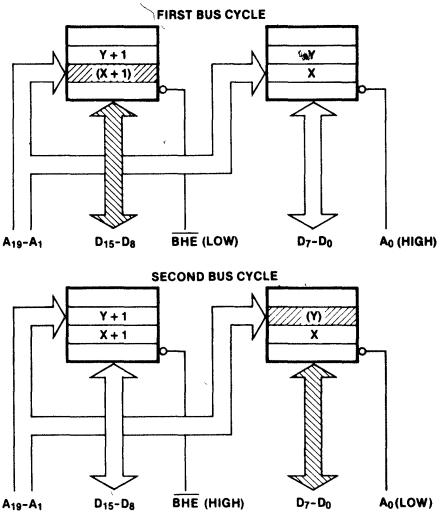


Figure 2B3d. Odd Addressed Word Transfer

During a byte read, the CPU floats the entire sixteen-bit data bus even though data is only expected on the upper or lower half of the data bus. As will be demonstrated later, this action simplifies the chip select decoding requirements for read only devices (ROM, EPROM). During a byte write operation, the 8086 will drive the entire sixteen-bit data bus. The information on the half of the data bus not transferring data is indeterminate. These concepts also apply to the I/O address space. Specific examples of I/O and memory interfacing are considered in the corresponding sections.

2C. System Data Bus Concepts

When referring to the system data bus, two implementation alternatives must be considered; (a) the multiplexed address/data bus (Fig. 2C1a) and a data bus buffered from the multiplexed bus by transceivers (Fig. 2C1b).

If memory or I/O devices are connected directly to the multiplexed bus, the designer must guarantee the devices do not corrupt the address on the bus during T1.

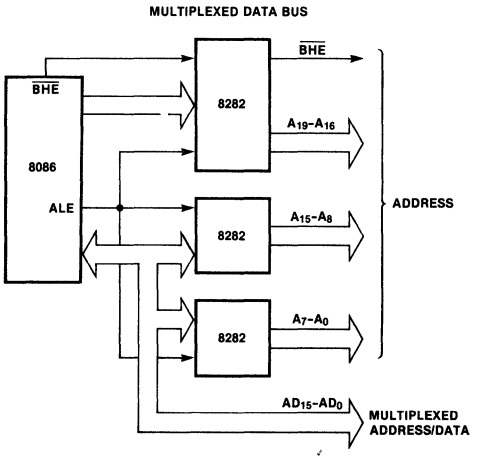


Figure 2C1a. Multiplexed Data Bus

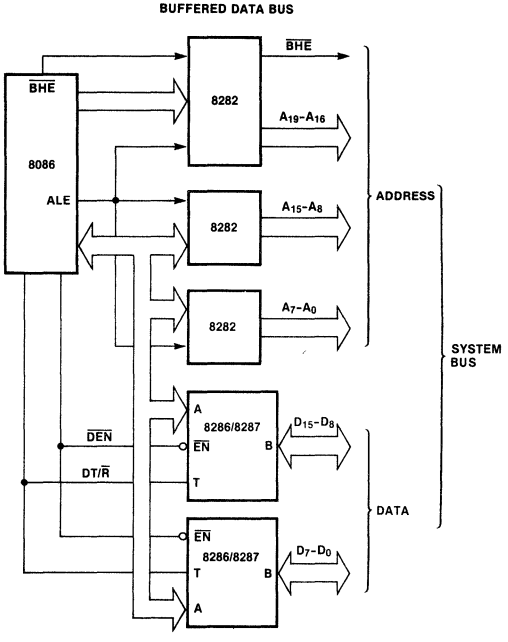


Figure 2C1b. Buffered Data Bus

To avoid this, device output drivers should not be enabled by the device chip select, but should have an output enable controlled by the system read signal (Fig. 2C2). The 8086 timing guarantees that read is not valid until after the address is latched by ALE (Diag. 2C1). All Intel peripherals, EPROM products and RAM's for microprocessors provide output enable or read inputs to allow connection to the multiplexed bus.

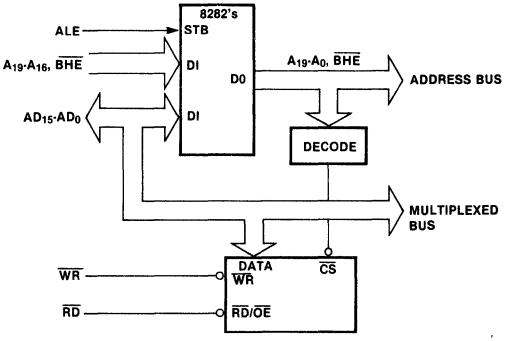


Figure 2C2. Devices with Output Enables on the Multiplexed Bus

Several techniques are available for interfacing devices without output enables to the multiplexed bus but each introduces other restrictions or limitations. Consider Figure 2C3 which has chip select gated with read and write. Two problems exist with this technique. First, the chip select access time is reduced to the read access time, and may require a faster device if maximum system performance (no wait states) is to be achieved (Diag. 2C2). Second, the designer must verify that chip select to write setup and hold times for the device are not violated (Diag. 2C3). Alternate techniques can be extracted from the bus interfacing techniques given later in this section but are subject to the associated restrictions. In general, the best solution is obtained with devices having output enables.

A subsequent limitation on the multiplexed bus is the 8086's drive capability of 2.0 mA and capacitive loading of 100 pF to guarantee the specified A.C. characteristics. Assuming capacitive loads of 20 pF per I/O device, 12 pF per address latch and 5-12 pF per memory device, a system mix of three peripherals and two to four memory devices (per bus line) are close to the loading limit.

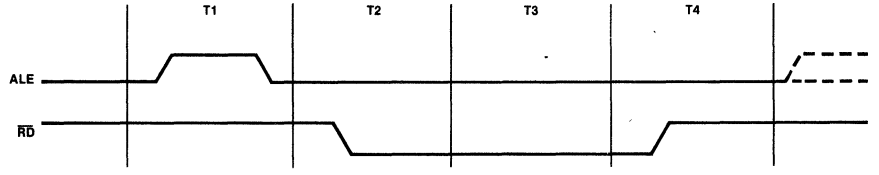


Diagram 2C1. Relationship of ALE to READ

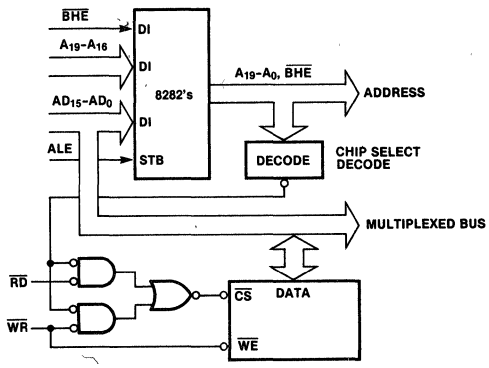
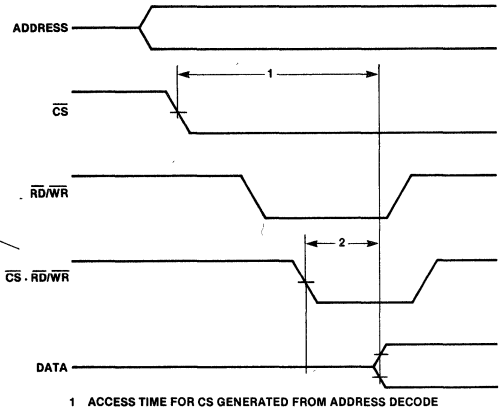


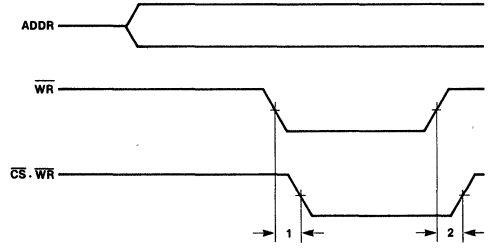
Figure 2C3. Devices without Output Enables on the Multiplexed Bus



1 ACCESS TIME FOR CS GENERATED FROM ADDRESS DECODE

2 ACCESS TIME IF CS IS GATED WITH RD/WR.

Diagram 2C2. Access Time: CS Gated with $\overline{RD}/\overline{WR}$



1 CS IS NOT VALID PRIOR TO WRITE AND BECOMES ACTIVE ONE OR TWO GATE DELAYS LATER.

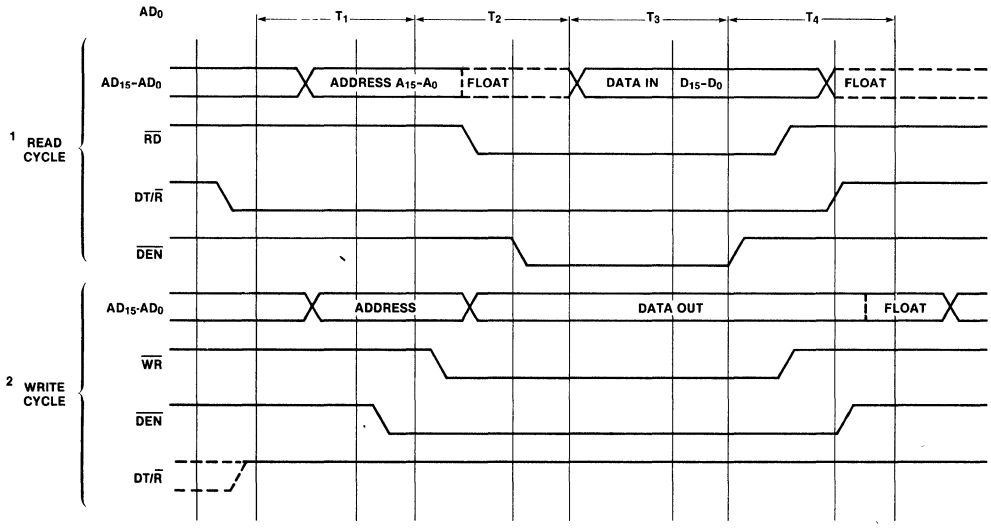
2 CS REMAINS VALID AFTER WRITE ONE OR TWO GATE DELAYS.

Diagram 2C3. CS to $\overline{WR}$ Set-Up and Hold

To satisfy the capacitive loading and drive requirements of larger systems, the data bus must be buffered. The 8286 non-inverting and 8287 inverting octal transceivers are offered as part of the 8086 family to satisfy this requirement. They have three-state output buffers that drive 32 mA on the bus interface and 10 mA on the CPU interface and can switch capacitive loads of 300 pF at the bus interface and 100 pF on the CPU interface in 22 ns (8287) or 30 ns (8286). To enable and control the direction of the transceivers, the 8086 system provides Data ENable (DEN) and Data Transmit/Receive ($DT/\overline{R}$) signals (Fig. 2C1b). These signals provide the appropriate timing to guarantee isolation of the multiplexed bus from the system during T1 and elimination of bus contention with the CPU during read and write (Diag. 2C4). Although the memory and peripheral devices are isolated from the CPU (Fig. 2C4), bus contention may still exist in the system if the devices do not have an output enable control other than chip select. As an example, bus contention will exist during transition from one chip select to another (the newly selected device begins driving the bus before the previous device has disabled its drivers). Another, more severe case exists during a write cycle. From chip select to write active, a device whose outputs are controlled only by chip select, will drive the bus simultaneously with write data being driven through the transceivers by the CPU (Diag. 2C5). The same technique given for circumventing these problems on the multiplexed bus can be applied here with the same limitations.

One last extension to the bus implementation is a second level of buffering to reduce the total load seen by devices on the system bus (Fig. 2C5). This is typically done for multiboard systems and isolation of memory arrays. The concerns with this configuration are the additional delay for access and more important, control of the second transceiver in relationship to the system bus and the device being interfaced to the system bus. Several techniques for controlling the transceiver are given in Figure 2C6. This first technique (Fig. 2C6a) simply distributes DEN and $DT/\overline{R}$ throughout the system. $DT/\overline{R}$ is inverted to provide proper direction control for the second level transceivers. The second example (Fig. 2C6b) provides control for devices with output enables. $\overline{RD}$ is used to normally direct data from the system bus to the peripheral. The buffer is selected whenever a device on the local bus is chip selected. Bus contention is possible on the device's local bus during a read as the read simultaneously enables the device output and changes the transceiver direction. The contention may also occur as the read is terminated.

For devices without output enables, the same technique can be applied (Fig. 2C6c) if the chip select to the device is conditioned by read or write. Controlling the chip select with read/write prevents the device from driving against the transceiver prior to the command being received. The limitations with this technique are access limited to read/write time and limited CS to write setup and hold times.



1 $\overline{DEN}$ IS ENABLED AFTER THE 8086 HAS FLOATED THE MULTIPLEXED BUS

2 $\overline{DEN}$ ENABLES THE TRANSCEIVERS EARLY IN THE CYCLE, BUT DT/R GUARANTEES THE TRANSCEIVERS ARE IN TRANSMIT RATHER THAN RECEIVE MODE AND WILL NOT DRIVE AGAINST THE CPU.

Diagram 2C4. Bus Transceiver Control

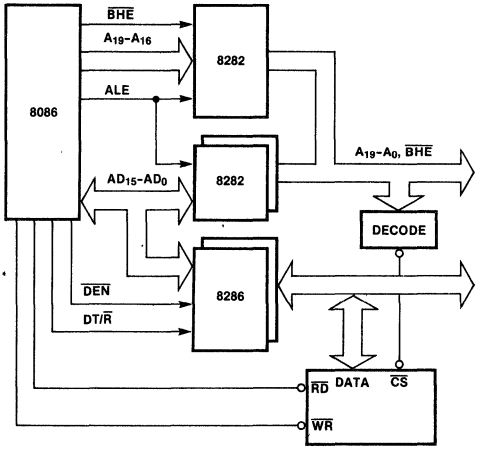


Figure 2C4. Devices with Output Enables on the System Bus

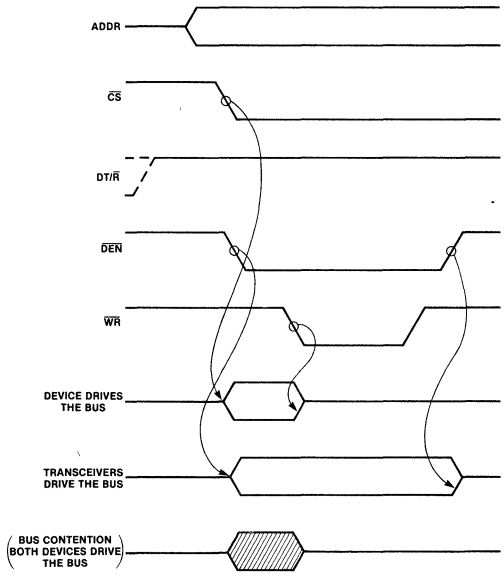


Diagram 2C5.

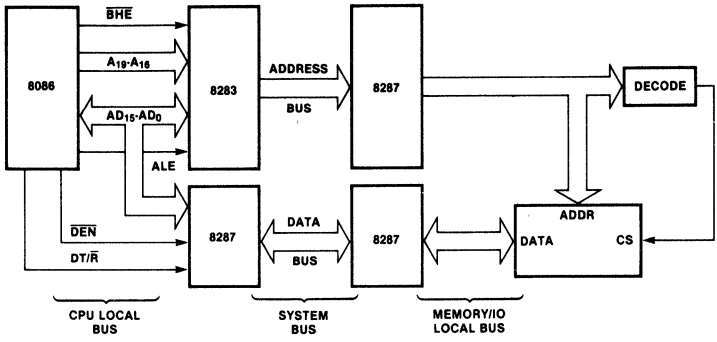


Figure 2C5. Fully Buffered System

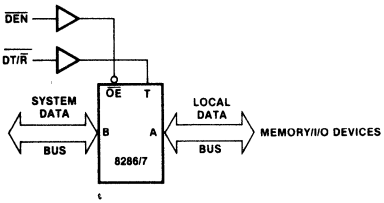


Figure 2C6a. Controlling System Transceivers with DEN and DT/R

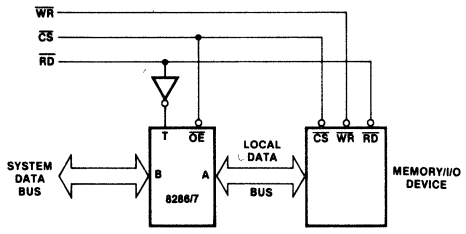


Figure 2C6b. Buffering Devices with OE/RD

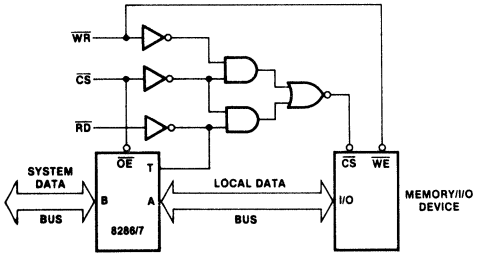


Figure 2C6c. Buffering Devices without OE/RD and with Common or Separate Input/Output

An alternate technique applicable to devices with and without output enables is shown in Figure 2C6d. RD again controls the direction of the transceiver but it is not enabled until a command and chip select are active. The possibility for bus contention still exists but is reduced to variations in output enable vs. direction change time for the transceiver. Full access time from chip select is now available, but data will not be valid prior to write and will only be held valid after write by the delay to disable the transceiver.

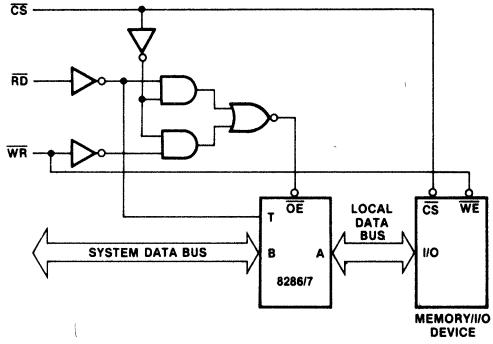


Figure 2C6d. Buffering Devices without OE/RD and with Common or Separate Input/Output

One last technique is given for devices with separate inputs and outputs (Fig. 2C6e). Separate bus receivers and drivers are provided rather than a single transceiver. The receiver is always enabled while the bus driver is controlled by RD and chip select. The only possibility for bus contention in this system occurs as multiple devices on each line of the local read bus are enabled and disabled during chip selection changes.

Throughout this note, the multiplexed bus will be considered the local CPU bus and the demultiplexed address and buffered data bus will be the system bus. For additional information on bus contention and the system problems associated with it, refer to Appendix 1.

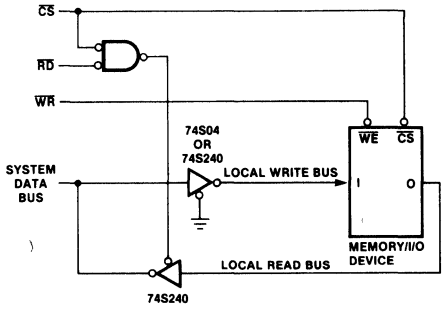


Figure 2C6e. Buffering Devices without $\overline{OE}/\overline{RD}$ and with Separate Input/Output

2D. Multiprocessor Environment

The 8086 architecture supports multiprocessor systems based on the concept of a shared system bus (Fig. 2D1). All CPU's in the system communicate with each other and share resources via the system bus. The bus may be either the Intel Multibus™ system bus or an extension of the system bus defined in the previous section. The major addition required to the demultiplexed system bus is arbitration logic to control access to the system bus. As each CPU asynchronously requests access to the shared bus, the arbitration logic resolves priorities and grants bus access to the highest priority CPU. Having gained access to the bus, the CPU completes its transfer and will either relinquish the bus or wait to be forced to relinquish the bus. For a discussion on Multibus™ arbitration techniques, refer to AP-28A, Intel Multibus™ Interfacing.

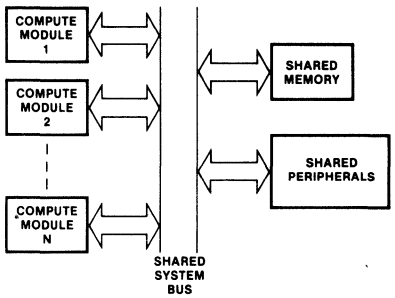


Figure 2D1. 8086 Family Multiprocessor System

To support a multimaster interface to the Multibus system bus for the 8086 family, the 8289 bus arbiter is included as part of the family. The 8289 is compatible with the 8086's local bus and in conjunction with the 8288 bus controller, implements the Multibus protocol for bus arbitration. The 8289 provides a variety of arbitration and prioritization techniques to allow optimization of bus availability, throughput and utilization of shared resources. Additional features (implemented through strapping options) extend the configuration options beyond a pure CPU interface to the multimaster system bus for access to shared resources to include concurrent support of a local CPU bus for private resources. For specific configurations and additional information on the 8289, refer to application note AP-51.

3. 8086 SYSTEM DETAILS

3A. Operating Modes

Possibly the most unique feature of the 8086 is the ability to select the base machine configuration most suited to the application. The MN/$\overline{MX}$ input to the 8086 is a strapping option which allows the designer to select between two functional definitions of a subset of the 8086 outputs.

MINIMUM MODE

The minimum mode 8086 (Fig. 3A1) is optimized for small to medium (one or two boards), single CPU systems. Its system architecture is directed at satisfying the requirements of the lower to middle segment of high performance 16-bit applications. The CPU maintains the full megabyte memory space, 64K byte I/O space and 16-bit data path. The CPU directly provides all bus control (DT/$\overline{R}$, $\overline{DEN}$, ALE, M/$\overline{IO}$), commands ($\overline{RD}$,$\overline{WR}$,$\overline{INTA}$) and a simple CPU preemption mechanism (HOLD, HLDA) compatible with existing DMA controllers.

MAXIMUM MODE

The maximum mode (Fig. 3A2) extends the system architecture to support multiprocessor configurations, and local instruction set extension processors (co-processors). Through addition of the 8288 bipolar bus controller, the 8086 outputs assigned to bus control and commands in the minimum mode are redefined to allow these extensions and enhance general system performance. Specifically, (1) two prioritized levels of processor preemption ($\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$) allow multiple processors to reside on the 8086's local bus and share its interface to the system bus, (2) Queue status (QS0,QS1) is available to allow external devices like ICE™-86 or special instruction set extension co-processors to track the CPU instruction execution, (3) access control to shared resources in multiprocessor systems is supported by a hardware bus lock mechanism and (4) system command and configuration options are expanded via ancillary devices like the 8288 bus controller and 8289 bus arbiter.

The queue status indicates what information is being removed from the internal queue and when the queue is being reset due to a transfer of control (Table 3A1). By monitoring the $\overline{S0}$,$\overline{S1}$,$\overline{S2}$ status lines for instructions entering the 8086 (1,0,0 indicates code access while A0 and $\overline{BHE}$ indicate word or byte) and QS0, QS1 for instructions leaving the 8086's internal queue, it is possible to track the instruction execution. Since instructions are executed from the 8086's internal queue, the queue status is presented each CPU clock cycle and is not related to the bus cycle activity. This mechanism (1) allows a co-processor to detect execution of an

ESCAPE instruction which directs the co-processor to perform a specific task and (2) allows ICE-86 to trap execution of a specific memory location. An example of a circuit used by ICE is given in Figure 3A3. The first up down counter tracks the depth of the queue while the second captures the queue depth on a match. The second counter decrements on further fetches from the queue until the queue is flushed or the count goes to zero indicating execution of the match address. The first counter decrements on fetch from the queue (QS0 = 1) and increments on code fetches into the queue. Note that a normal code fetch will transfer two bytes into the queue so two clock increments are given to the counter (T201 and T301) unless a single byte is loaded over the upper half of the bus (A0-P is high). Since the execution unit (EU) is not synchronized to the bus interface unit (BIU), a fetch from the queue can occur simultaneously with a transfer into the queue. The exclusive-or gate driving the ENP input of the first counter allows these simultaneous operations to cancel each other and not modify the queue depth.

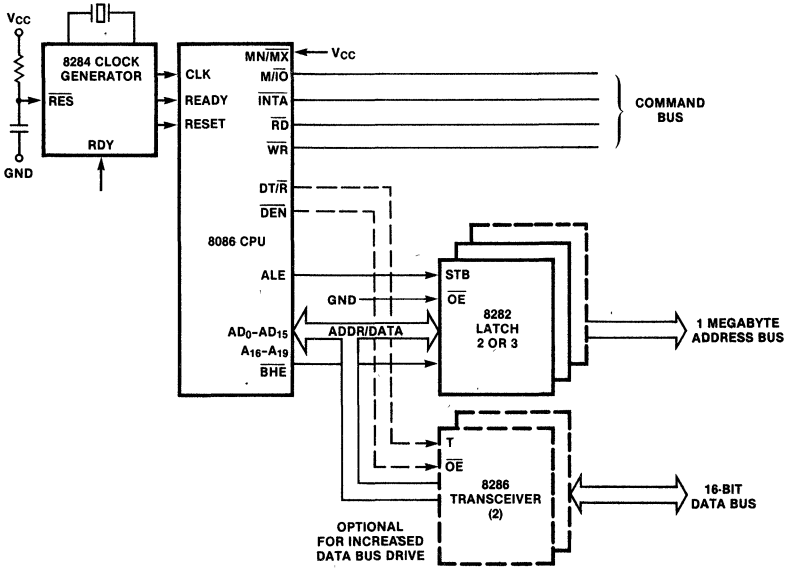


Figure 3A1. Minimum Mode 8086

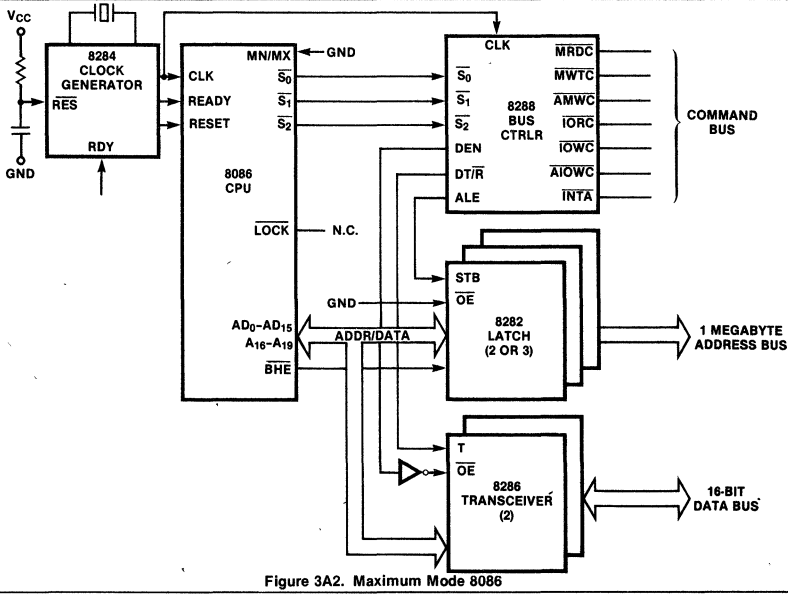


Figure 3A2. Maximum Mode 8086

TABLE 3A1. QUEUE STATUS

QS_1	QS_0	
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 queue status is valid during the CLK cycle after which the queue operation is performed.

To address the problem of controlling access to shared resources, the maximum mode 8086 provides a hardware $\overline{LOCK}$ output. The $\overline{LOCK}$ output is activated through the instruction stream by execution of the LOCK prefix instruction. The $\overline{LOCK}$ output goes active in the first CPU clock cycle following execution of the prefix and remains active until the clock following the completion of the instruction following the LOCK prefix. To provide bus access control in multiprocessor systems, the $\overline{LOCK}$ signal should be incorporated into the system bus arbitration logic resident to the CPU.

During normal multiprocessor system operation, priority of the shared system bus is determined by the arbitration circuitry on a cycle by cycle basis. As each CPU requires a transfer over the system bus, it requests access to the bus via its resident bus arbitration logic. When the CPU gains priority (determined by the system bus arbitration scheme and any associated logic), it takes control of the bus, performs its bus cycle and either maintains bus control, voluntarily releases the bus or is forced off the bus by the loss of priority. The lock mechanism prevents the CPU from losing bus control (either voluntarily or by force) and guarantees a CPU the ability to execute multiple bus cycles (during execution of the locked instruction) without intervention and possible corruption of the data by another CPU. A classic use of the mechanism is the 'TEST and SET semaphore' during which a CPU must read from a shared memory location and return data to the location without allowing another CPU to reference the same location between the TEST operation (read) and the SET operation (write). In the 8086 this is accomplished with a locked exchange instruction.

```
LOCK XCHG reg, MEMORY ; reg is any register
                      ;MEMORY is the address of the
                      ;semaphore
```

The activity of the $\overline{LOCK}$ output is shown in Diagram 3A1. Another interesting use of the $\overline{LOCK}$ for multiprocessor systems is a locked block move which allows high speed message transfer from one CPU's message buffer to another.

During the locked instruction, a request for processor preemption ($\overline{RQ}/\overline{GT}$) is recorded but not acknowledged until completion of the locked instruction. The $\overline{LOCK}$ has no direct affect on interrupts. As an example, a locked HALT instruction will cause HOLD (or $\overline{RQ}/\overline{GT}$) requests to be ignored but will allow the CPU to exit the HALT state on an interrupt. In general, prefix bytes are considered extensions of the instructions they precede. Therefore, interrupts that occur during execution of a prefix are not acknowledged (assuming interrupts are enabled) until completion of the instruction following the prefixes (except for instructions which allow servicing interrupts during their execution, i.e., HALT, WAIT and repeated string primitives). Note that multiple prefix bytes may precede an instruction. As another example, consider a 'string primitive' preceded by the repetition

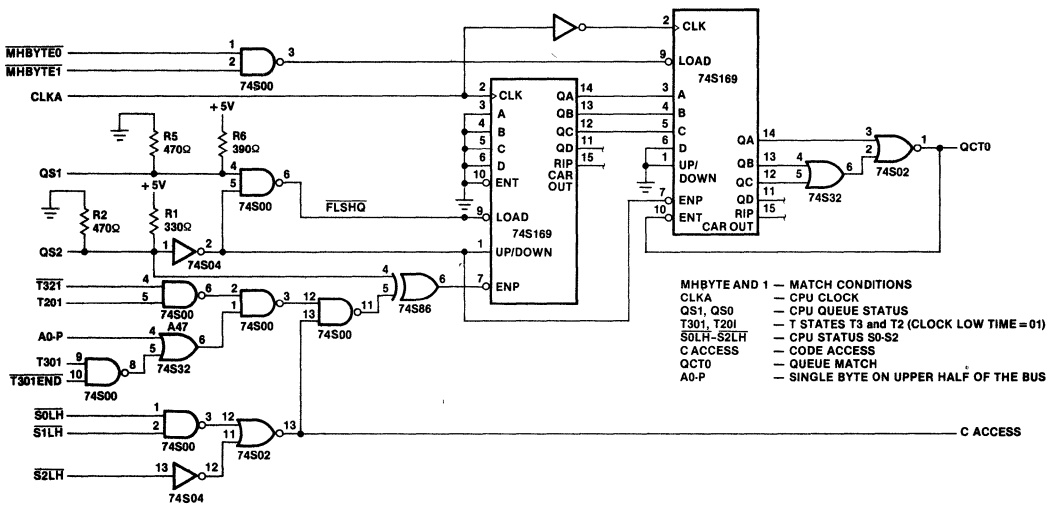


Figure 3A3. Example Circuit to Track the 8086 Queue

prefix (REP) which is interruptible after each execution of the string primitive. This holds even if the REP prefix is combined with the LOCK prefix and prevents interrupts from being locked out during a block move or other repeated string operation. As long as the operation is not interrupted, $\overline{LOCK}$ remains active. Further information on the operation of an interrupted string operation with multiple prefixes is presented in the section dealing with the 8086 interrupt structure.

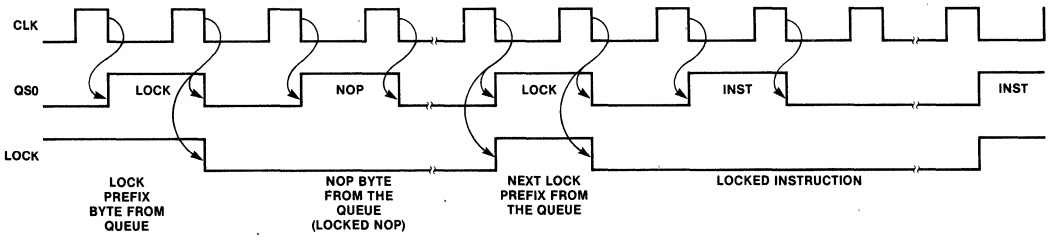
Three additional status lines ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$) are defined to provide communications with the 8288 and 8289. The status lines tell the 8288 when to initiate a bus cycle, what type of command to issue and when to terminate the bus cycle. The 8288 samples the status lines at the beginning of each CPU clock (CLK). To initiate a bus cycle, the CPU drives the status lines from the passive state ($\overline{S0}$, $\overline{S1}$, $\overline{S2} = 1$) to one of seven possible command codes (Table 3A2). This occurs on the rising edge of the clock during T4 of the previous bus cycle or a TI (idle cycle, no current bus activity). The 8288 detects the status change by sampling the status lines on the high to low transition of each clock cycle. The 8288 starts a bus cycle by generating ALE and appropriate buffer direction control in the clock cycle immediately following detection of the status change (T1). The bus transceivers and the selected command are enabled in the next clock cycle (T2) (or T3 for normal write commands). When the status returns to the passive state, the 8288 will terminate the command as shown in Diagram 3A2. Since the CPU will not return the status to the passive state until the 'ready' indication is received, the 8288 will maintain active command and bus control for any number of wait cycles. The status lines may also be used by other processors on the 8086's local bus to monitor bus activity and control the 8288 if they gain control of the local bus.

TABLE 3A2. STATUS LINE DECODES

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
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

The 8288 provides the bus control (DEN, DT/$\overline{R}$, ALE) and commands ($\overline{INTA}$, $\overline{MRDC}$, $\overline{IORC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IOWC}$, $\overline{AIOWC}$) removed from the CPU. The command structure has separate read and write commands for memory and I/O to provide compatibility with the Multibus command structure.

The advanced write commands are enabled one clock period earlier than the normal write to accommodate the wider write pulse widths often required by peripherals and static RAMs. The normal write provides data setup prior to write to accommodate dynamic RAM memories and I/O devices which strobe data on the leading edge of write. The advanced write commands do not guarantee that data is valid prior to the leading edge of the command. The DEN signal in the maximum mode is inverted from the minimum mode to extend transceiver control by allowing logical conjunction of DEN with other signals. While not appearing to be a significant benefit in the basic maximum mode configuration, introduction of interrupt control and various system configurations will demonstrate the usefulness of qualifying DEN. Diagram 3A3 compares the timing of the minimum and maximum mode bus transfer commands. Although the



1 QUEUE STATUS INDICATES FIRST BYTE OF OPCODE FROM THE QUEUE.

2 THE $\overline{LOCK}$ OUTPUT WILL GO INACTIVE BETWEEN SEPARATE LOCKED INSTRUCTIONS.

3 TWO CLOCKS ARE REQUIRED FOR DECODE OF THE LOCK PREFIX AND ACTIVATION OF THE $\overline{LOCK}$ SIGNAL.

4 SINCE QUEUE STATUS REFLECTS THE QUEUE OPERATION IN THE PREVIOUS CLOCK CYCLE, THE $\overline{LOCK}$ OUTPUT ACTUALLY GOES ACTIVE COINCIDENT WITH THE START OF THE NEXT INSTRUCTION AND REMAINS ACTIVE FOR ONE CLOCK CYCLE FOLLOWING THE INSTRUCTION.

5 IF THE INSTRUCTION FOLLOWING THE LOCK PREFIX IS NOT IN THE QUEUE, THE $\overline{LOCK}$ OUTPUT STILL GOES ACTIVE AS SHOWN WHILE THE INSTRUCTION IS BEING FETCHED.

6 THE BIU WILL STILL PERFORM INSTRUCTION FETCH CYCLES DURING EXECUTION OF A LOCKED INSTRUCTION. THE $\overline{LOCK}$ MERELY LOCKS THE BUS TO THIS CPU FOR WHATEVER BUS CYCLES THE CPU PERFORMS DURING THE LOCKED INSTRUCTION.

Diagram 3A1. 8086 Lock Activity

maximum mode configuration is designed for multiprocessor environments, large single CPU designs (either Multibus systems or greater than two PC boards) should also use the maximum mode. Since the 8288 is a bipolar dedicated controller device, its output drive for the commands (32 mA) and tolerances on AC characteristics (timing parameters and worse case delays) provide better large system performance than the minimum mode 8086.

In addition to assuming the functions removed from the CPU, the 8288 provides additional strapping options and controls to support multiprocessor configurations and peripheral devices on the CPU local bus. These capabilities allow assigning resources (memory or I/O) as shared (available on the Multibus system bus) or private (accessible only by this CPU) to reduce contention for access to the Multibus system bus and improve multi-CPU system performance. Specific configuration possibilities are discussed in AP-51.

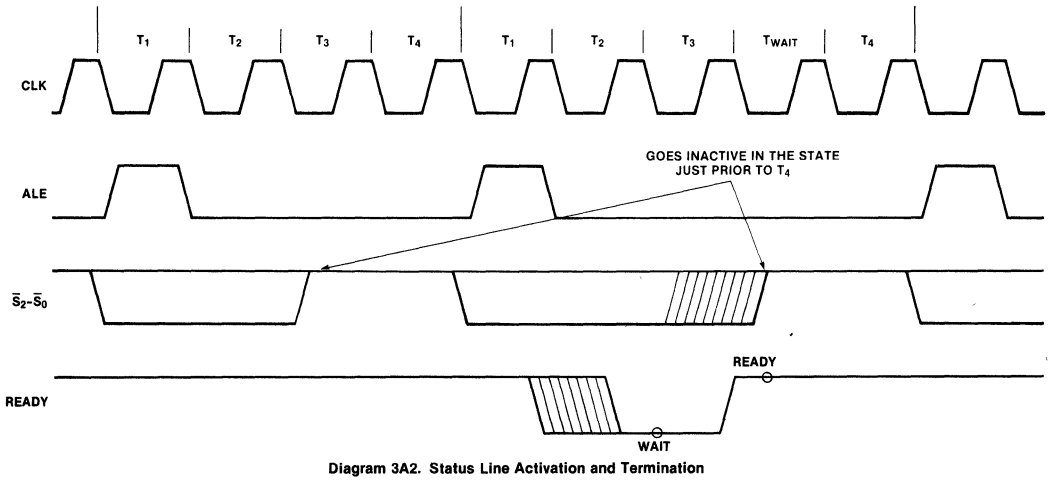


Diagram 3A2. Status Line Activation and Termination

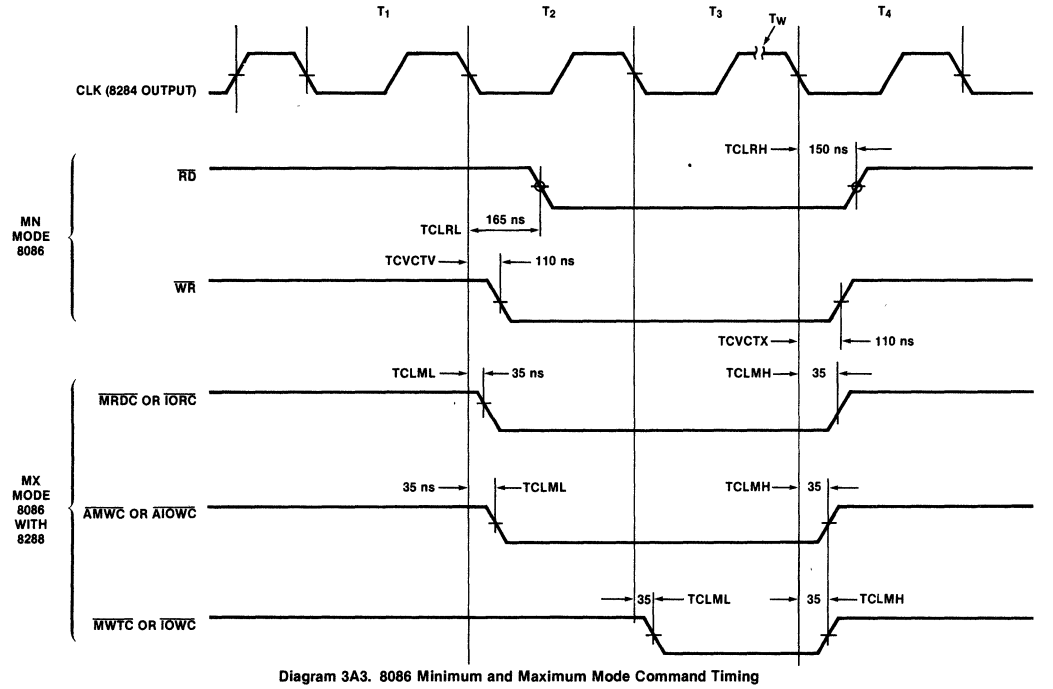


Diagram 3A3. 8086 Minimum and Maximum Mode Command Timing

3B. Clock Generation

The 8086 requires a clock signal with fast rise and fall times (10 ns max) between low and high voltages of −0.5 to +0.6 low and 3.9 to VCC + 1.0 high. The maximum clock frequency of the 8086 is 5 MHz and 8 MHz for the 8086-2. Since the design of the 8086 incorporates dynamic cells, a minimum frequency of 2 MHz is required to retain the state of the machine. Due to the minimum frequency requirement, single stepping or cycling of the CPU may not be accomplished by disabling the clock. The timing and voltage requirements of the CPU clock are shown in Figure 3B1. In general, for frequencies below the maximum, the CPU clock need not satisfy the frequency dependent pulse width limitations stated in the 8086 data sheet. The values specified only reflect the minimum values which must be satisfied and are stated in terms of the maximum clock frequency. As the clock frequency approaches the maximum frequency of the CPU, the clock must conform to a 33% duty cycle to satisfy the CPU minimum clock low and high time specifications.

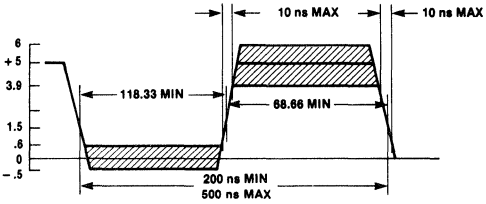


Figure 3B1. 8086 Clock

An optimum 33% duty cycle clock with the required voltage levels and transition times can be obtained with the 8284 clock generator (Fig. 3B2). Either an external frequency source or a series resonant crystal may drive the 8284. The selected source must oscillate at 3X the desired CPU frequency. To select the crystal inputs of the 8284 as the frequency source for clock generation, the F/$\overline{C}$ input to the 8284 must be strapped to ground. The strapping option allows selecting either the crystal or the external frequency input as the source for clock generation. Although the 8284 provides an input for a tank circuit to accommodate overtone mode crystals, fundamental mode crystals are recommended for more accurate and stable frequency generation. When selecting a crystal for use with the 8284, the series resistance should be as low as possible. Since other circuit components will tend to shift the operating frequency from resonance, the operating impedance will typically be higher than the specified series resistance. If the attenuation of the oscillator's feedback circuit reduces the loop gain to less than one, the oscillator will fail. Since the oscillator delays in the 8284 appear as inductive elements to the crystal, causing it to run at a frequency below that of the pure series resonance, a capacitor should be placed in series with the crystal and the X2 input of the 8284. This capacitor serves to cancel this inductive element. The value of the capacitor (CL) must not cause the impedance of the feedback circuit to reduce the loop gain below one. The impedance of the capacitor is a function of the operating frequency and can be determined from the following equation:

$$XCL = 1/2\pi * F * CL$$

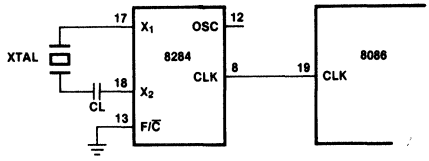


Figure 3B2. 8284 Clock Generator

It is recommended that the crystal series resistance plus XCL be kept less than 1K ohms. This capacitor also serves to debias the crystal and prevent a DC voltage bias from straining and perhaps damaging the crystalline structure. As the crystal frequency increases, the amount of capacitance should be decreased. For example, a 12 MHz crystal may require CL ~ 24 pF while 22 MHz may require CL ~ 8 pF. If very close correlation with the pure series resonance is not necessary, a nominal CL value of 12-15 pF may be used with a 15 MHz crystal (5 MHz 8086 operation). Board layout and component variances will affect the actual amount of inductance and therefore the series capacitance required to cancel it out (this is especially true for wire-wrapped layouts).

Two of the many vendors which supply crystals for Intel microprocessors are listed in Table 3B1 along with a list of crystal part numbers for various frequencies which may be of interest. For additional information on specifying crystals for Intel components refer to application note AP-35.

TABLE 3B1. CRYSTAL VENDORS

f	Parallel/ Series	Crystek[1] Corp.	CTS Knight,[2] Inc.
15.0 MHz	S	CY15A	MP150
18.432	S	CY19B*	MP184*
24.0 MHz	S	CY24A	MP240

*Intel also supplies a crystal numbered 8801 for this application.

Notes: 1. Address: 1000 Crystal Drive, Fort Meyers, Florida 33901
2. Address: 400 Reimann Ave., Sandwich, Illinois

If a high accuracy frequency source, externally variable frequency source or a common source for driving multiple 8284's is desired, the External Frequency Input (EFI) of the 8284 can be selected by strapping the F/$\overline{C}$ input to 5 volts through ~1K ohms (Fig. 3B3). The external frequency source should be TTL compatible, have a 50% duty cycle and oscillate at three times the desired CPU operating frequency. The maximum EFI frequency the 8284 can accept is slightly above 24 MHz with minimum clock low and high times of 13 ns. Although

no minimum EFI frequency is specified, it should not violate the CPU minimum clock rate. If a common frequency source is used to drive multiple 8284's distributed throughout the system, each 8284 should be driven by its own line from the source. To minimize noise in the system, each line should be a twisted pair driven by a buffer like the 74LS04 with the ground of the twisted pair connecting the grounds of the source and receiver. To minimize clock skew, the lines to all 8284's should be of equal length. A simple technique for generating a master frequency source for additional 8284's is shown in Figure 3B4. One 8284 with a crystal is used to generate the desired frequency. The oscillator output of the 8284 (OSC) equals the crystal frequency and is used to drive the external frequency to all other 8284's in the system.

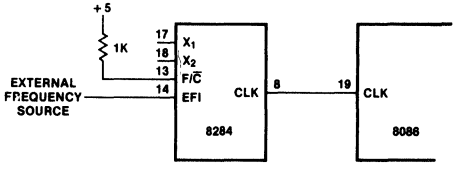


Figure 3B3. 8284 with External Frequency Source

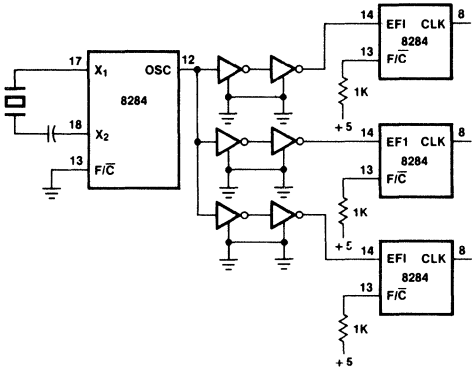


Figure 3B4. External Frequency for Multiple 8284s

The oscillator output is inverted from the oscillator signal used to drive the CPU clock generator circuit. Therefore, the oscillator output of one 8284 should not drive the EFI input of a second 8284 if both are driving clock inputs of separate CPU's that are to be synchronized. The variation on EFI to CLK delay over a range of 8284's may approach 35 to 45 ns. If, however, all 8284's are of the same package type, have the same relative supply voltage and operate in the same temperature environment, the variation will be reduced to between 15 and 25 ns.

There are three frequency outputs from the 8284, the oscillator (OSC) mentioned above, the system clock (CLK) which drives the CPU, and a peripheral clock (PCLK) that runs at one half the CPU clock frequency. The oscillator output is only driven by the crystal and is not affected by the F/$\overline{C}$ strapping option. If a crystal is not connected to the 8284 when the external frequency input is used, the oscillator output is indeterminate. The CPU clock is derived from the selected frequency source by an internal divide by three counter. The counter generates the 33% duty cycle clock which is optimum for the CPU at maximum frequency. The peripheral clock has a 50% duty cycle and is derived from the CPU clock. Diagram 3B0 shows the relationship of CLK to OSC and PCLK to CLK. The maximum skew is 20 ns between OSC and CLK, and 22 ns between CLK and PCLK.

Since the state of the 8284 divide by three counter is indeterminate at system initialization (power on), an external sync to the counter (CSYNC) is provided to allow synchronization of the CPU clock to an external event. When CSYNC is brought high, the CLK and PCLK outputs are forced high. When CSYNC returns low, the next positive clock from the frequency source starts clock generation. CSYNC must be active for a minimum of two periods of the frequency source. If CSYNC is asynchronous to the frequency source, the circuit in Figure 3B5 should be used for synchronization. The two latches minimize the probability of a meta-stable state in the latch driving CSYNC. The latches are clocked with the inverse of the frequency source to guarantee the 8284 setup and hold time of CSYNC to the frequency source (Diag. 3B1). If a single 8284 is to be synchronized to an external event and an external frequency source is not used, the oscillator output of the 8284 may be used to

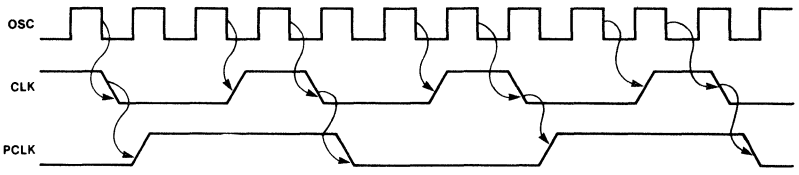


Diagram 3B0. OSC → CLK and CLK → PCLK Relationships

synchronize CSYNC (Fig. 3B6). Since the oscillator output is inverted from the internal oscillator signal, the inverter in the previous example is not required. If multiple 8284's are to be synchronized, an external frequency source must drive all 8284's and a single CSYNC synchronization circuit must drive the CSYNC input of all 8284's (Fig. 3B7). Since activation of CSYNC may cause violation of CPU minimum clock low time, it should only be enabled during reset or CPU clock high. CSYNC must also be disabled a minimum of four CPU clocks before the end of reset to guarantee proper CPU reset.

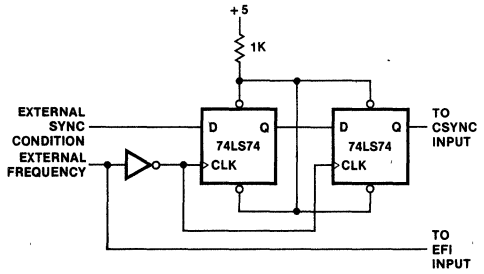
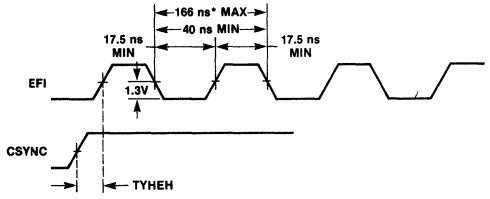


Figure 3B5. Synchronizing CSYNC with EFI



*MAX IS SPEC'ED TO GUARANTEE MAX 8086 CLOCK FREQUENCY

Diagram 3B1. CSYNC Setup and Hold to EFI

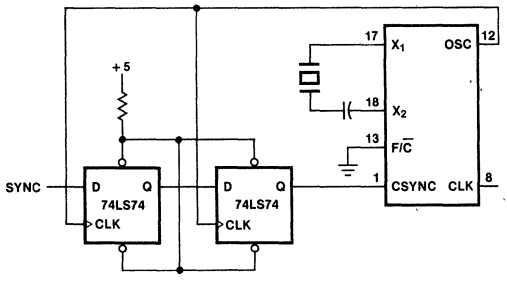


Figure 3B6. EFI from 8284 Oscillator

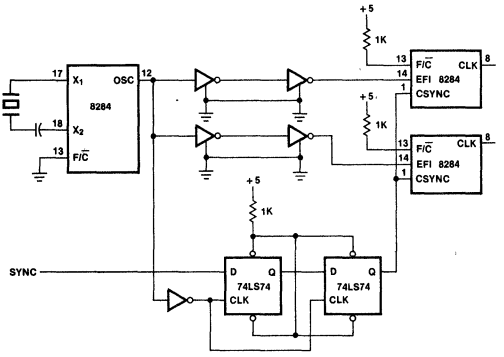


Figure 3B7. Synchronizing Multiple 8284s

Due to the fast transitions and high drive (5 mA) of the 8284 CLK output, it may be necessary to put a 10 to 100 ohm resistor in series with the clock line to eliminate ringing (resistor value depending on the amount of drive required). If multiple sources of CLK are needed with minimum skew, CLK can be buffered by a high drive device (74S241) with outputs tied to 5 volts through 100 ohms to guarantee VOH = 3.9 min (8086 minimum clock input high voltage) (Fig. 3B8). A single 8284 should not be used to generate the CLK for multiple CPU's that do not share a common local (multiplexed) bus since the 8284 synchronizes ready to the CPU and can only accommodate ready for a single CPU. If multiple CPU's share a local bus, they should be driven with the same clock to optimize transfer of bus control. Under these circumstances, only one CPU will be using the bus for a particular bus cycle which allows sharing a common READY signal (Fig. 3B9).

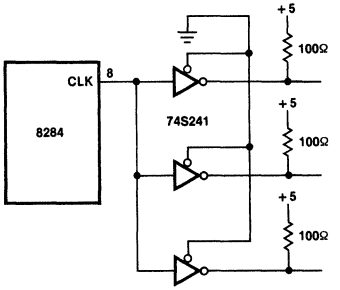


Figure 3B8. Buffering the 8284 CLK Output

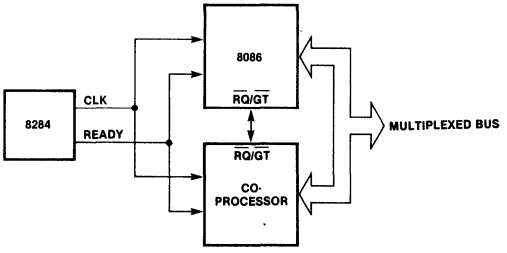


Figure 3B9. 8086 and Co-Processor on the Local Bus Share a Common 8284

3C. Reset

The 8086 requires a high active reset with minimum pulse width of four CPU clocks except after power on which requires a 50 μs reset pulse. Since the CPU internally synchronizes reset with the clock, the reset is internally active for up to one clock period after the external reset. Non-Maskable Interrupts (NMI) or hold requests on $\overline{RQ}/\overline{GT}$ which occur during the internal reset, are not acknowledged. A minimum mode hold request or maximum mode $\overline{RQ}$ pulses active immediately after the internal reset will be honored before the first instruction fetch.

From reset, the 8086 will condition the bus as shown in Table 3C1. The multiplexed bus will three-state upon detection of reset by the CPU. Other signals which three-state will be driven to the inactive state for one clock low interval prior to entering three-state (Fig. 3C1). In the minimum mode, ALE and HLDA are driven inactive and are not three-stated. In the maximum mode, $\overline{RQ}/\overline{GT}$ lines are held inactive and the queue status indicates no activity. The queue status will not indicate a reset of the queue so any user defined external circuits monitoring the queue should also be reset by the system reset. 22K ohm pull-up resistors should be connected to the CPU command and bus control lines to guarantee the inactive state of these lines in systems where leakage currents or bus capacitance may cause the voltage levels to settle below the minimum high voltage of devices in the system. In maximum mode systems, the 8288 contains internal pull-ups on the $\overline{S0}$-$\overline{S2}$ inputs to maintain the inactive state for these lines when the CPU floats the bus. The high state of the status lines during reset causes the 8288 to treat the reset sequence as a passive state. The condition of the 8288 outputs for the passive state are shown in Table 3C2. If the reset occurs during a bus cycle, the return of the status lines to the passive state will terminate the bus cycle and return the command lines to the inactive state. Note that the 8288 does not three-state the command outputs based on the passive state of the status lines. If the designer needs to three-state the CPU off the bus during reset in a single CPU system, the reset signal should also be connected to the 8288's $\overline{AEN}$ input and the output enable of the address latches (Fig. 3C2). This forces the command and address bus interface to three-state while the inactive state of DEN from the 8288 three-states the transceivers on the data bus.

Table 3C1. 8086 Bus During Reset

Signals	Condition
AD_{15-0}	Three-State
A_{19-16}/S_{6-3}	Three-State
BHE/S_7	Three-State
$\overline{S2}/(M/\overline{IO})$	Driven to "1" then three-state
$\overline{S1}/(DT/\overline{R})$	Driven to "1" then three-state
$\overline{S0}/\overline{DEN}$	Driven to "1" then three-state
$LOCK/\overline{WR}$	Driven to "1" then three-state
$\overline{RD}$	Driven to "1" then three-state
$\overline{INTA}$	Driven to "1" then three-state
ALE	0
HLDA	0
$\overline{RQ}/\overline{GT0}$	1
$\overline{RQ}/\overline{GT1}$	1
QS0	0
QS1	0

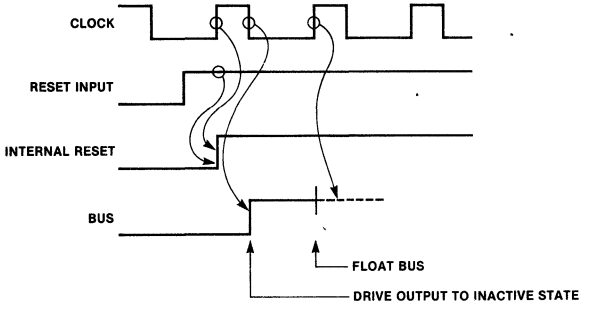


Figure 3C1. 8086 Bus Conditioning on Reset

TABLE 3C2. 8288 OUTPUTS DURING PASSIVE MODE

ALE	0
DEN	0
DT/$\overline{R}$	1
MCE/$\overline{PDEN}$	0/1
COMMANDS	1

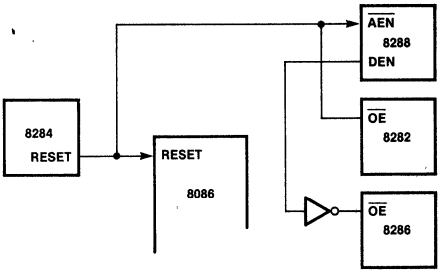


Figure 3C2. Reset Disable for Max Mode 8086 Bus Interface

For multiple processor systems using arbitration of a multimaster bus, the system reset should be connected to the $\overline{INIT}$ input of the 8289 bus arbiter in addition to the 8284 reset input (Fig. 3C3). The low active $\overline{INIT}$ input forces all 8289 outputs to their inactive state. The inactive state of the 8289 $\overline{AEN}$ output will force the 8288 to three-state the command outputs and the address latches to three-state the address bus interface. DEN inactive from the 8288 will three-state the data bus interface. For the multimaster CPU configuration, the reset should be common to all CPU's (8289's and 8284's) and satisfy the maximum of either the CPU reset requirements or 3 TBLBL (3 8289 bus clock times) + 3 TCLCL (3 8086 clock cycle times) to satisfy 8289 reset requirements.

If the 8288 command outputs are three-stated during reset, the command lines should be pulled up to V_{CC} through 2.2K ohm resistors.

The reset signal to the 8086 can be generated by the 8284. The 8284 has a schmitt trigger input ($\overline{RES}$) for generating reset from a low active external reset. The hysteresis specified in the 8284 data sheet implies that at least .25 volts will separate the 0 and 1 switching point of the 8284 reset input. Inputs without hysteresis will switch from low to high and high to low at approximately the same voltage threshold. The inputs are guaranteed to switch at specified low and high voltages (VIL and VIH) but the actual switching point is anywhere in-between. Since VIL min is specified at .8 volts, the hysteresis guarantees that the reset will be active until the input reaches at least 1.05 volts. A reset will not be recognized until the input drops at least .25 volts below the reset inputs VIH of 2.6 volts.

To guarantee reset from power up, the reset input must remain below 1.05 volts for 50 microseconds after V_{CC} has reached the minimum supply voltage of 4.5 volts. The hysteresis allows the reset input to be driven by a simple RC circuit as shown in Figure 3C4. The calculated RC value does not include time for the power supply to reach 4.5 volts or the charge accumulated during this interval. Without the hysteresis, the reset output might oscillate as the input voltage passes through the switching voltage of the input. The calculated RC value provides the minimum required reset period of 50 microseconds for 8284's that switch at the 1.05 volt level and a reset period of approximately 162 microseconds for 8284's that switch at the 2.6 volt level. If tighter tolerance between the minimum and maximum reset times is necessary, the reset circuit shown in Figure 3C5 might be used rather than the simple RC circuit. This circuit provides a constant current source and a linear charge rate on the capacitor rather than the inverse exponential charge rate of the RC circuit. The maximum reset period for this implementation is 124 microseconds.

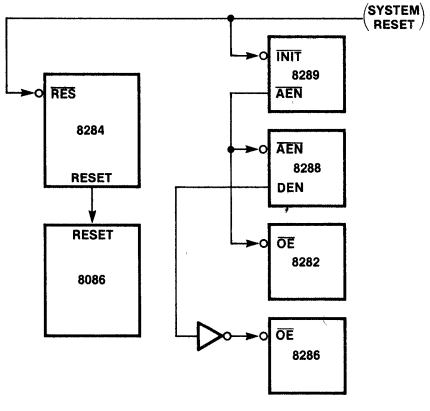


Figure 3C3. Reset Disable of for Max Mode 8086 Bus Interface in Multi CPU System

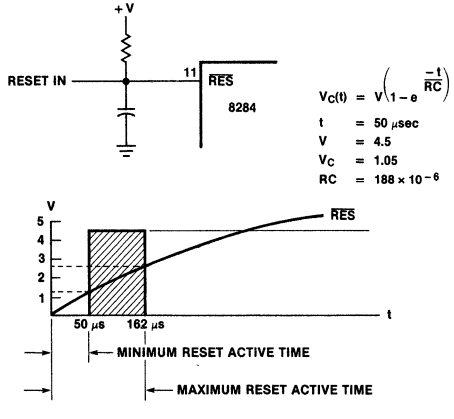


Figure 3C4. 8284 Reset Circuit

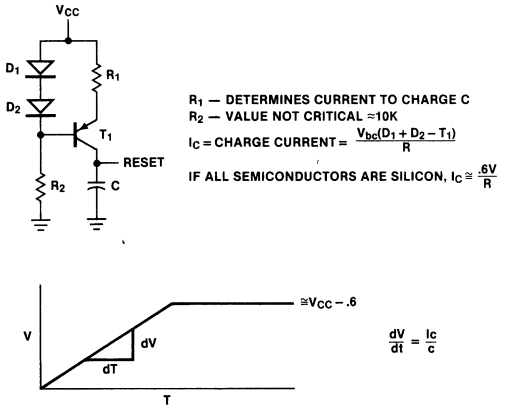


Figure 3C5. Constant Current Power-On Reset Circuit

The 8284 synchronizes the reset input with the CPU clock to generate the RESET signal to the CPU (Fig. 3C6). The output is also available as a general reset to the entire system. The reset has no effect on any clock circuits in the 8284.

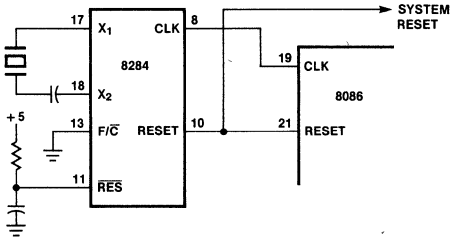


Figure 3C6. 8086 Reset and System Reset

3D. Ready Implementation and Timing

As discussed previously, the ready signal is used in the system to accommodate memory and I/O devices that cannot transfer information at the maximum CPU bus bandwidth. Ready is also used in multiprocessor systems to force the CPU to wait for access to the system bus or Multibus system bus. To insert a wait state in the bus cycle, the READY signal to the CPU must be inactive (low) by the end of T2. To avoid insertion of a wait state, READY must be active (high) within a specified setup time prior to the positive transition during T3. Depending on the size and characteristics of the system, ready implementation may take one of two approaches.

The classical ready implementation is to have the system 'normally not ready.' When the selected device receives the command ($\overline{RD}$/$\overline{WR}$/$\overline{INTA}$) and has had sufficient time to complete the command, it activates READY to the CPU, allowing the CPU to terminate the bus cycle. This implementation is characteristic of large multiprocessor, Multibus systems or systems where propagation delays, bus access delays and device characteristics inherently slow down the system. For maximum system performance, devices that can run with no wait states must return 'READY' within the previously described limit. Failure to respond in time will only result in the insertion of one or more wait cycles.

An alternate technique is to have the system 'normally ready.' All devices are assumed to operate at the maximum CPU bus bandwidth. Devices that do not meet the requirement must disable READY by the end of T2 to guarantee the insertion of wait cycles. This implementation is typically applied to small single CPU systems and reduces the logic required to control the ready signal. Since the failure of a device requiring wait states to disable READY by the end of T2 will result in premature termination of the bus cycle, the system timing must be carefully analyzed when using this approach.

The 8086 has two different timing requirements on READY depending on the system implementation. For a 'normally ready' system to insert a wait state, the READY must be disabled within 8 ns (TRYLCL) after the end of T2 (start of T3) (Diag. 3D1). To guarantee proper

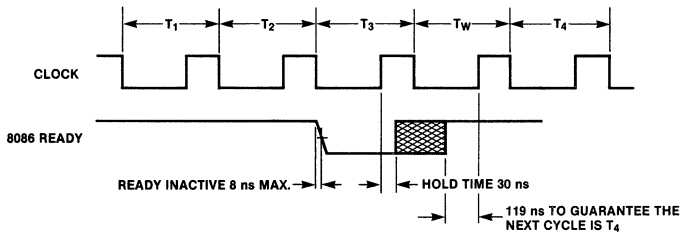


Diagram 3D1. Normally Ready System Inserting a Wait State

operation of the 8086, the READY input must not change from ready to not ready during the clock low time of T3. For a 'normally not ready' system to avoid wait states, READY must be active within 119 ns (TRYHCH) of the positive clock transition during T3 (Diag. 3D2). For both cases, READY must satisfy a hold time of 30 ns (TCHRYX) from the T3 or TW positive clock transition.

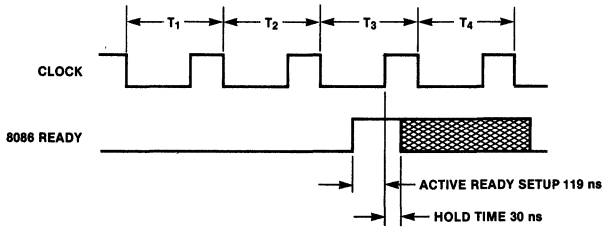


Diagram 3D2. Normally Not Ready System Avoiding a Wait State

To generate a stable READY signal which satisfies the previous setup and hold times, the 8284 provides two separate system ready inputs (RDY1, RDY2) and a single synchronized ready output (READY) for the CPU. The RDY inputs are qualified with separate access enables ($\overline{AEN1}$,$\overline{AEN2}$, low active) to allow selecting one of the two ready signals (Fig. 3D1). The gated signals are logically OR'ed and sampled at the beginning of each CLK cycle to generate READY to the CPU (Diag. 3D3). The sampled READY signal is valid within 8 ns (TRYLCL) after CLK to satisfy the CPU timing requirements on 'not ready' and ready. Since READY cannot change until the next CLK, the hold time requirements are also satisfied. The system ready inputs to the 8284 (RDY1,RDY2) must be valid 35 ns (TRIVCL) before T3 and $\overline{AEN}$ must be valid 60 ns before T3. For a system using only one RDY input, the associated $\overline{AEN}$ is tied to ground while the other $\overline{AEN}$ is connected to 5 volts through ~1K ohms (Fig. 3D2a). If the system generates a low active ready signal, it can be connected to the 8284 $\overline{AEN}$ input if the additional setup time required by the 8284 $\overline{AEN}$ input is satisfied. In this case, the associated RDY input would be tied high (Fig. 3D2b).

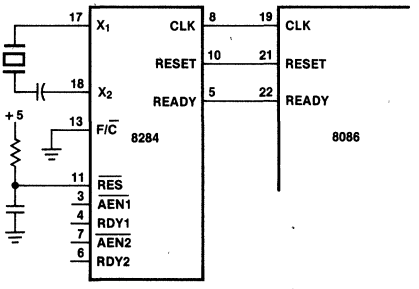
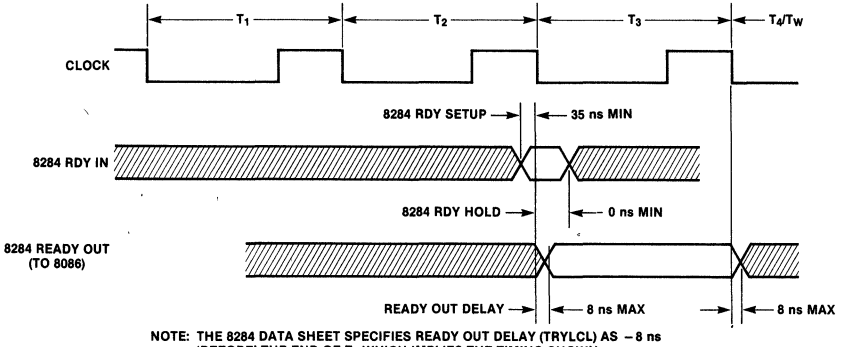


Figure 3D1. Ready Inputs to the 8284 and Output to the 8086



NOTE: THE 8284 DATA SHEET SPECIFIES READY OUT DELAY (TRYLCL) AS −8 ns 'BEFORE' THE END OF T_2 WHICH IMPLIES THE TIMING SHOWN.

Diagram 3D3. 8284 with 8086 Ready Timing

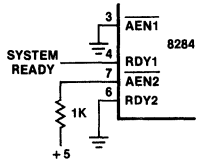


Figure 3D2a. Using RDY1/RDY2 to Generate Ready

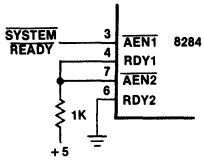


Figure 3D2b. Using $\overline{AEN1}$/$\overline{AEN2}$ to Generate Ready

The majority of memory and peripheral devices which fail to operate at the maximum CPU frequency typically do not require more than one wait state. The circuit given in Figure 3D3 is an example of a simple wait state generator. The system ready line is driven low whenever a device requiring one wait state is selected. The flip flop is cleared by ALE, enabling RDY to the 8284. If no wait states are required, the flip flop does not change. If the system ready is driven low, the flip flop toggles on the low to high clock transition of T2 to force one wait state. The next low to high clock transition toggles the flip flop again to indicate ready and allow completion of the bus cycle. Further changes in the state of the flip flop will not affect the bus cycle. The circuit allows approximately 100 ns for chip select decode and conditioning of the system ready (Diag. 3D4).

If the system is 'normally not ready,' the programmer should not assign executable code to the last six bytes of physical memory. Since the 8086 prefetches instructions, the CPU may attempt to access non-existent memory when executing code at the end of physical memory. If the access to non-existent memory fails to enable READY, the system will be caught in an indefinite wait.

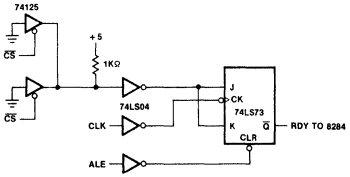


Figure 3D3. Single Wait State Generator

3E. Interrupt Structure

The 8086 interrupt structure is based on a table of interrupt vectors stored in memory locations 0H through 003FFH. Each vector consists of two bytes for the instruction pointer and two bytes for the code segment. These two values combine to form the address of the interrupt service routine. This allows the table to contain up to 256 interrupt vectors which specify the starting address of the service routines anywhere in the one megabyte address space of the 8086. If fewer than 256 different interrupts are defined in the system, the user need only allocate enough memory for the interrupt vector table to provide the vectors for the defined interrupts. During initial system debug, however, it may be desirable to assign all undefined interrupt types to a trap routine to detect erroneous interrupts.

Each vector is associated with an interrupt type number which points to the vector's location in the interrupt vector table. The interrupt type number multiplied by four gives the displacement of the first byte of the associated interrupt vector from the beginning of the table. As an example, interrupt type number 5 points to the sixth entry in the interrupt vector table. The contents of this entry in the table points to the interrupt service routine for type 5 (Fig. 3E1). This structure allows the user to specify the memory address of each service routine by placing the address (instruction pointer and code segment values) in the table location provided for that type interrupt.

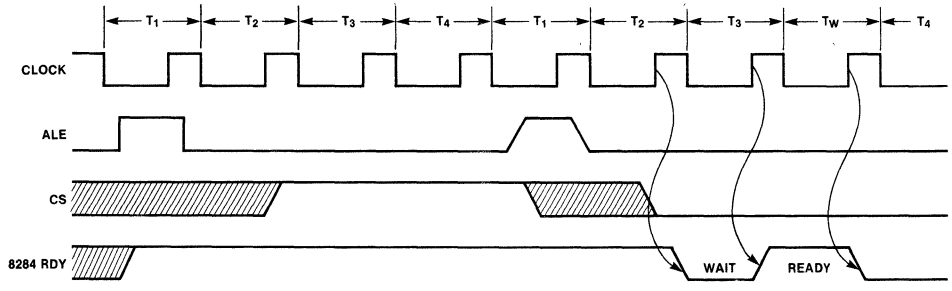


Diagram 3D4.

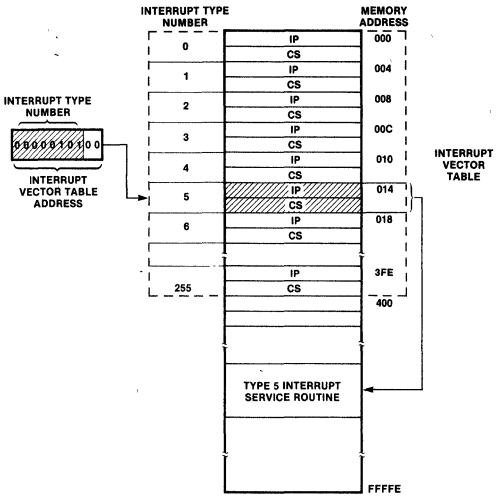


Figure 3E1. Direction to Interrupt Service Routine through the Interrupt Vector Table

All interrupts in the 8086 must be assigned an interrupt type which uniquely identifies each interrupt. There are three classes of interrupt types in the 8086; predefined interrupt types which are issued by specific functions within the 8086 and user defined hardware and software interrupts. Note that any interrupt type including the predefined interrupts can be issued by the user's hardware and/or software.

PREDEFINED INTERRUPTS

The predefined interrupt types in the 8086 are listed below with a brief description of how each is invoked. When invoked, the CPU will transfer control to the memory location specified by the vector associated with the specific type. The user must provide the interrupt service routine and initialize the interrupt vector table with the appropriate service routine address. The user may additionally invoke these interrupts through hardware or software. If the preassigned function is not used in the system, the user may assign some other function to the associated type. However, for compatibility with future Intel hardware and software products for the 8086 family, interrupt types 0-31 should not be assigned as user defined interrupts.

TYPE 0 — DIVIDE ERROR

This interrupt type is invoked whenever a division operation is attempted during which the quotient exceeds the maximum value (ex. division by zero). The interrupt is non-maskable and is entered as part of the execution of the divide instruction. If interrupts are not reenabled by the divide error interrupt service routine, the service routine execution time should be included in the worst case divide instruction execution time (primarily when considering the longest instruction execution time and its effect on latency to servicing hardware interrupts).

TYPE 1 — SINGLE STEP

This interrupt type occurs one instruction after the TF (Trap Flag) is set in the flag register. It is used to allow software single stepping through a sequence of code. Single stepping is initiated by copying the flags onto the stack, setting the TF bit on the stack and popping the flags. The interrupt routine should be the single step routine. The interrupt sequence saves the flags and program counter, then resets the TF flag to allow the single step routine to execute normally. To return to the routine under test, an interrupt return restores the IP, CS and flags with TF set. This allows the execution of the next instruction in the program under test before trapping back to the single step routine. Single Step is not masked by the IF (Interrupt Flag) bit in the flag register.

TYPE 2 — NMI (Non-Maskable Interrupt)

This is the highest priority hardware interrupt and is non-maskable. The input is edge triggered but is synchronized with the CPU clock and must be active for two clock cycles to guarantee recognition. The interrupt signal may be removed prior to entry to the service routine. Since the input must make a low to high transition to generate an interrupt, spurious transitions on the input should be suppressed. If the input is normally high, the NMI low time to guarantee triggering is two CPU clock times. This input is typically reserved for catastrophic failures like power failure or timeout of a system watchdog timer.

TYPE 3 — ONE BYTE INTERRUPT

This is invoked by a special form of the software interrupt instruction which requires a single byte of code space. Its primary use is as a breakpoint interrupt for software debug. With full representation within a single byte, the instruction can map into the smallest instruction for absolute resolution in setting breakpoints. The interrupt is not maskable.

TYPE 4 — INTERRUPT ON OVERFLOW

This interrupt occurs if the overflow flag (OF) is set in the flag register and the INTO instruction is executed. The instruction allows trapping to an overflow error service routine. The interrupt is non-maskable.

Interrupt types 0 and 2 can occur without specific action by the programmer (except for performing a divide for Type 0) while types 1, 3, and 4 require a conscious act by the programmer to generate these interrupt types. All but type 2 are invoked through software activity and are directly associated with a specific instruction.

USER DEFINED SOFTWARE INTERRUPTS

The user can generate an interrupt through the software with a two byte interrupt instruction INT nn. The first byte is the INT opcode while the second byte (nn) contains the type number of the interrupt to be performed. The INT instruction is not maskable by the interrupt enable flag. This instruction can be used to transfer control to routines that are dynamically relocatable and whose location in memory is not known by the calling

program. This technique also saves the flags of the calling program on the stack prior to transferring control. The called procedure must return control with an interrupt return (IRET) instruction to remove the flags from the stack and fully restore the state of the calling program.

All interrupts invoked through software (all interrupts discussed thus far with the exception of NMI) are not maskable with the IF flag and initiate the transfer of control at the end of the instruction in which they occur. They do not initiate interrupt acknowledge bus cycles and will disable subsequent maskable interrupts by resetting the IF and TF flags. The interrupt vector for these interrupt types is either implied or specified in the instruction. Since the NMI is an asynchronous event to the CPU, the point of recognition and initiation of the transfer of control is similar to the maskable hardware interrupts.

USER DEFINED HARDWARE INTERRUPTS

The maskable interrupts initiated by the system hardware are activated through the INTR pin of the 8086 and are masked by the IF bit of the status register (interrupt flag). During the last clock cycle of each instruction, the state of the INTR pin is sampled. The 8086 deviates from this rule when the instruction is a MOV or POP to a segment register. For this case, the interrupts are not sampled until completion of the following instruction. This allows a 32-bit pointer to be loaded to the stack pointer registers SS and SP without the danger of an interrupt occurring between the two loads. Another exception is the WAIT instruction which waits for a low active input on the $\overline{TEST}$ pin. This instruction also continuously samples the interrupt request during its execution and allows servicing interrupts during the wait. When an interrupt is detected, the WAIT instruction is again fetched prior to servicing the interrupt to guarantee the interrupt routine will return to the WAIT instruction.

UNINTERRUPTABLE INSTRUCTION SEQUENCE

```
MOV SS, NEW$STACK$SEGMENT
MOV SP, NEW$STACK$POINTER
```

Also, since prefixes are considered part of the instruction they precede, the 8086 will not sample the interrupt line until completion of the instruction the prefix(es) precede(s). An exception to this (other than HALT or WAIT) is the string primatives preceded by the repeat (REP) prefix. The repeated string operations will sample the interrupt line at the completion of each repetition. This includes repeat string operations which include the lock prefix. If multiple prefixes precede a repeated string operation, and the instruction is interrupted, only the prefix immediately preceding the string primative is restored. To allow correct resumption of the operation, the following programming technique may be used:

```
LOCKED$BLOCK$MOVE: LOCK REP MOVS DEST, CS:SOURCE
                   AND CX,     CX
                   JNZ LOCKED$BLOCK$MOVE
```

The code bytes generated by the 8086 assembler for the MOVS instruction are (in descending order): LOCK prefix, REP prefix, Segment Override prefix and MOVS. Upon return from the interrupt, the segment override prefix is restored to guarantee one additional transfer is performed between the correct memory locations. The instructions following the move operation test the repetition count value to determine if the move was completed and return if not.

If the INTR pin is high when sampled and the IF bit is set to enable interrupts, the 8086 executes an interrupt acknowledge sequence. To guarantee the interrupt will be acknowledged, the INTR input must be held active until the interrupt acknowledge is issued by the CPU. If the BIU is running a bus cycle when the interrupt condition is detected (as would occur if the BIU is fetching an instruction when the current instruction completes), the

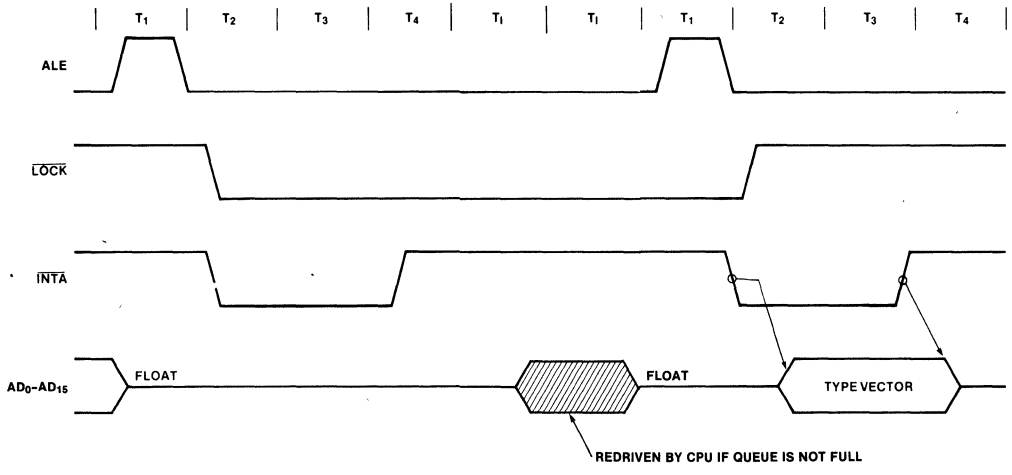


Figure 3E2. Interrupt Acknowledge Sequence

interrupt must be valid at the 8086 2 clock cycles prior to T4 of the bus cycle if the next cycle is to be an interrupt acknowledge cycle. If the 2 clock setup is not satisfied, another pending bus cycle will be executed before the interrupt acknowledge is issued. If a hold request is also pending (this might occur if an interrupt and hold request are made during execution of a locked instruction), the interrupt is serviced after the hold request is serviced.

The interrupt acknowledge sequence is only generated in response to an interrupt on the 8086 INTR input. The associated bus activity is shown in Figure 3E2. The cycle consists of two $\overline{INTA}$ bus cycles separated by two idle clock cycles. During the bus cycles the $\overline{INTA}$ command is issued rather than read. No address is provided by the 8086 during either bus cycle ($\overline{BHE}$ and status are valid), however, ALE is still generated and will load the address latches with indeterminate information. This condition requires that devices in the system do not drive their outputs without being qualified by the Read Command. As will be shown later, the ALE is useful in maximum mode systems with multiple 8259A priority interrupt controllers. During the $\overline{INTA}$ bus cycles, DT/$\overline{R}$ and DEN are conditioned to allow the 8086 to receive a one byte interrupt type number from the interrupt system. The first $\overline{INTA}$ bus cycle signals an interrupt acknowledge cycle is in progress and allows the system to prepare to present the interrupt type number on the next $\overline{INTA}$ bus cycle. The CPU does not capture information on the bus during the first cycle. The type number must be transferred to the 8086 on the lower half of the 16-bit data bus during the second cycle. This implies that devices which present interrupt type numbers to the 8086 must be located on the lower half of the 16-bit data bus. The timing of the $\overline{INTA}$ bus cycles (with exception of address timing) is similar to read cycle timing. The 8086 interrupt acknowledge sequence deviates from the form used on 8080 and 8085 in that no instruction is issued as part of the sequence. The 8080 and 8085 required either a restart or call instruction be issued to affect the transfer of control.

In the minimum mode system, the M/$\overline{IO}$ signal will be low indicating I/O during the $\overline{INTA}$ bus cycles. The 8086 internal $\overline{LOCK}$ signal will be active from T2 of the first bus cycle until T2 of the second to prevent the BIU from honoring a hold request between the two $\overline{INTA}$ cycles.

In the maximum mode, the status lines $\overline{S0}$-$\overline{S2}$ will request the 8288 to activate the $\overline{INTA}$ output for each cycle. The $\overline{LOCK}$ output of the 8086 will be active from T2 of the first cycle until T2 of the second to prevent the 8086 from honoring a hold request on either $\overline{RQ}$/$\overline{GT}$ input and to prevent bus arbitration logic from relinquishing the bus between $\overline{INTA}$'s in multi-master systems. The consequences of READY are identical to those for READ and WRITE cycles.

Once the 8086 has the interrupt type number (from the bus for hardware interrupts, from the instruction stream for software interrupts or from the predefined condition), the type number is multiplied by four to form the displacement to the corresponding interrupt vector in the interrupt vector table. The four bytes of the interrupt vector are: least significant byte of the instruction pointer, most significant byte of the instruction pointer, least significant byte of the code segment register, most significant byte of the code segment register. During the transfer of control, the CPU pushes the flags and current code segment register and instruction pointer onto the stack. The new code segment and instruction pointer values are loaded and the single step and interrupt flags are reset. Resetting the interrupt flag disables response to further hardware interrupts in the service routine unless the flags are specifically re-enabled by the service routine. The CS and IP values are read from the interrupt vector table with data read cycles. No segment registers are used when referencing the vector table during the interrupt context switch. The vector displacement is added to zero to form the 20-bit address and S4, S3 = 10 indicating no segment register selection.

The actual bus activity associated with the hardware interrupt acknowledge sequence is as follows: Two interrupt acknowledge bus cycles, read new IP from the interrupt vector table, read new CS from the interrupt vector table, Push flags, Push old CS, Opcode fetch of the first instruction of the interrupt service routine, and Push old IP. After saving the old IP, the BIU will resume normal operation of prefetching instructions into the queue and servicing EU requests for operands. S5 (interrupt enable flag status) will go inactive in the second clock cycle following reading the new CS.

The number of clock cycles from the end of the instruction during which the interrupt occurred to the start of interrupt routine execution is 61 clock cycles. For software generated interrupts, the sequence of bus cycles is the same except no interrupt acknowledge bus cycles are executed. This reduces the delay to service routine execution to 51 clocks for INT nn and single step, 52 clocks for INT3 and 53 clocks for INTO. The same interrupt setup requirements with respect to the BIU that were stated for the hardware interrupts also apply to the software interrupts. If wait states are inserted by either the memories or the device supplying the interrupt type number, the given clock times will increase accordingly.

When considering the precedence of interrupts for multiple simultaneous interrupts, the following guidelines apply: 1. INTR is the only maskable interrupt and if detected simultaneously with other interrupts, resetting of IF by the other interrupts will mask INTR. This causes INTR to be the lowest priority interrupt serviced after all other interrupts unless the other interrupt service routines reenable interrupts. 2. Of the nonmaskable interrupts (NMI, Single Step and software generated), in general, Single Step has highest priority (will be serviced first) followed by NMI, followed by the software interrupts. This implies that a simultaneous NMI and Single Step trap will cause the NMI service routine to follow single step; a simultaneous software trap and Single Step trap will cause the software interrupt service routine to follow single step and a simultaneous NMI and software trap will cause the NMI service routine to be executed followed by the software interrupt service routine. An exception to this priority structure occurs if all three interrupts are pending. For this case, transfer of control to the software interrupt ser-

vice routine followed by the NMI trap will cause both the NMI and software interrupt service routines to be executed without single stepping. Single stepping resumes upon execution of the instruction following the instruction causing the software interrupt (the next instruction in the routine being single stepped).

If the user does not wish to single step before INTR service routines, the single step routine need only disable interrupts during execution of the program being single stepped and reenable interrupts on entry to the single step routine. Disabling the interrupts during the program under test prevents entry into the interrupt service routine while single step (TF = 1) is active. To prevent single stepping before NMI service routines, the single step routine must check the return address on the stack for the NMI service routine address and return control to that routine without single step enabled. As examples, consider Figures 3E3a and 3E3b. In 3E3a Single Step and NMI occur simultaneously while in 3E3b, NMI, INTR and a divide error all occur during a divide instruction being single stepped.

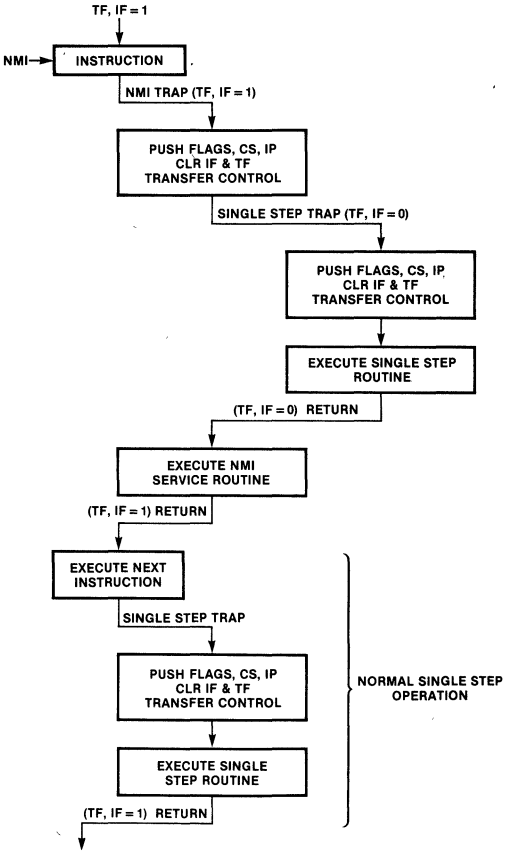


Figure 3E3a. NMI During Single Stepping and Normal Single Step Operation

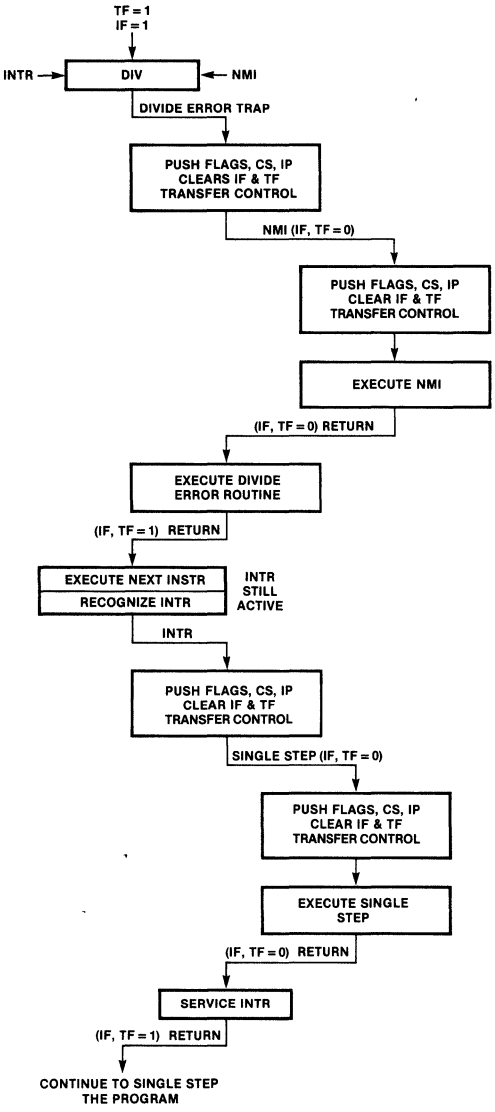


Figure 3E3b. NMI, INTR, Single Step and Divide Error Simultaneous Interrupts

SYSTEM CONFIGURATIONS

To accommodate the $\overline{\text{INTA}}$ protocol of the maskable hardware interrupts, the 8259A is provided as part of the 8086 family. This component is programmable to operate in both 8080/8085 systems and 8086 systems. The devices are cascadable in master/slave arrangements to allow up to 64 interrupts in the system. Figures 3E4 and 3E5 are examples of 8259A's in minimum and maximum mode 8086 systems. The minimum mode configuration (a) shows an 8259A connected to the CPU's

multiplexed bus. Configuration (b) illustrates an 8259A connected to a demultiplexed bus system. These interconnects are also applicable to maximum mode systems. The configuration given for a maximum mode system shows a master 8259A on the CPU's multiplexed bus with additional slave 8259A's out on the buffered system bus. This configuration demonstrates several unique features of the maximum mode system interface. If the master 8259A receives interrupts from a mix of slave 8259A's and regular interrupting devices, the slaves must provide the type number for devices connected to them while the master provides the type number for devices directly attached to its interrupt inputs. The master 8259A is programmable to determine if an interrupt is from a direct input or a slave 8259A and will use this information to enable or disable the data bus transceivers (via the 'nand' function of DEN and $\overline{EN}$). If the master must provide the type number, it will disable the data bus transceivers. If the slave provides the type number, the master will enable the data bus transceivers. The $\overline{EN}$ output is normally high to allow the 8086/8288 to control the bus transceivers. To select the proper slave when servicing a slave interrupt, the master must provide a cascade address to the slave. If the 8288 is not strapped in the I/O bus mode (the 8288 IOB input connected to ground), the MCE/$\overline{PDEN}$ output becomes a MCE or Master Cascade Enable output. This signal is only active during $\overline{INTA}$ cycles as shown in Figure 3E6 and enables the master 8259A's cascade address onto the 8086's local bus during ALE. This allows the address latches to capture the cascade address with ALE and allows use of the system address bus for selecting the proper slave 8259A. The MCE is gated with $\overline{LOCK}$ to minimize local bus contention between the 8086 three-stating its bus outputs and the cascade address being enabled onto the bus. The first $\overline{INTA}$ bus cycle allows the master to resolve internal priorities and output a cascade address to be transmitted to the slaves on the subsequent $\overline{INTA}$ bus cycle. For additional information on the 8259A, reference application note AP-59.

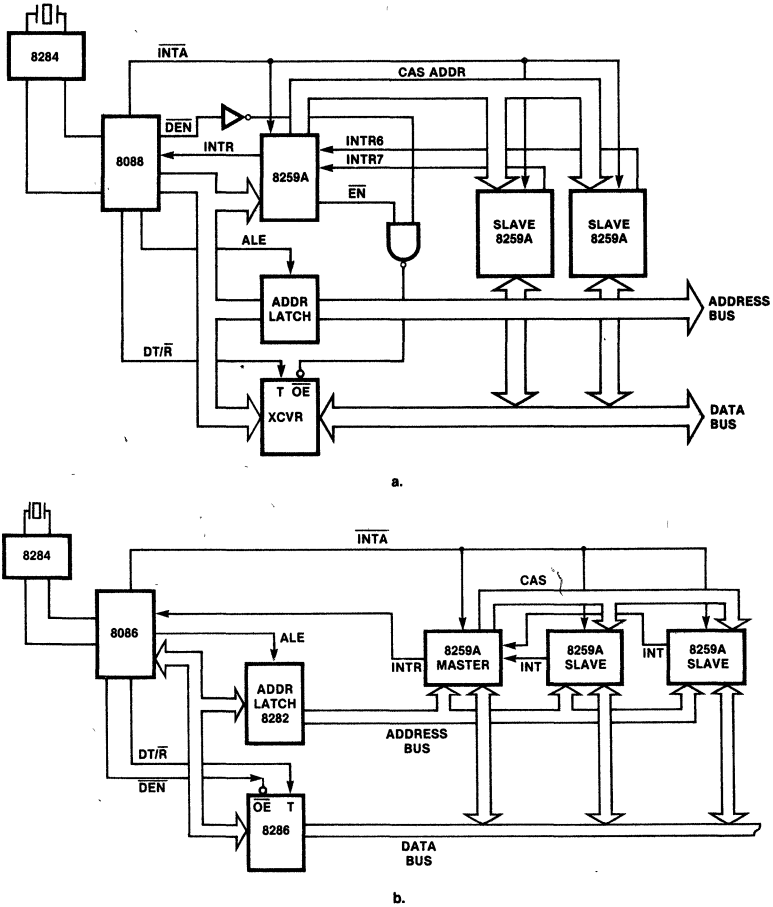


Figure 3E4. Min Mode 8086 with Master 8259A on the Local Bus and Slave 8259As on the System Bus

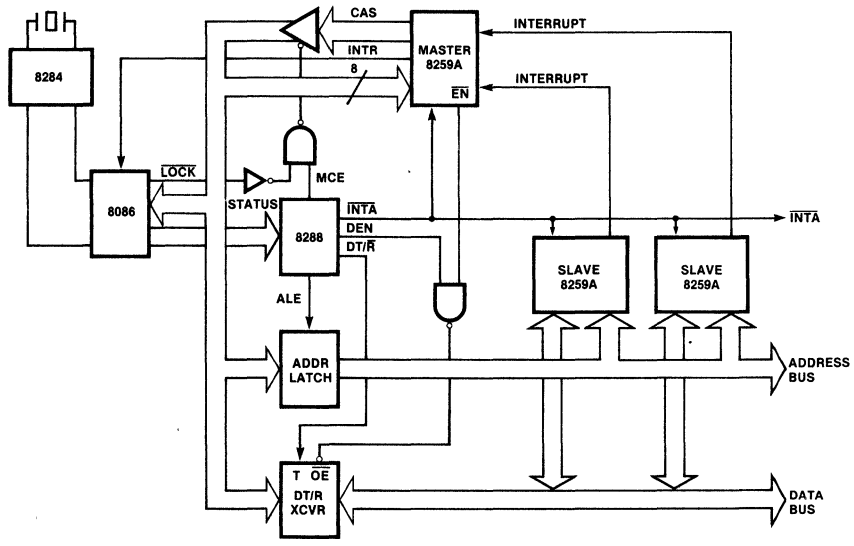


Figure 3E5. Max Mode 8086 with Master 8259A on the Local Bus and Slave 8259As on the System Bus

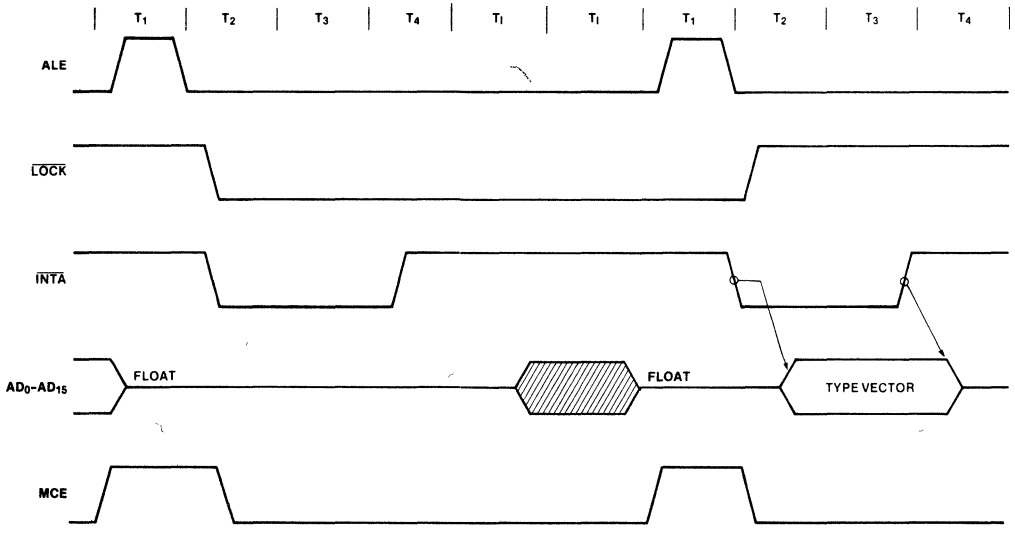


Figure 3E6. MCE Timing to Gate 8259A CAS Address onto the 8086 Local Bus

3F. Interpreting the 8086 Bus Timing Diagrams

At first glance, the 8086 bus timing diagrams (Diag. 3F1 min mode and Diag. 3F2 max mode) appear rather complex. However, with a few words of explanation on how to interpret them, they become a powerful tool in determining system requirements. The timing diagrams for both the minimum and maximum modes may be divided into six sections: (1) address and ALE timing; (2) read cycle timing; (3) write cycle timing; (4) interrupt acknowledge timing; (5) ready timing; and (6) HOLD/HLDA or $\overline{RQ}/\overline{GT}$ timing. Since the A.C. characteristics of the signals are specified relative to the CPU clock, the relationship between the majority of signals can be deduced by simply determining the clock cycles between the clock edges the signals are relative to and adding or subtracting the appropriate minimum or maximum parameter values. One aspect of system timing not compensated for in this approach is the worst case relationship between minimum and maximum parameter values (also known as tracking relationships). As an example, consider a signal which has specified minimum and maximum turn on and turn off delays. Depending on device characteristics, it may not be possible for the component to simultaneously demonstrate a maximum turn-on and minimum turn-off delay even though worst case analysis might imply the possibility. This argument is characteristic of MOS devices and is therefore applicable to the 8086 A.C. characteristics. The message is: worst case analysis mixing minimum and maximum delay parameters will typically exceed the worst case obtainable and therefore should not be subjected to further subjective degradation to obtain worst-worst case values. This section will provide guidelines for specific areas of 8086 timing sensitive to tracking relationships.

A. MINIMUM MODE BUS TIMING

1. ADDRESS and ALE

The address/ALE timing relationship is important to determine the ability to capture a valid address from the multiplexed bus. Since the 8282 and 8283 latches capture the address on the trailing edge of ALE, the critical timing involves the state of the address lines when ALE terminates. If the address valid delay is assumed to be maximum TCLAV and ALE terminates at its earliest point, TCHLLmin (assuming zero minimum delay), the address would be valid only TCLCHmin-TCLAVmax = 8 ns prior to ALE termination. This result is unrealistic in the assumption of maximum TCLAV and minimum TCHLL. To provide an accurate measure of the true worst case, a separate parameter specifies the minimum time for address valid prior to the end of ALE (TAVAL). TAVAL = TCLCH-60 ns overrides the clock related timings and guarantees 58 ns of address setup to ALE termination for a 5 MHz 8086. The address is guaranteed to remain valid beyond the end of ALE by the TLLAX parameter. This specification overrides the relationship between TCHLL and TCLAX which might seem to imply the address may not be valid by the end of the latest possible ALE. TLLAX holds for the entire address bus. The TCLAXmin spec on the address indicates the earliest the bus will go invalid if not restrained by a slow ALE. TLLAX and TCLAX apply to the entire multiplexed bus for both read and write cycles. AD15-0 is three-stated for read cycles and immediately switched to write data during write cycles. AD19-16 immediately switch from address to status for both read and write cycles. The minimum ALE pulse width is guaranteed by TLHLLmin which takes precedence over the value obtained by relating TCLLHmax and TCHLLmin.

To determine the worst case delay to valid address on a demultiplexed address bus, two paths must be considered: (1) delay of valid address and (2) delay to ALE. Since the 8282 and 8283 are flow through latches, a valid address is not transmitted to the address bus until ALE is active. A comparison of address valid delay TCLAVmax with ALE active delay TCLLHmax indicates TCLAVmax is the worst case. Subtracting the latch propagation delay gives the worst case address bus valid delay from the start of the bus cycle.

2. Read Cycle Timing

Read timing consists of conditioning the bus, activating the read command and establishing the data transceiver enable and direction controls. $DT/\overline{R}$ is established early in the bus cycle and requires no further consideration. During read, the $\overline{DEN}$ signal must allow the transceivers to propagate data to the CPU with the appropriate data setup time and continue to do so until the required data hold time. The $\overline{DEN}$ turn on delay allows TCLCL + TCHCLmin – TCVCTVmax – TDVCL = 127 ns transceiver enable time prior to valid data required by the CPU. Since the CPU data hold time TCLDXmin and minimum $\overline{DEN}$ turnoff delay TCVCTXmin are both 10 ns relative to the same clock edge, the hold time is guaranteed. Additionally, $\overline{DEN}$ must disable the transceivers prior to the CPU redriving the bus with the address for the next bus cycle. The maximum $\overline{DEN}$ turn off delay (TCVCTXmax) compared with the minimum delay for addresses out of the 8086 (TCLCL + TCLAVmin) indicates the transceivers are disabled at least 105 ns before the CPU drives the address onto the multiplexed bus.

If memory or I/O devices are connected directly to the multiplexed address and data bus, the TAZRL parameter guarantees the CPU will float the bus before activating read and allowing the selected device to drive the bus. At the end of the bus cycle, the TRHAV parameter specifies the bus float delay the device being deselected must satisfy to avoid contention with the CPU driving the address for the next bus cycle. The next bus cycle may start as soon as the cycle following T4 or any number of clock cycles later.

The minimum delay from read active to valid data at the CPU is 2TCLCL – TCLRLmax – TDVCL = 205 ns. The minimum pulse width is 2TCLCL – 75 ns = 325 ns. This specification (TRLRH) overrides the result which could be derived from clock relative delays (2TCLCL – TCLRLmax + TCLRHmin).

3. Write Cycle Timing

The write cycle involves providing write data to the system, generating the write command and controlling data bus transceivers. The transceiver direction control signal $DT/\overline{R}$ is conditioned to transmit at the end of each read cycle and does not change during a write cycle.

This allows the transceiver enable signal $\overline{DEN}$ to be active early in the cycle (while addresses are valid) without corrupting the address on the multiplexed bus. The write data and write command are both enabled from the leading edge of T2. Comparing minimum $\overline{WR}$ active delay TCVCTVmin with the maximum write data delay TCLDV indicates that write data may be not valid until 100 ns after write is active. The devices in the system should capture data on the trailing edge of the write command rather than the leading edge to guarantee valid data. The data from the 8086 is valid a minimum of 2TCLCL – TCLDVmax + TCVCTXmin = 300 ns before the trailing edge of write. The minimum write pulse width is TWLWH = 2TCLCL – 60 ns = 340 ns. The CPU maintains valid write data TWHDX ns after write. The TWHDZ specification overrides the result derived by relating TCLCHmin and TCHDZmin which implies write data may only be valid 18 ns after $\overline{WR}$. The 8086 floats the bus after write only if being forced off the bus by a HOLD or $\overline{RQ}$ input. Otherwise, the CPU simply switches the output drivers from data to address at the beginning of the next bus cycle. As with the read cycle, the next bus cycle may start in the clock cycle following T4 or any clock cycle later.

$\overline{DEN}$ is disabled a minimum of TCLCHmin + TCVCTXmin – TCVCTXmax = 18 ns after write to guarantee data hold time to the selected device. Since we are again evaluating a minimum TCVCTX with a maximum TCVCTX, the real minimum delay from the end of write to transceiver disable is approximately 60 ns.

4. Interrupt Acknowledge Timing

The interrupt acknowledge sequence consists of two interrupt acknowledge bus cycles as previously described. The detailed timing of each cycle is identical to the read cycle timing with two exceptions: command timing and address/data bus timing.

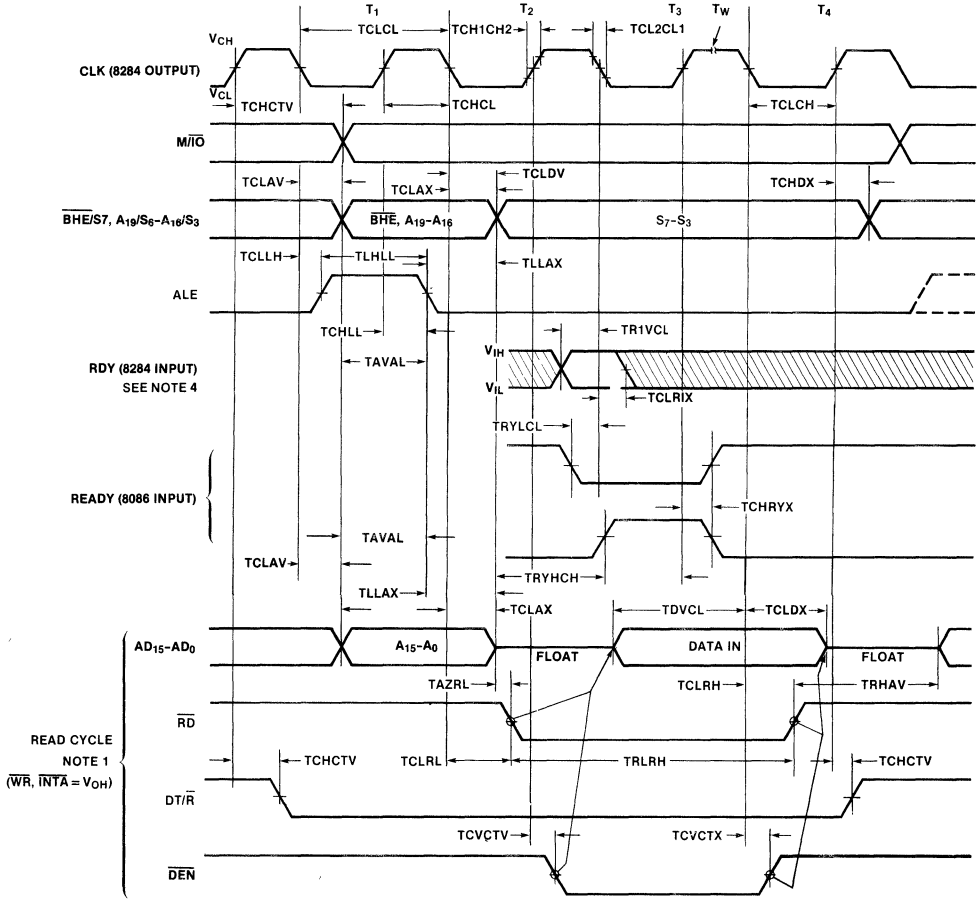
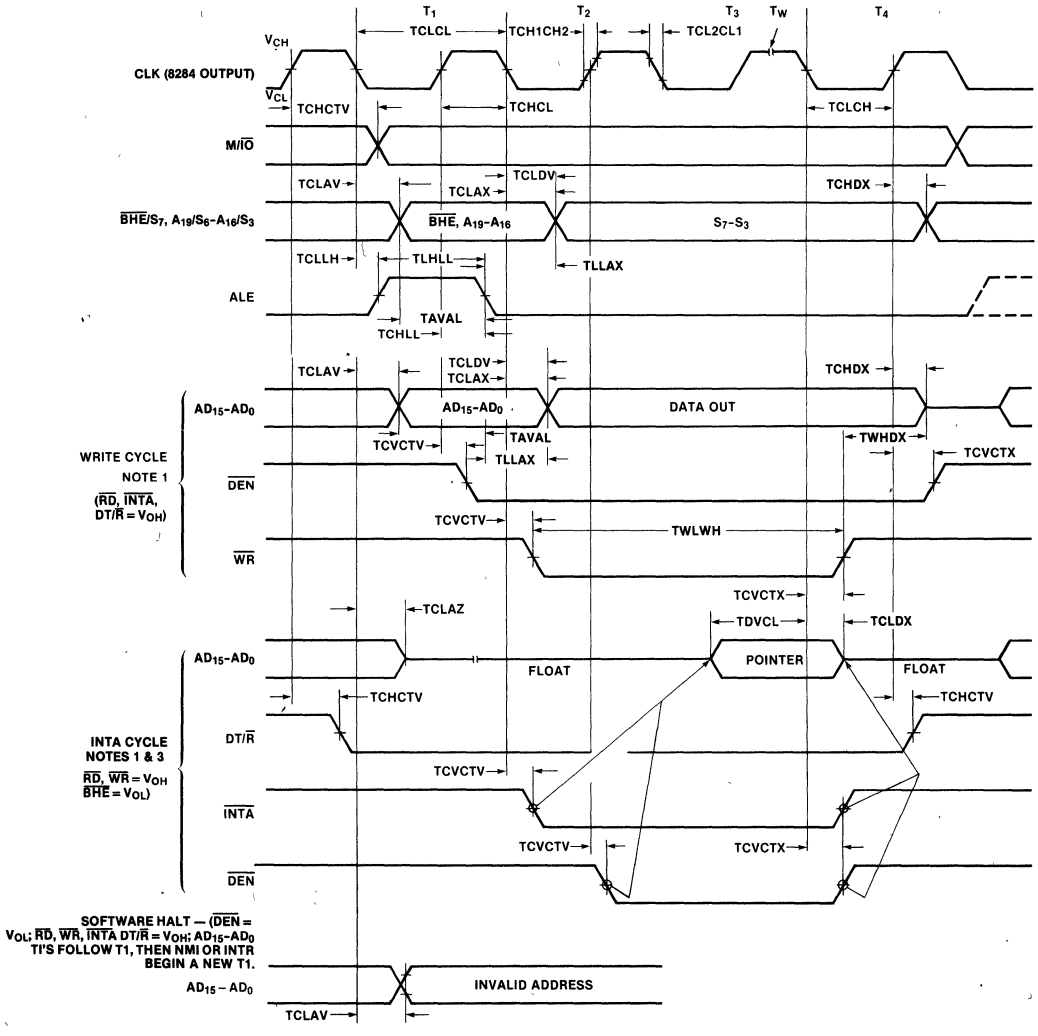


Figure 3F1. 8086 Bus Timing — Minimum Mode System



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OI} 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. BOTH INTA CYCLES RUN BACK-TO-BACK. THE 8088 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE. CONTROL SIGNALS SHOWN FOR SECOND INTA CYCLE.
4. SIGNALS AT 8284 ARE SHOWN FOR REFERENCE ONLY.
5. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

Figure 3F1. 8086 Bus Timing — Minimum Mode System (Con't)

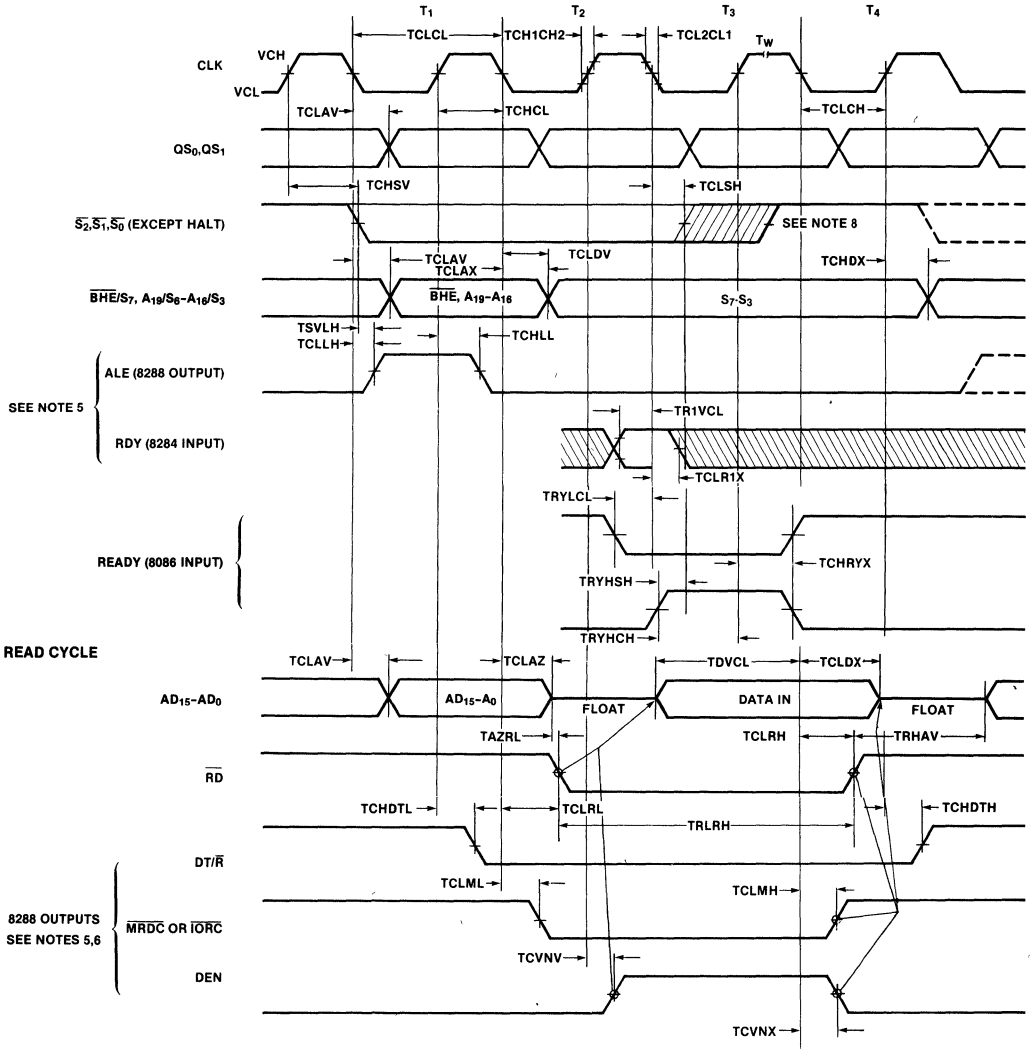
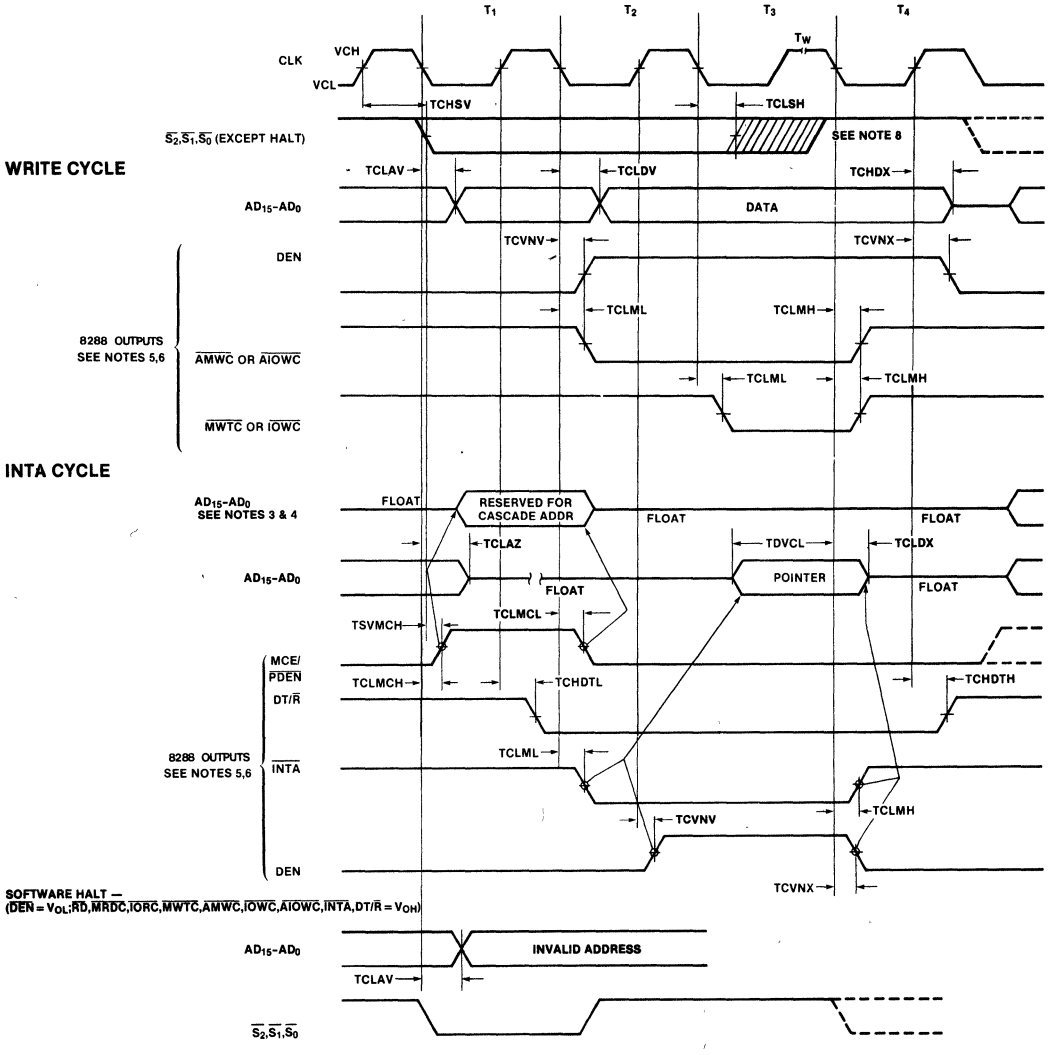


Figure 3F2a. 8086 Bus Timing — Maximum Mode System (Using 8288)



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. BOTH INTA CYCLES RUN BACK-TO-BACK. THE 8086 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE. 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.

Figure 3F2b. 8086 Bus Timing — Maximum Mode System (Using 8288) (Con't)

The multiplexed address/data bus floats from the beginning (T1) of the $\overline{INTA}$ cycle (within TCLAZ ns). The upper four multiplexed address/status lines do not three-state. The address value on A19-A16 is indeterminate but the status information will be valid (S3 = 0, S4 = 0, S5 = IF, S6 = 0, S7 = $\overline{BHE}$ = 0). The multiplexed address/data lines will remain in three-state until the cycle after T4 of the $\overline{INTA}$ cycle. This sequence occurs for each of the $\overline{INTA}$ bus cycles. The interrupt type number read by the 8086 on the second $\overline{INTA}$ bus cycle must satisfy the same setup and hold times required for data during a read cycle.

The $\overline{DEN}$ and DT/$\overline{R}$ signals are enabled for each $\overline{INTA}$ cycle and do not remain active between the two cycles. Their timing for each cycle is identical to the read cycle.

The $\overline{INTA}$ command has the same timing as the write command. It is active within 110 ns of the start of T2 providing 260 ns of access time from command to data valid at the 8086. The command is active a minimum of TCVCTXmin = 10 ns into T4 to satisfy the data hold time of the 8086. This provides minimum $\overline{INTA}$ pulse width of 300 ns, however taking signal delay tracking into consideration gives a minimum pulse width of 340 ns. Since the maximum inactive delay of $\overline{INTA}$ is TCVCTXmax = 110 ns and the CPU will not drive the bus until 15 ns (TCLAVmin) into the next clock cycle, 105 ns are available for interrupt devices on the local bus to float their outputs. If the data bus is buffered, $\overline{DEN}$ provides the same amount of time for local bus transceivers to three-state their outputs.

5. Ready Timing

The detailed timing requirements of the 8086 ready signal and the system ready signal into the 8284 are described in Section 3D. The system ready signal is typically generated from either the address decode of the selected device or the address decode and the command ($\overline{RD}$, $\overline{WR}$, $\overline{INTA}$). For a system which is normally not ready, the time to generate ready from a valid address and not insert a wait state, is 2TCLCL – TCLAVmax – TR1VCLmax = 255 ns. This time is available for buffer delays and address decoding to determine if the selected device does not require a wait state and drive the RDY line high. If wait cycles are required, the user hardware must provide the appropriate ready delay. Since the address will not change until the next ALE, the RDY will remain valid throughout the cycle. If the system is normally ready, selected devices requiring wait states also have 255 ns to disable the RDY line. The user circuitry must delay re-enabling RDY by the appropriate number of wait states.

If the $\overline{RD}$ command is used to enable the RDY signal, TCLCL – TCLRLmax – TRIVCLmax = 15 ns are available for external logic. If the $\overline{WR}$ command is used, TCLCL – TCVCTVmax – TRIVCLmax = 55 ns are available. Comparison of RDY control by address or command indicates that address decoding provides the best timing. If the system is normally not ready, address decode alone could be used to provide RDY for devices not requiring wait states while devices requiring wait states may use a combination of address decode and command to activate a wait state generator. If the system is normally ready, devices not requiring wait states do nothing to RDY while devices needing wait states should disable RDY via the address decode and use a combination of address decode and command to activate a delay to re-enable RDY.

If the system requires no wait states for memory and a fixed number of wait states for $\overline{RD}$ and $\overline{WR}$ to all I/O devices, the M/$\overline{IO}$ signal can be used as an early indication of the need for wait cycles. This allows a common circuit to control ready timing for the entire system without feedback of address decodes.

6. Other Considerations

Detailed HOLD/HLDA timing is covered in the next section and is not examined here. One last signal consideration needs to be mentioned for the minimum mode system. The $\overline{TEST}$ input is sampled by the 8086 only during execution of the WAIT instruction. The $\overline{TEST}$ signal should be active for a minimum of 6 clock cycles during the WAIT instruction to guarantee detection.

B. MAXIMUM MODE BUS TIMING

The maximum mode 8086 bus operations are logically equivalent to the minimum mode operation. Detailed timing analysis now involves signals generated by the CPU and the 8288 bus controller. The 8288 also provides additional control and command signals which expand the flexibility of the system.

1. ADDRESS and ALE

In the maximum mode, the address information continues to come from the CPU while the ALE strobe is generated by the 8288. To determine the worst case relationships between ALE and the address, we first must determine 8288 ALE activation relative to the $\overline{S0}$-$\overline{S2}$ status from the CPU. The maximum mode timing diagram specifies two possible delay paths to generate ALE. The first is TCHSV + TSVLH measured from the rising edge of the clock cycle preceding T1. The second path is TCLLH measured from the start of T1. Since the 8288 initiates a bus cycle from the status lines leaving the passive state ($\overline{S0}$-$\overline{S2}$ = 1), if the 8086 is late in issuing the status (TCHSVmax) while the clock high time is a minimum (TCHCLmin), the status will not have changed by the start of T1 and ALE is issued TSVLH ns after the status changes. If the status changes prior to the beginning of T1, the 8288 will not issue the ALE until TCLLH ns after the start of T1. The resulting worst case delay to enable ALE (relative to the start of T1) is TCHSVmax + TSVLHmax – TCHCLmin = 58 ns. Note, when calculating signal relationships, be sure to use the proper maximum mode values rather than equivalent minimum mode values.

The trailing edge of ALE is triggered in the 8288 by the positive clock edge in T1 regardless of the delay to enable ALE. The resulting minimum ALE pulse width is TCLCHmax – 58 ns = 75 ns assuming TCHLL = 0. TCLCHmax must be used since TCHCLmin was assumed to derive the 58 ns ALE enable delay. The address is guaranteed to be valid TCLCHmin + TCHLLmin – TCLAVmax = 8 ns prior to the trailing edge

of ALE to capture the address in the 8282 or 8283 latches. Again we have assumed a very conservative TCHLL = 0. Note, since the address and ALE are driven by separate devices, no tracking of A.C. characteristics can be assumed.

The address hold time to the latches is guaranteed by the address remaining valid until the end of T1 while ALE is disabled a maximum of 15 ns from the positive clock transition in T1 (TCHCLmin – TCHLLmax = 52 ns address hold time). The multiplexed bus transitions from address to status and write data or three-state (for read) are identical to the minimum mode timing. Also, since the address valid delay (TCLAV) remains the critical path in establishing a valid address, the address access times to valid data and ready are the same as the minimum mode system.

2. Read Cycle Timing

The maximum mode system offers read signals generated by both the 8086 and the 8288. The 8086 $\overline{RD}$ output signal timing is identical to the minimum mode system. Since the A.C. characteristics of the read commands generated by the 8288 are significantly better than the 8086 output, access to devices on the demultiplexed buffered system bus should use the 8288 commands. The 8086 $\overline{RD}$ signal is available for devices which reside directly on the multiplexed bus. The following evaluations for read, write and interrupt acknowledge only consider the 8288 command timing.

The 8288 provides separate memory and I/O read signals which conform to the same A.C. characteristics. The commands are issued TCLML ns after the start of T2 and terminate TCLMH ns after the start of T4. The minimum command length is 2TCLCL – TCLMLmax + TCLMLmin = 375 ns. The access time to valid data at the CPU is 2TCLCL – TCLMLmax – TDVCLmax = 335 ns. Since the 8288 was designed for systems with buffered data busses, the commands are enabled before the CPU has three-stated the multiplexed bus and should not be used with devices which reside directly on the multiplexed bus (to do so could result in bus contention during 8086 bus float and device turn-on).

The direction control for data bus transceivers is established in T1 while the transceivers are not enabled by DEN until the positive clock transition of T2. This provides TCLCH + TCVNVmin = 123 ns for 8086 bus float delay and TCHCLmin + TCLCL – TCVNVmax – TDVCLmax = 187 ns of transceiver active to data valid at the CPU. Since both DEN and command are valid a minimum of 10 ns into T4, the CPU data hold time TCLDX is guaranteed. A maximum DEN disable of 45 ns (TCVNX max) guarantees the transceivers are disabled by the start of the next 8086 bus cycle (215 ns minimum from the same clock edge). On the positive clock transition of T4, DT/$\overline{R}$ is returned to transmit in preparation for a possible write operation on the next bus cycle. Since the system memory and I/O devices reside on a buffered system bus, they must three-state their outputs before the device for the next bus cycle is selected (approximately 2TCLCL) or the transceivers drive write data onto the bus (approximately 2TCLCL).

3. Write Cycle Timing

In the maximum mode, the 8288 provides normal and advanced write commands for memory and I/O. The advanced write commands are active a full clock cycle ahead of the normal write commands and have timing identical to the read commands. The advanced write pulse width is 2TCLCL – TCLMLmax + TCLMHmin = 375 ns while the normal write pulse width is TCLCL – TCLMLmax + TCLMHmin = 175 ns. Write data setup time to the selected device is a function of either the data valid delay from the 8086 (TCLDV) or the transceiver enable delay TCVNV. The worst case delay to valid write data is TCLDV = 110 ns minus transceiver propagation delays. This implies the data may not be valid until 100 ns after the advanced write command but will be valid approximately TCLCL – TCLDVmax + TCLMLmin = 100 ns prior to the leading edge of the normal write command. Data will be valid 2TCLCL – TCLDVmax + TCLMHmin = 300 ns before the trailing edge of either write command. The data and command overlap for the advanced command is 300 ns while the overlap with the normal write command is 175 ns. The transceivers are disabled a minimum of TCLCHmin – TCLMHmax + TCVNXmin = 85 ns after the write command while the CPU provides valid data a minimum of TCLCHmin – TCLMHmax + TCHDZmin = 85 ns. This guarantees write data hold of 85 ns after the write command. The transceivers are disabled TCLCL – TCVNXmax + TCHDTLmin = 155 ns (assuming TCHDTL = 0) prior to transceiver direction change for a subsequent read cycle.

4. Interrupt Acknowledge Timing

The maximum mode $\overline{INTA}$ sequence is logically identical to the minimum mode sequence. The transceiver control (DEN and DT/$\overline{R}$) and $\overline{INTA}$ command timing of each interrupt acknowledge cycle is identical to the read cycle. As in the minimum mode system, the multiplexed address/data bus will float from the leading edge of T1 for each $\overline{INTA}$ bus cycle and not be driven by the CPU until after T4 of each $\overline{INTA}$ cycle. The setup and hold times on the vector number for the second cycle are the same as data setup and hold for the read. If the device providing the interrupt vector number is connected to the local bus, TCLCL – TCLAZmax + TCLMLmin = 130 ns are available from 8086 bus float to $\overline{INTA}$ command active. The selected device on the local bus must disable the system data bus transceivers since DEN is still generated by the 8288.

If the 8288 is not in the IOB (I/O Bus) mode, the 8288 MCE/$\overline{PDEN}$ output becomes the MCE output. This output is active during each $\overline{INTA}$ cycle and overlaps the ALE signal during T1. The MCE is available for gating cascade addresses from a master 8259A onto three of the upper AD15-AD8 lines and allowing ALE to latch the cascade address into the address latches. The address lines may then be used to provide CAS address selection to slave 8259A's located on the system bus (reference Figure 3E5). MCE is active within 15 ns of status or the start of T1 for each $\overline{INTA}$ cycle. MCE should not enable the CAS lines onto the multiplexed bus during the first cycle since the CPU does not guarantee to float

the bus until 80 ns into the first $\overline{INTA}$ cycle. The first MCE can be inhibited by gating MCE with $\overline{LOCK}$. The 8086 $\overline{LOCK}$ output is activated during T2 of the first cycle and disabled during T2 of the second cycle. The overlap of $\overline{LOCK}$ with MCE allows the first MCE to be masked and the second MCE to gate the cascade address onto the local bus. Since the 8259A will not provide a cascade address until the second cycle, no information is lost. As with ALE, MCE is guaranteed valid within 58 ns of the start of T1 to allow 75 ns CAS address setup to the trailing edge of ALE. MCE remains active TCHCLmin – TCHLLmax + TCLMCLmin = 52 ns after ALE to provide data hold time to the latches.

If the 8288 is strapped in the IOB mode, the MCE output becomes $\overline{PDEN}$ and all I/O references are assumed to be devices on the local bus rather than the demultiplexed system bus. Since $\overline{INTA}$ cycles are considered I/O cycles, all interrupts are assumed to come from the local system and cascade addresses are not gated onto the system address bus. Additionally, the DEN signal is not enabled since no I/O transfers occur on the system bus. If the local I/O bus is also buffered by transceivers, the $\overline{PDEN}$ signal is used to enable those transceivers. $\overline{PDEN}$ A.C. characteristics are identical to DEN with $\overline{PDEN}$ enabled for I/O references and DEN enabled for instruction or memory data references.

5. Ready Timing

Ready timing based on address valid timing is the same for maximum and minimum mode systems. The delay from 8288 command valid to RDY valid at the 8284 is TCLCL – TCLMLmax – TRIVCLmin = 130 ns. This time is available for external circuits to determine the need to insert wait states and disable RDY or enable RDY to avoid wait states. $\overline{INTA}$, all read commands and advanced write commands provide this timing. The normal write command is not valid until after the RDY signal must be valid. Since both normal and advanced write commands are generated by the 8288 for all write cycles, the advanced write may be used to generate a RDY indication even though the selected device uses the normal write command.

Since separate commands are provided for memory and I/O, no M/$\overline{IO}$ signal is specifically available as in the minimum mode to allow an early 'wait state required' indication for I/O devices. The $\overline{S2}$ status line, however is logically equivalent to the M/$\overline{IO}$ signal and can be used for this purpose.

6. Other Considerations

The $\overline{RQ}$/$\overline{GT}$ timing is covered in the next section and will not be duplicated here. The only additional signals to be considered in the maximum mode are the queue status lines QS0, QS1. These signals are changed on the leading edge of each clock cycle (high to low transition) including idle and wait cycles (the queue status is independent of the bus activity). External logic may sample the lines on the low to high transition of each clock cycle. When sampled, the signals indicate the queue activity in the previous clock cycle and therefore lag the CPU's activity by one cycle. The $\overline{TEST}$ input requirements are identical to those stated for the minimum mode.

To inform the 8288 of HALT status when a HALT instruction is executed, the 8086 will initiate a status transition from passive to HALT status. The status change will cause the 8288 to emit an ALE pulse with an indeterminate address. Since no bus cycle is initiated (no command is issued), the results of this address will not affect CPU operation (i.e., no response such as READY is expected from the system). This allows external hardware to latch and decode all transitions in system status.

3G. Bus Control Transfer (HOLD/HLDA and $\overline{RQ}$/$\overline{GT}$)

The 8086 supports protocols for transferring control of the local bus between itself and other devices capable of acting as bus masters. The minimum mode configuration offers a signal level handshake similar to the 8080 and 8085 systems. The maximum mode provides an enhanced pulse sequence protocol designed to optimize utilization of CPU pins while extending the system configurations to two prioritized levels of alternate bus masters. These protocols are simply techniques for arbitration of control of the CPU's local bus and should not be confused with the need for arbitration of a system bus.

1. MINIMUM MODE

The minimum mode 8086 system uses a hold request input (HOLD) to the CPU and a hold acknowledge (HLDA) output from the CPU. To gain control of the bus, a device must assert HOLD to the CPU and wait for the HLDA before driving the bus. When the 8086 can relinquish the bus, it floats the $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ and M/$\overline{IO}$ command lines, the $\overline{DEN}$ and DT/$\overline{R}$ bus control lines and the multiplexed address/data/status lines. The ALE signal is not three-stated. The CPU acknowledges the request with HLDA to allow the requestor to take control of the bus. The requestor must maintain the HOLD request active until it no longer requires the bus. The HOLD request to the 8086 directly affects the bus interface unit and only indirectly affects the execution unit. The CPU will continue to execute from its internal queue until either more instructions are needed or an operand transfer is required. This allows a high degree of overlap between CPU and auxiliary bus master operation. When the requestor drops the HOLD signal, the 8086 will respond by dropping HLDA. The CPU will not re-drive the bus, command and control signals from three-state until it needs to perform a bus transfer. Since the 8086 may still be executing from its internal queue when HOLD drops, there may exist a period of time during which no device is driving the bus. To prevent the command lines from drifting below the minimum VIH level during the transition of bus control, 22K ohm pull up resistors should be connected to the bus command lines. The timing diagram in Figure 3G1 shows the handshake sequence and 8086 timing to sample HOLD, float the bus, and enable/disable HLDA relative to the CPU clock.

To guarantee valid system operation, the designer must assure that the requesting device does not assert con-

trol of the bus prior to the 8086 relinquishing control and that the device relinquishes control of the bus prior to the 8086 driving the bus. The HOLD request into the 8086 must be stable THVCH ns prior to the CPU's low to high clock transition. Since this input is not synchronized by the CPU, signals driving the HOLD input should be synchronized with the CPU clock to guarantee the setup time is not violated. Either clock edge may be used. The maximum delay between HLDA and the 8086 floating the bus is TCLAZmax – TCLHAVmin = 70 ns. If the system cannot tolerate the 70 ns overlap, HLDA active from the 8086 should be delayed to the device. The minimum delay for the CPU to drive the control bus from HOLD inactive is THVCHmin + 3TCLCL = 635 ns and THVCHmin + 3TCLCL + TCHCL = 701 ns to drive the multiplexed bus. If the device does not satisfy these requirements, HOLD inactive to the 8086 should be delayed. The delay from HLDA inactive to driving the busses is TCLCL + TCLCHmin – TCLHAVmax = 158 ns for the control bus and 2TCLCL – TCLHAVmax = 240 ns for the data bus.

1.1 Latency of HLDA to HOLD

The decision to respond to a HOLD request is made in the bus interface unit. The major factors that influence the decision are the current bus activity, the state of the $\overline{\text{LOCK}}$ signal internal to the CPU (activated by the software LOCK prefix) and interrupts.

If the $\overline{\text{LOCK}}$ is not active, an interrupt acknowledge cycle is not in progress and the BIU (Bus Interface Unit) is executing a T4 or TI when the HOLD request is received, the minimum latency to HLDA is:

35 ns	THVCH min (Hold setup)
65 ns	TCHCL min
200 ns	TCLCL (bus float delay)
10 ns	TCLHAV min (HLDA delay)
310 ns	@ 5 MHz

The maximum delay under these conditions is:

34 ns	(just missed setup time)
200 ns	delay to next sample
82 ns	TCHCL max
200 ns	TCLCL (bus float delay)
160 ns	TCLHAV max (HLDA delay)
677 ns	@ 5 MHz

If the BIU just initiated a bus cycle when the HOLD Request was received, the worst case response time is:

34 ns	THVCH (just missed)
82 ns	TCHCL max
7*200	bus cycle execution
N*200	N wait states/bus cycle
160 ns	TCLHAV max (HLDA delay)
1.676 μs	@ 5 MHz, no wait states

Note, the 200 ns delay for just missing is included in the delay for bus cycle execution. If the operand transfer is a word transfer to an odd byte boundary, two bus cycles are executed to perform the transfer. The BIU will not acknowledge a HOLD request between the two bus cycles. This type of transfer would extend the above maximum latency by four additional clocks plus N additional wait states. With no wait states in the bus cycle, the maximum would be 2.476 microseconds.

Although the minimum mode 8086 does not have a hardware $\overline{\text{LOCK}}$ output, the software LOCK prefix may still be included in the instruction stream. The CPU internally reacts to the LOCK prefix as would the maximum mode 8086. Therefore, the LOCK does not allow a HOLD request to be honored until completion of the instruction following the prefix. This allows an instruction which performs more than one memory reference (ex. ADD [BX], CX; which adds CX to [BX]) to execute without another bus master gaining control of the bus between memory references. Since the LOCK signal is active for one clock longer than the instruction execution, the maximum latency to HLDA is:

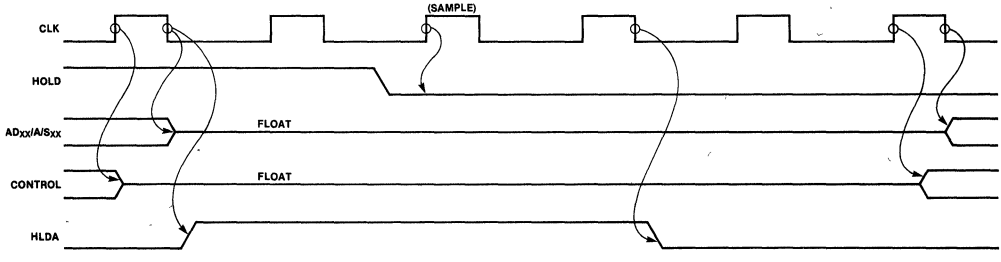


Figure 3G1. HOLD/HLDA Sequence

34 ns	THVCH (just miss)
200 ns	delay to next sample
82 ns	TCHCL max
(M + 1)*200 ns	LOCK instruction execution
200 ns	set up HLDA (internal)
160 ns	TCLHAV max (HLDA delay)
(M*200 ns) + 876 ns	@ 5 MHz

If the HOLD request is made at the beginning of an interrupt acknowledge sequence, the maximum latency to HLDA is:

34 ns	THVCH (just missed)
82 ns	TCHCL max
2600 ns	13 clock cycles for INTA
160 ns	TCLHAV max
2.876 μs	@ 5 MHz

1.2 Minimum Mode DMA Configuration

A typical use of the HOLD/HLDA signals in the minimum mode 8086 system is bus control exchange with DMA devices like the Intel 8257-5 or 8237 DMA controllers. Figure 3G2 gives a general interconnect for this type of configuration using the 8237-2. The DMA controller resides on the upper half of the 8086's local bus and shares the A8-A15 demultiplexing address latch of the 8086. All registers in the 8237-2 must be assigned odd addresses to allow initialization and interrogation by the CPU over the upper half of the data bus. The 8086 $\overline{RD}/\overline{WR}$ commands must be demultiplexed to provide separate I/O and memory commands which are compatible with the 8237-2 commands. The AEN control from the 8237-2 must disable the 8086 commands from the command bus, disable the address latches from the lower (A0-A7) and upper (A19-A16) address bus and select the 8237-2 address strobe (ADSTB) to the A8-A15 address latch. If the data bus is buffered, a pull-up resistor on the DEN line will keep the buffers disabled. The DMA controller will only transfer bytes between

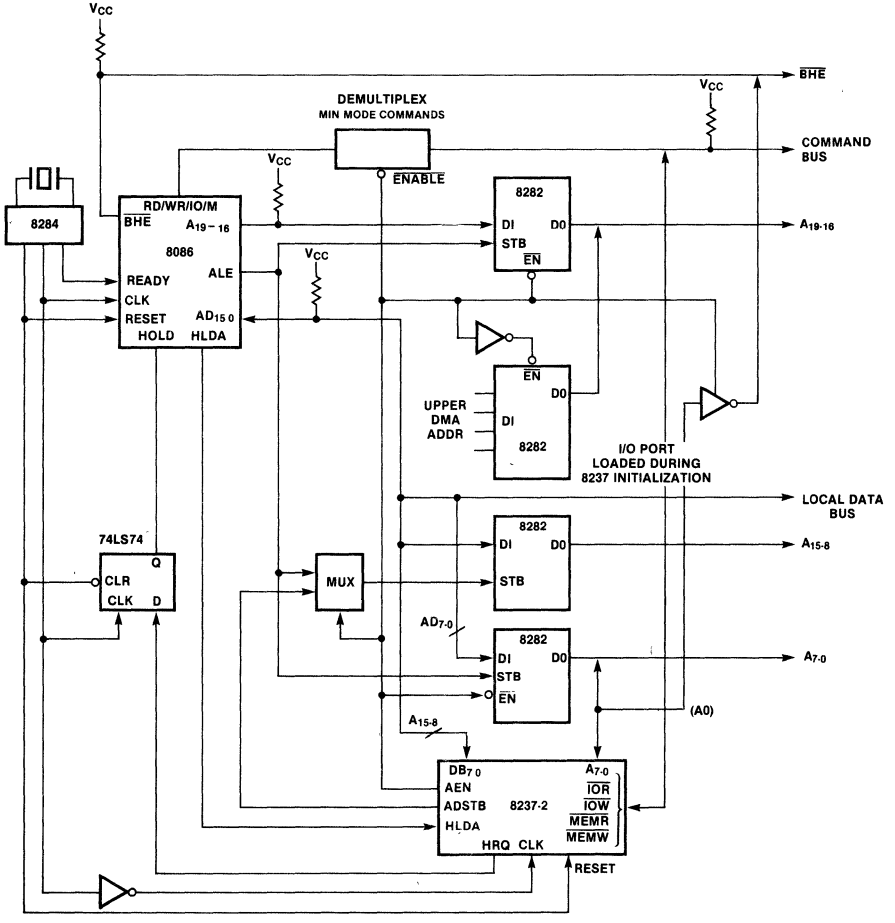


Figure 3G2. DMA Using the 8237-2

memory and I/O and requires the I/O devices to reside on an 8-bit bus derived from the 16-bit to 8-bit bus multiplex circuit given in Section 4. Address lines A7-A0 are driven directly by the 8237 and $\overline{BHE}$ is generated by inverting A0. If A19-A16 are used, they must be provided by an additional port with either a fixed value or initialized by software and enabled onto the address bus by AEN.

Figure 3G3 gives an interconnection for placing the 8257 on the system bus. By using a separate latch to hold the upper address from the 8257-5 and connecting the outputs to the address bus as shown, 16-bit DMA transfers are provided. In this configuration, AEN simultaneously enables A0 and $\overline{BHE}$ to allow word transfers. AEN still disables the CPU interface to the command and address busses.

2. MAXIMUM MODE ($\overline{RQ}/\overline{GT}$)

The maximum mode 8086 configuration supports a significantly different protocol for transferring bus control. When viewed with respect to the HOLD/HLDA sequence of the minimum mode, the protocol appears difficult to implement externally. However, it is necessary to understand the intent of the protocol and its purpose within the system architecture.

2.1 Shared System Bus ($\overline{RQ}/\overline{GT}$ Alternative)

The maximum mode $\overline{RQ}/\overline{GT}$ sequence is intended to transfer control of the CPU local bus between the CPU and alternate bus masters which reside totally on the local bus and share the complete CPU interface to the system bus. The complete interface includes the address latches, data transceivers, 8288 bus controller and 8289 multi master bus arbiter. If the alternate bus masters in the system do not reside directly on the 8086 local bus, system bus arbitration is required rather than local CPU bus arbitration. To satisfy the need for multimaster system bus arbitration at each CPU's system interface, the 8289 bus arbiter should be used rather than the CPU $\overline{RQ}/\overline{GT}$ logic.

To allow a device with a simple HOLD/HLDA protocol to gain control of a single CPU system bus, the circuit in Figure 3G4 could be used. The design is effectively a simple bus arbiter which isolates the CPU from the system bus when an alternate bus master issues a HOLD request. The output of the circuit, $\overline{AEN}$ (Address ENable), disables the 8288 and 8284 when the 8086 indicates idle status ($\overline{S0}$,$\overline{S1}$,$\overline{S2}$ = 1), $\overline{LOCK}$ is not active and a HOLD request is active. With $\overline{AEN}$ inactive, the 8288 three-states the command outputs and disables DEN

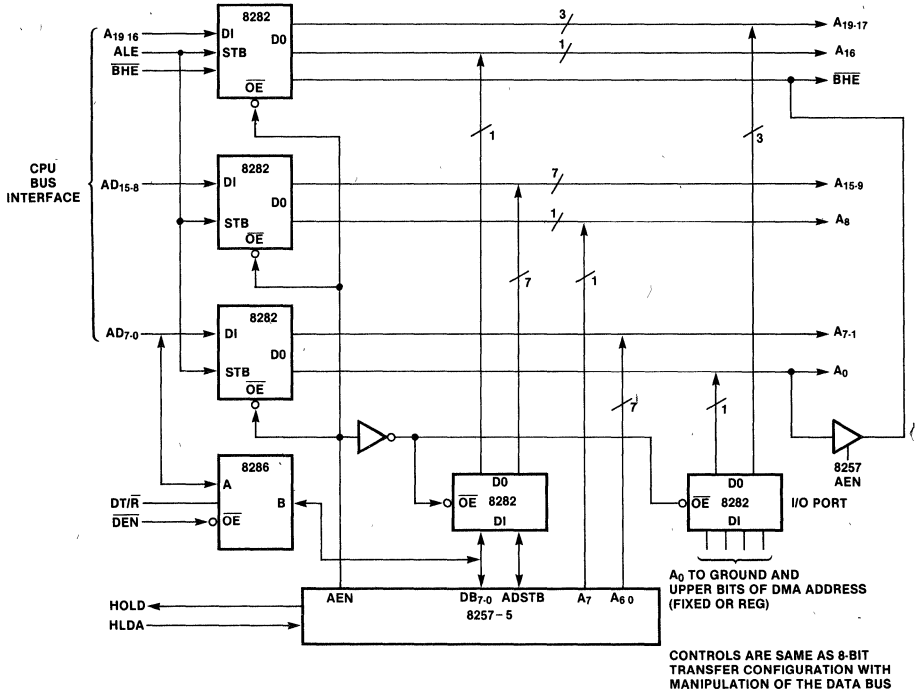


Figure 3G3. 8086 Min System, 8257 on System Bus 16-Bit Transfers

which three-states the data bus transceivers. $\overline{AEN}$ must also three-state the address latch (8282 or 8283) outputs. These actions remove the 8086 from the system bus and allow the requesting device to drive the system bus. The $\overline{AEN}$ signal to the 8284 disables the ready input and forces a bus cycle initiated by the 8086 to wait until the 8086 regains control of the system bus. The CPU may actively drive its local bus during this interval.

The requesting device will not gain control of the bus during an 8086 initiated bus cycle, a locked instruction or an interrupt acknowledge cycle. The $\overline{LOCK}$ signal from the 8086 is active between $\overline{INTA}$ cycles to guarantee the CPU maintains control of the bus. Unlike the minimum mode 8086 HOLD response, this arbitration circuit allows the requestor to gain control of the bus between consecutive bus cycles which transfer a word operand on an odd address boundary and are not locked. Depending on the characteristics of the requesting device, any of the 74LS74 outputs can be used to generate a HLDA to the device.

Upon completion of its bus operations, the alternate bus master must relinquish control of the system bus and drop the HOLD request. After $\overline{AEN}$ goes inactive, the address latches and data transceivers are enabled but, if a CPU initiated bus cycle is pending, the 8288 will not drive the command bus until a minimum of 105 ns or maximum of 275 ns later. If the system is normally not ready, the 8284 $\overline{AEN}$ input may immediately be enabled with ready returning to the CPU when the selected device completes the transfer. If the system is normally ready, the 8284 $\overline{AEN}$ input must be delayed long enough to provide access time equivalent to a normal bus cycle. The 74LS74 latches in the design provide a minimum of TCLCHmin for the alternate device to float the system bus after releasing HOLD. They also provide 2TCLCL ns address access and 2TCLCL – TAEVCHmax ns (8288 command enable delay) command access prior to enabling 8284 ready detection. If HLDA is generated as shown in Figure 3G4, TCLCL ns are available for the 8086 to release the bus prior to issuing HLDA while HLDA is dropped almost immediately upon loss of HOLD.

A circuit configuration for an 8257-5 using this technique to interface with a maximum mode 8086 can be derived from Figure 3G3. The 8257-5 has its own address latch for buffering the address lines A15-A8 and uses its $\overline{AEN}$ output to enable the latch onto the address bus. The maximum latency from HOLD to HLDA for this circuit is dependent on the state of the system when the HOLD is issued. For an idle system the maximum delay is the propagation delay through the nand gate and R/S flip-flop (TD1) plus 2TCLCL plus TCLCHmax plus propagation delay of the 74LS74 and 74LS02 (TD2). For a locked instruction it becomes: TD1 + TD2 + (M + 2) *TCLCL + TCLCHmax where M is the number of clocks required for execution of the locked instruction. For the interrupt acknowledge cycle the latency is TD1 + TD2 + 9 *TCLCL + TCLCHmax.

2.2 Shared Local Bus ($\overline{RQ}/\overline{GT}$ Usage)

The $\overline{RQ}/\overline{GT}$ protocol was developed to allow up to two instruction set extension processors (co-processors) or other special function processors (like the 8089 I/O processor in local mode) to reside directly on the 8086 local bus. Each $\overline{RQ}/\overline{GT}$ pin of the 8086 supports the full protocol for exchange of bus control (Fig. 3G5). The sequence consists of a request from the alternate bus master to gain control of the system bus, a grant from the CPU to indicate the bus has been relinquished and a release pulse from the alternate master when done. The two $\overline{RQ}/\overline{GT}$ pins ($\overline{RQ}/\overline{GT0}$ and $\overline{RQ}/\overline{GT1}$) are prioritized with $\overline{RQ}/\overline{GT0}$ having the highest priority. The prioritization only occurs if requests have been received on both pins before a response has been given to either. For example, if a request is received on $\overline{RQ}/\overline{GT1}$ followed by a request on $\overline{RQ}/\overline{GT0}$ prior to a grant on $\overline{RQ}/\overline{GT1}$, $\overline{RQ}/\overline{GT0}$ will gain priority over $\overline{RQ}/\overline{GT1}$. However, if $\overline{RQ}/\overline{GT1}$ had already received a grant, a request on $\overline{RQ}/\overline{GT0}$ must wait until a release pulse is received on $\overline{RQ}/\overline{GT1}$.

The request/grant sequence interaction with the bus interface unit is similar to HOLD/HLDA. The CPU continues to execute until a bus transfer for additional instructions or data is required. If the release pulse is

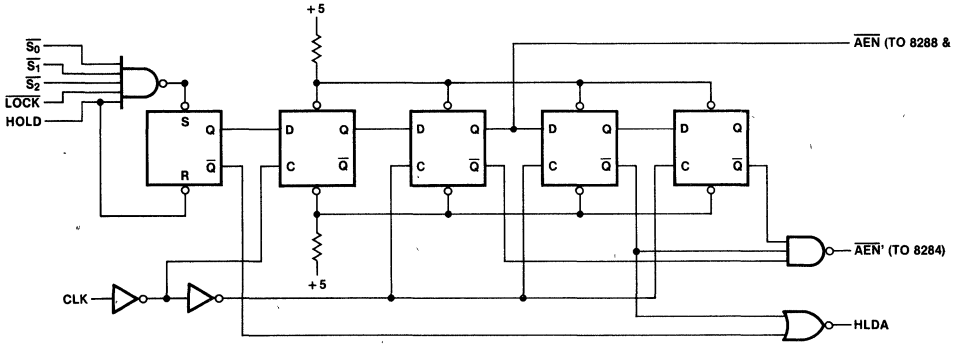


Figure 3G4. Circuit to Translate HOLD into AEN Disable for Max Mode 8086

received before the CPU needs the bus, it will not drive the bus until a transfer is required.

Upon receipt of a request pulse, the 8086 floats the multiplexed address, data and status bus, the $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ status lines, the $\overline{LOCK}$ pin and $\overline{RD}$. This action does not disable the 8288 command outputs from driving the command bus and does not disable the address latches from driving the address bus. The 8288 contains internal pull-up resistors on the $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ status lines to maintain the passive state while the 8086 outputs are three-state. The passive state prevents the 8288 from initiating any commands or activating DEN to enable the transceivers buffering the data bus. If the device issuing the $\overline{RQ}$ does not use the 8288, it must disable the 8288 command outputs by disabling the 8288 $\overline{AEN}$ input. Also, address latches not used by the requesting device must be disabled.

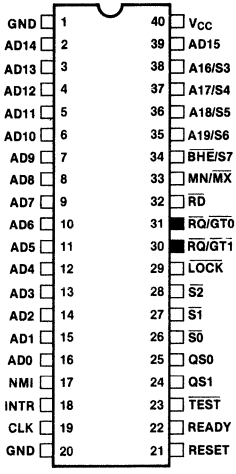


Figure 3G5. 8086 RQ/GT Connections

2.3 $\overline{RQ}/\overline{GT}$ Operation

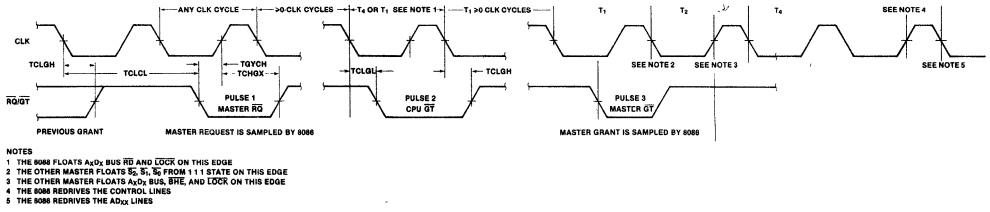
Detailed timing of the $\overline{RQ}/\overline{GT}$ sequence is given in Figure 3G6. To request a transfer of bus control via the $\overline{RQ}/\overline{GT}$ lines, the device must drive the line low for no more than one CPU clock interval to generate a request pulse. The pulse must be synchronized with the CPU clock to guarantee the appropriate setup and hold times to the clock edge which samples the $\overline{RQ}/\overline{GT}$ lines in the CPU. After issuing a request pulse, the device must begin sampling for a grant pulse with the next low to high clock edge. Since the 8086 can respond with a grant pulse in the clock cycle immediately following the request, the $\overline{RQ}/\overline{GT}$ line may not return to the positive level between the request and grant pulses. Therefore edge triggered logic is not valid for capturing a grant pulse. It also implies the circuitry which generates the request pulse must guarantee the request is removed in time to detect a grant from the CPU. After receiving the grant pulse, the requesting device may drive the local bus. Since the 8086 does not float the address and data bus, $\overline{LOCK}$ or $\overline{RD}$ until the high to low clock transition following the low to high clock transition the requestor uses to sample for the grant, the requestor should wait the float delay of the 8086 (TCLAZ) before driving the local bus. This precaution prevents bus contention during the access of bus control by the requestor.

To return control of the bus to the 8086, the alternate bus master relinquishes bus control and issues a release pulse on the same $\overline{RQ}/\overline{GT}$ line. The 8086 may drive the $\overline{S0}$-$\overline{S2}$ status lines, $\overline{RD}$ and $\overline{LOCK}$, three clock cycles after detecting the release pulse and the address/data bus TCHCLmin ns (clock high time) after the status lines. The alternate bus master should be three-stated off the local bus and have other 8086 interface circuits (8288 and address latches) re-enabled within the 8086 delay to regain control of the bus.

2.4 $\overline{RQ}/\overline{GT}$ Latency

The $\overline{RQ}$ to $\overline{GT}$ latency for a single $\overline{RQ}/\overline{GT}$ line is similar to the HOLD to HLDA latency. The cases given for the minimum mode 8086 also apply to the maximum mode. For each case the delay from $\overline{RQ}$ detection by the CPU to $\overline{GT}$ detection by the requestor is:

(HOLD to HLDA delay) – (THVCH + TCHCL + TCLHAV)



NOTES
1 THE 8086 FLOATS A_{XD_X} BUS $\overline{RD}$ AND $\overline{LOCK}$ ON THIS EDGE
2 THE OTHER MASTER FLOATS $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$ FROM 1 1 1 STATE ON THIS EDGE
3 THE OTHER MASTER FLOATS A_{XD_X} BUS, $\overline{BHE}$, AND $\overline{LOCK}$ ON THIS EDGE
4 THE 8086 REDRIVES THE CONTROL LINES
5 THE 8086 REDRIVES THE AD_{XX} LINES

Figure 3G6. Request/Grant Sequence

This gives a clock cycle maximum delay for an idle bus interface. All other cases are the minimum mode result minus 476 ns. If the 8086 has previously issued a grant on one of the $\overline{RQ}/\overline{GT}$ lines, a request on the other $\overline{RQ}/\overline{GT}$ line will not receive a grant until the first device releases the interface with a release pulse on its $\overline{RQ}/\overline{GT}$ line. The delay from release on one $\overline{RQ}/\overline{GT}$ line to a grant on the other is typically one clock period as shown in Figure 3G7. Occasionally the delay from a release on $\overline{RQ}/\overline{GT1}$ to a grant on $\overline{RQ}/\overline{GT0}$ will take two clock cycles and is a function of a pending request for transfer of control from the execution unit. The latency from request to grant when the interface is under control of a bus master on the other $\overline{RQ}/\overline{GT}$ line is a function of the other bus master. The protocol embodies no mechanism for the CPU to force an alternate bus master off the bus. A watchdog timer should be used to prevent an errant alternate bus master from 'hanging' the system.

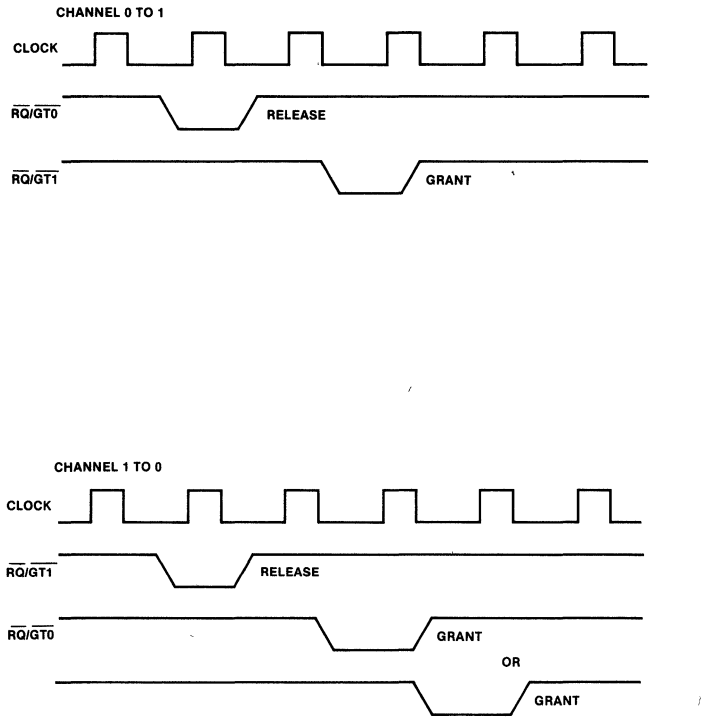


Figure 3G7. Channel Transfer Delay

2.5 RQ/GT to HOLD/HLDA Conversion

A circuit for translating a HOLD/HLDA hand-shake sequence into a $\overline{RQ}/\overline{GT}$ pulse sequence is given in Figure 3G8. After receiving the grant pulse, the HLDA is enabled TCHCLmin ns before the CPU has three-stated the bus. If the requesting circuit drives the bus within 20 ns of HLDA, it may be desirable to delay the acknowledge one clock period. The HLDA is dropped no later than one clock period after HOLD is disabled. The HLDA also drops at the beginning of the release pulse to provide 2TCLCL + TCLCH for the requestor to relinquish control of the status lines and 3TCLCL to float the remaining signals.

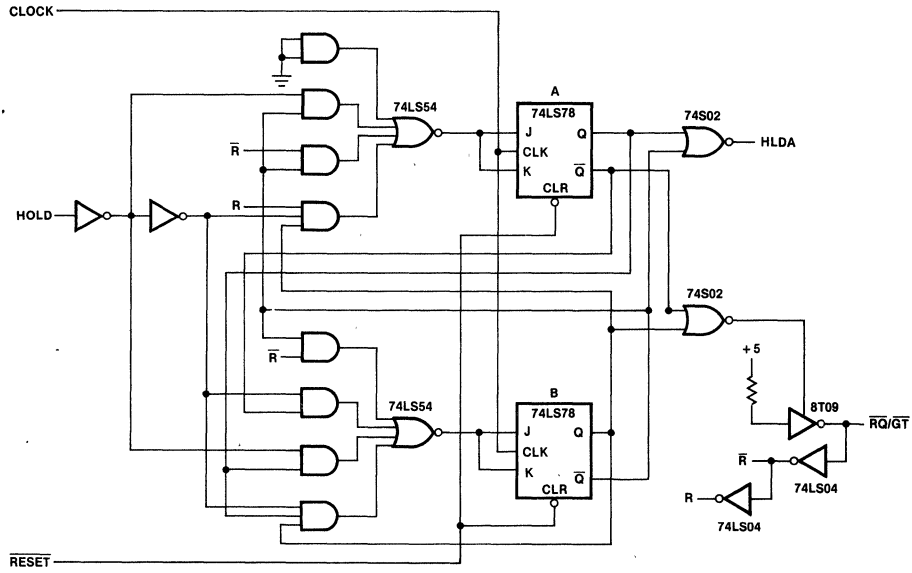


Figure 3G8a. HOLD/HLDA◄►$\overline{RQ}/\overline{GT}$ Conversion Circuit

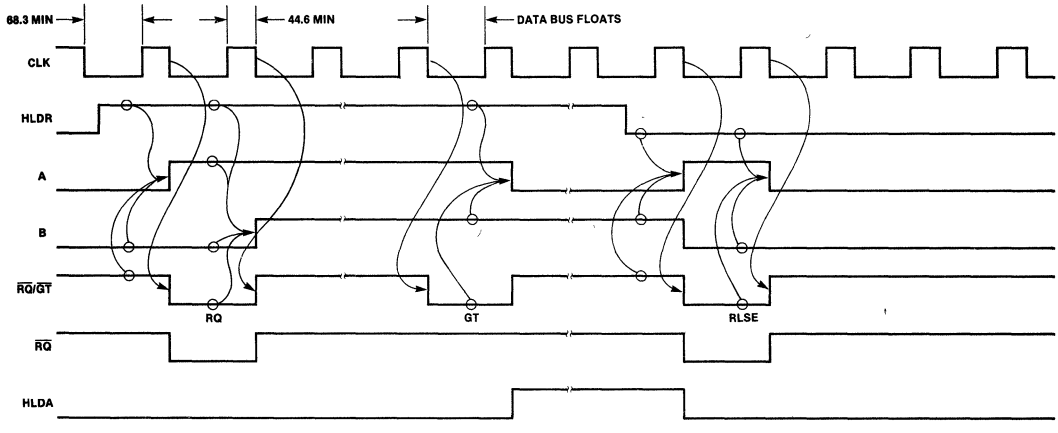


Figure 3G8b. HOLD/HLDA◄►$\overline{RQ}/\overline{GT}$ Conversion Timing

4. INTERFACING WITH I/O

The 8086 is capable of interfacing with 8- and 16-bit I/O devices using either I/O instructions or memory mapped I/O. The I/O instructions allow the I/O devices to reside in a separate I/O address space while memory mapped I/O allows the full power of the instruction set to be used for I/O operations. Up to 64K bytes of I/O mapped I/O may be defined in an 8086 system. To the programmer, the separate I/O address space is only accessible with INPUT and OUTPUT commands which transfer data between I/O devices and the AX (for 16-bit data transfers) or AL (for 8-bit data transfers) register. The first 256 bytes of the I/O space (0 to 255) are directly addressable by the I/O instructions while the entire 64K is accessible via register indirect addressing through the DX register. The later technique is particularly desirable for service procedures that handle more than one device by allowing the desired device address to be passed to the procedure as a parameter. I/O devices may be connected to the local CPU bus or the buffered system bus.

4A. Eight-Bit I/O

Eight-bit I/O devices may be connected to either the upper or lower half of the data bus. Assigning an equal number of devices to the upper and lower halves of the bus will distribute the bus loading. If a device is connected to the upper half of the data bus, all I/O addresses assigned to the device must be odd (A0 = 1). If the device is on the lower half of the bus, its addresses must be even (A0 = 0). The address assignment directs the eight-bit transfer to the upper (odd byte address) or lower (even byte address) half of the sixteen-bit data bus. Since A0 will always be a one or zero for a specific device, A0 cannot be used as an address input to select registers within a specific device. If a device on the upper half of the bus and one on the lower half are assigned addresses that differ only in A0 (adjacent odd and even addresses), A0 and $\overline{BHE}$ must be conditions of chip select decode to prevent a write to one device from erroneously performing a write to the other. Several techniques for generating I/O device chip selects are given in Figure 4A1.

The first technique (a) uses separate 8205's to generate chip selects for odd and even addressed byte peripherals. If a word transfer is performed to an even addressed device, the adjacent odd addressed I/O device is also selected. This allows accessing the devices individually with byte transfers or simultaneously as a 16-bit device with word transfers. Figure 4A1(b) restricts the chip selects to byte transfers, however a word transfer to an odd address will cause the 8086 to run two byte transfers that the decode technique will not detect. The third technique simply uses a single 8205 to generate odd and even device selects for byte transfers and will only select the even addressed eight-bit device on a word transfer to an even address.

If greater than 256 bytes of the I/O space or memory mapped I/O is used, additional decoding beyond what is shown in the examples may be necessary. This can be done with additional TTL, 8205's or bipolar PROMs (Intel's 3605A). The bipolar PROMs are slightly slower than multiple levels of TTL (50 ns vs 30 to 40 ns for TTL) but provide full decoding in a single package and allow inserting a new PROM to reconfigure the system I/O map without circuit board or wiring modifications (Fig. 4A2).

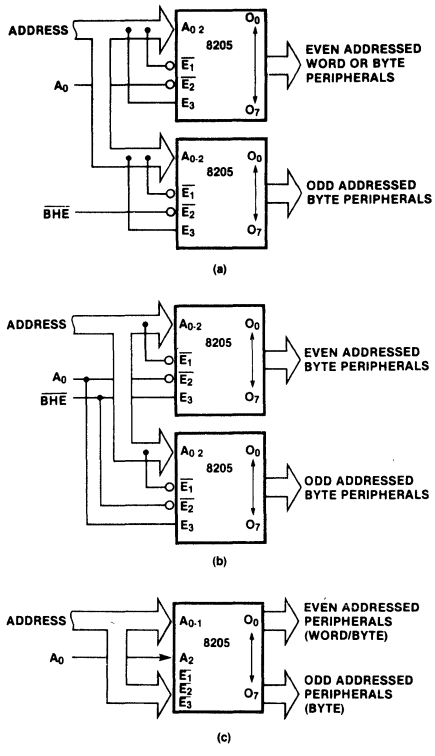


Figure 4A1. Techniques for I/O Device Chip Selects

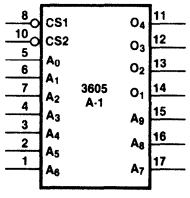


Figure 4A2. Bipolar PROM Decoder

One last technique for interfacing with eight-bit peripherals is considered in Figure 4A3. The sixteen-bit data bus is multiplexed onto an eight-bit bus to accommodate byte oriented DMA or block transfers to memory mapped eight-bit I/O. Devices connected to this interface may be assigned a sequence of odd and even addresses rather than all odd or even.

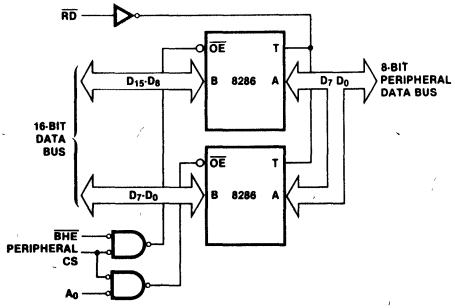
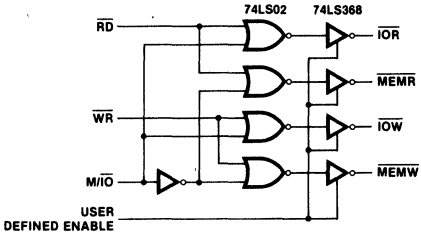


Figure 4A3. 16- to 8-Bit Bus Conversion



NOTE: IF IT IS NOT NECESSARY TO THREE-STATE THE COMMAND LINES, A DECODER (8205 OR 74S138) COULD BE USED. THE 74LS257 IS NOT RECOMMENDED SINCE THE OUTPUTS MAY EXPERIENCE VOLTAGE SPIKES WHEN ENTERING OR LEAVING THREE-STATE.

Figure 4C1. Decoding Memory and I/O $\overline{RD}$ and $\overline{WR}$ Commands for Minimum Mode 8086 Systems

4B. Sixteen-Bit I/O

For obvious reasons of efficient bus utilization and simplicity of device selection, sixteen-bit I/O devices should be assigned even addresses. To guarantee the device is selected only for word operations, A0 and $\overline{BHE}$ should be conditions of chip select code (Fig. 4B1).

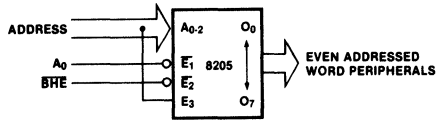


Figure 4B1. Sixteen-Bit I/O Decode

4C. General Design Considerations

MIN/MAX, MEMORY I/O MAPPED AND LINEAR SELECT

Since the minimum mode 8086 has common read and write commands for memory and I/O, if the memory and I/O address spaces overlap, the chip selects must be qualified by M/$\overline{IO}$ to determine which address space the devices are assigned to. This restriction on chip select decoding can be removed if the I/O and memory addresses in the system do not overlap and are properly decoded; all I/O is memory mapped; or $\overline{RD}$, $\overline{WR}$ and M/$\overline{IO}$ are decoded to provide separate memory and I/O read/write commands (Fig. 4C1). The 8288 bus controller in the maximum mode 8086 system generates separate I/O and memory commands in place of a M/$\overline{IO}$ signal. An I/O device is assigned to the I/O space or memory space (memory mapped I/O) by connection of either I/O or memory command lines to the command inputs of the device. To allow overlap of the memory and I/O address space, the device must not respond to chip select alone but must require a combination of chip select and a read or write command.

Linear select techniques (Fig. 4C2) for I/O devices can only be used with devices that either reside in the I/O address space or require more than one active chip select (at least one low active and one high active). Devices with a single chip select input cannot use linear select if they are memory mapped. This is due to the assignment of memory address space FFFFF0H-FFFFFFH to reset startup and memory space 00000H-003FFH to interrupt vectors.

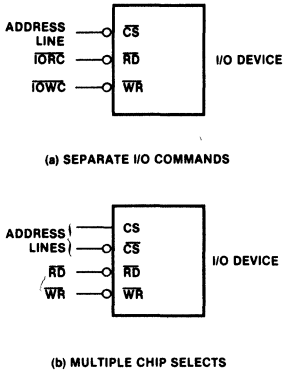


Figure 4C2. Linear Select for I/O

4D. Determining I/O Device Compatibility

This section presents a set of A.C. characteristics which represent the timing of the asynchronous bus interface of the 8086. The equations are expressed in terms of the CPU clock (when applicable) and are derived for minimum and maximum modes of the 8086. They represent the bus characteristics at the CPU.

The results can be used to determine I/O device requirements for operation on a single CPU local bus or buffered system bus. These values are not applicable to

a Multibus system bus interface. The requirements for a Multibus system bus are available in the Multibus interface specification.

A list of bus parameters, their definition and how they relate to the A.C. characteristics of Intel peripherals are given in Table 4D1. Cycle dependent values of the parameters are given in Table 4D2. For each equation, if more than one signal path is involved, the equation reflects the worst case path.

ex. TAVRL(address valid before read active) =
(1) Address from CPU to $\overline{RD}$ active
(or)
(2) ALE (to enable the address through the address latches) to $\overline{RD}$ active

The worst case delay path is (1).

For the maximum mode 8086 configurations, TAVWLA, TWLWHA and TWLCLA are relative to the advanced write signal while TAVWL, TWLWH and TWLCL are relative to the normal write signal.

TABLE 4D1. PARAMETERS FOR PERIPHERAL COMPATIBILITY

TAVRL — Address stable before RD leading edge	(TAR)
TRHAX — Address hold after RD trailing edge	(TRA)
TRLRH — Read pulse width	(TRR)
TRLDV — Read to data valid delay	(TRD)
TRHDZ — Read trailing edge to data floating	(TDF)
TAVDV — Address to valid data delay	(TAD)
TRLRL — Read cycle time	(TRCYC)
TAVWL — Address valid before write leading edge	(TAW)
TAVWLA — Address valid before advanced write	(TAW)
TWHAX — Address hold after write trailing edge	(TWA)
TWLWH — Write pulse width	(TWW)
TWLWHA — Advanced write pulse width	(TWW)
TDVWH — Data set up to write trailing edge	(TDW)
TWHDX — Data hold from write trailing edge	(TWD)
TWLCL — Write recovery time	(TRV)
TWLCLA — Advanced write recovery time	(TRV)
TSVRL — Chip select stable before RD leading edge	(TAR)
TRHSX — Chip select hold after RD trailing edge	(TRA)
TSLDV — Chip select to data valid delay	(TRD)
TSVWL — Chip select stable before WR leading edge	(TAW)
TWHSX — Chip select hold after WR trailing edge	(TWA)
TSVWLA — Chip select stable before advanced write	(TAW)

Symbols in parentheses are equivalent parameters specified for Intel peripherals

In the given list of equations, TWHDXB is the data hold time from the trailing edge of write for the minimum mode with a buffered data bus. For this equation, TCVCTX cannot be a minimum for data hold and a maximum for write inactive. The maximum difference is 50 ns giving the result TCLCH-50. If the reader wishes to verify the equations or derive others, refer to Section 3F for assistance with interpreting the 8086 bus timing diagrams.

Figure 4D1 shows four representative configurations and the compatible Intel peripherals (including wait states if required) for each configuration are given in Table 4D3. Configuration 1 and 2 are minimum mode demultiplexed bus 8086 systems without (1) and with (2) data bus transceivers. Configurations 3 and 4 are maximum mode systems with one (3) and two (4) levels of address and data buffering. The last configuration is characteristic of a multi-board system with bus buffers on each board. The 5 MHz parameter values for these configurations are given in Table 4D4 and demonstrate the relaxed device requirements for even a large complex configuration. The analysis assumes all components are exhibiting the specified worst case parameter values and are under the corresponding temperature, voltage and capacitive load conditions. If the capacitive loading on the 8282/83 or 8286/87 is less than the maximum, graphs of delay vs. capacitive loading in the respective data sheets should be used to determine the appropriate delay values.

TABLE 4D2. CYCLE DEPENDENT PARAMETER REQUIREMENTS FOR PERIPHERALS

(a) Minimum Mode

TAVRL = TCLCL + TCLRLmin – TCLAVmax = TCLCL – 100
TRHAX = TCLCL – TCLRHmax + TCLLHmin = TCLCL – 150
TRLRH = 2TCLCL – 60 = 2TCLCL – 60
TRLDV = 2TCLCL – TCLRLmax – TDVCLmin = 2TCLCL – 195
TRHDZ = TRHAVmin = 155 ns
TAVDV = 3TCLCL – TDVCLmin – TCLAVmax = 3TCLCL – 140
TRLRL = 4TCLCL = 4TCLCL
TAVWL = TCLCL + TCVCTVmin – TCLAVmax = TCLCL – 100
TWHAX = TCLCL + TCLLHmin – TCVCTXmax = TCLCL – 110
TWLWH = 2TCLCL – 40 = 2TCLCL – 40
TDVWH = 2TCLCL + TCVCTXmin – TCLDVmax = 2TCLCL – 100
TWHDX = TWHDZmin = 89
TWLCL = 4TCLCL = 4TCLCL
TWHDXB = TCLCHmin + (– TCVCTXmax + TCVCTXmin) = TCLCHmin – 50

Note Delays relative to chip select are a function of the chip select decode technique used and are equal to equivalent delay from address – chip select decode delay.

(b) Maximum Mode

TAVRL = TCLCL + TCLMLmin – TCLAVmax = TCLCL – 100
TRHAX = TCLCL – TCLMHmax + TCLLHmin = TCLCL – 40
TRLRH = 2TCLCL – TCLMLmax + TCLMHmin = 2TCLCL – 25
TRLDV = 2TCLCL – TCLMLmax – TDVCLmin = 2TCLCL – 65
TRHDZ = TRHAVmin = 155
TAVDV = 3TCLCL – TDVCLmin – TCLAVmax = 3TCLCL – 140
TRLRL = 4TCLCL = 4TCLCL
TAVWLA = TAVRL = TCLCL – 100
TAVWL = TAVRL + TCLCL = 2TCLCL – 100
TWHAX = TRHAX = TCLCL – 40
TWLWHA = TRLRH = 2TCLCL – 25
TWLWH = TRLRH – TCLCL = TCLCL – 25
TDVWH = 2TCLCL + TCLMHmin – TCLDVmax = 2TCLCL – 100
TWHDX = TCLCHmin – TCLMHmax + TCHDZmin = TCLCHmin – 30
TWLCL = 3TCLCL = 3TCLCL
TWLCLA = 4TCLCL = 4TCLCL

TABLE 4D3. COMPATIBLE PERIPHERALS (5 MHz 8086)

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
8251A	✓	1W	✓	✓
8253-5	✓	1W	✓	✓
8255A-5	✓	1W	✓	✓
8257-5	✓	1W	✓	✓
8259A	✓	✓	✓	✓
8271	✓	1W	✓	✓
8273	✓	1W	✓	✓
8275	✓	1W	✓	✓
8279-5	✓	1W	✓	✓
8041A*	✓	1W	✓	✓
8741A	✓	1W	✓	✓
8291	✓	✓	✓	✓

*Includes other Intel peripherals based on the 8041A (i.e., 8292, 8294, 8295)

✓ implies full operation with no wait states

W implies the number of wait states required

TABLE 4D4. PERIPHERAL REQUIREMENTS FOR FULL SPEED OPERATION WITH 5 MHz 8086

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
TAVRL	70	72	70	58
TRHAX	57	27	169	141
TRLRH	340	320	375	347
TRLDV	205	150	305	261
TRHDZ	155	158	382	360
TAVDV	430	400	400	372
TRLRL	800	770	800	772
TAVWL	70	72	270	258
TAVWLA	—	—	70	58
TWHAX	97	67	169	141
TWLWH	360	340	175	147
TWLWHA	—	—	375	347
TDVWH	300	339	270	258
TWHDX	88	15	95	13
TWLCL	800	772	600	572
TWLCLA	—	—	800	772
TSVRL	52	54	52	40
TRHSX	50	50	171	143
TSLDV	412	382	382	354
TSVWL	52	54	252	240
TWHSX	90	90	171	143
TSVWLA	—	—	52	40

— Not applicable.

Peripheral compatibility is determined from the equations given for the CPU by modifying them to account for additional delays from address latches and data transceivers in the configuration. Once the system configuration is selected, the system requirements can be determined at the peripheral interface and used to evaluate compatibility of the peripheral to the system. During this process, two areas must be considered. First, can the device operate at maximum bus bandwidth and if not, how many wait states are required. Second, are there any problems that cannot be resolved by wait states.

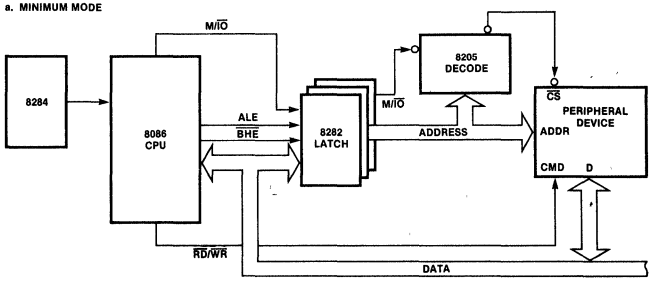
Examples of the first are TRLRH (read pulse width) and TRLDV (read access or $\overline{RD}$ active to output data valid). Consider address access time (valid address to valid data) for the maximum mode fully buffered configuration.

TAVDV = 3TCYC − 140 ns — address latch delay — address buffer delay — chip select decode delay — 2 transceiver delays

Assuming inverting latches, buffers and transceivers with 22 ns max delays (8283, 8287) and a bipolar PROM decode with 50 ns delay, the result is:

TAVDV = 322 ns @ 5 MHz

a. MINIMUM MODE



b MINIMUM MODE BUFFERED DATA AND COMMAND BUSSES

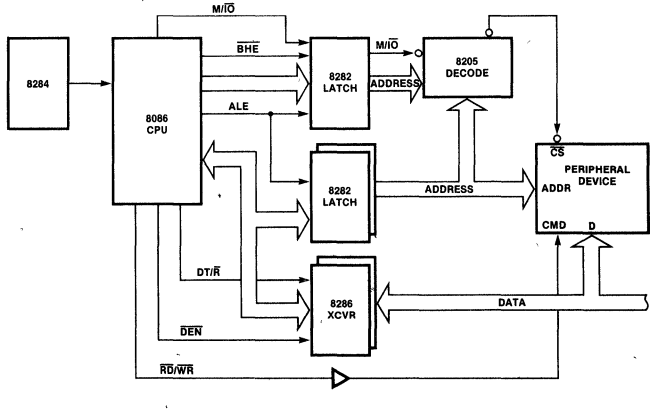
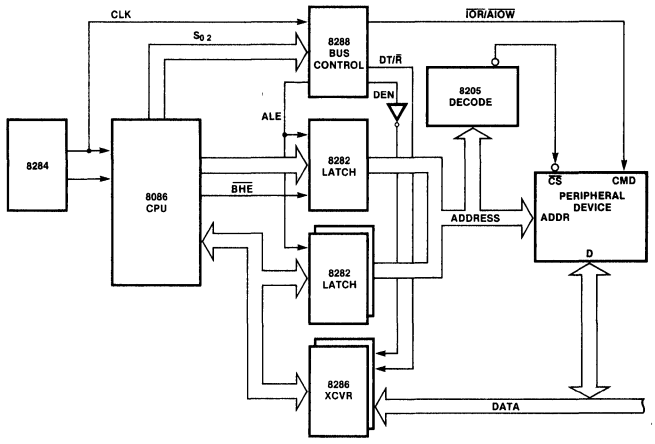


Figure 4D1. 8086 System Configurations

c MAXIMUM MODE BUFFERED DATA BUS



NOTE FOR OPTIMUM PERFORMANCE WITH INTEL PERIPHERALS, $\overline{AIOW}$ (ADVANCED WRITE) SHOULD BE USED

d MAXIMUM MODE DOUBLE BUFFERED SYSTEM

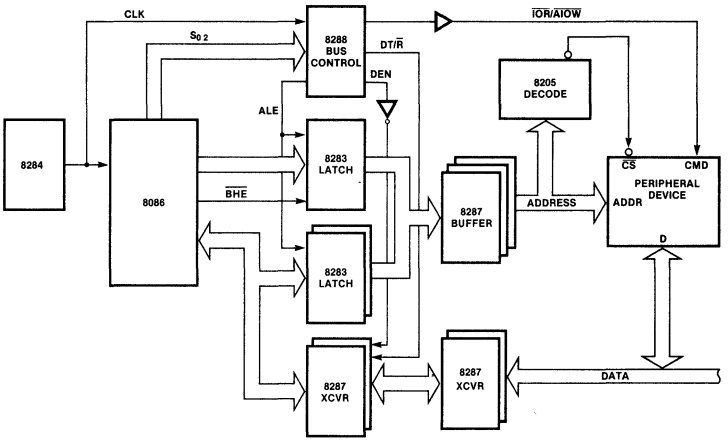


Figure 4D1. 8086 System Configurations (Con't)

The result gives the address to data valid delay required at the peripheral (in this configuration) to satisfy zero wait state CPU access time. If the maximum delay specified for the peripheral is less than the result, this parameter is compatible with zero wait state CPU operation. If not, wait states must be inserted until TAVDV + n * TCYC (n is the number of wait states) is greater than the peripherals maximum delay. If several parameters require wait states, either the largest number required should always be used or different transfer cycles can insert the maximum number required for that cycle.

The second area of concern includes TAVRL (address set up to read) and TWHDX (data hold after write). Incompatibilities in this area cannot be resolved by the insertion of wait states and may require either additional hardware, slowing down the CPU (if the parameter is related to the clock) or not using the device.

As an example consider address valid prior to advanced write low (TAVWLA) for the maximum mode fully buffered system.

TAVWLA = TCYC – 100 ns — address latch delay — address buffer delay — chip select decode delay + write buffer delay (minimum)

Assuming inverting latches and buffers with 22 ns delay (8283, 8287) and an 8205 address decoder with 18 ns delay

TAVWLA = 38 ns which is the time a 5 MHz 8086 system provides

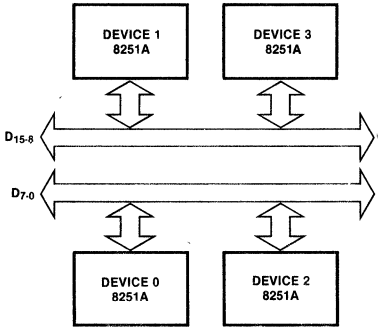
4E. I/O Examples

1. Consider an interrupt driven procedure for handling multiple communication lines. On receiving an interrupt from one of the lines, the invoked procedure polls the lines (reading the status of each) to determine which line to service. The procedure does not enable lines but simply services input and output requests until the associated output buffer is empty (for output requests) or until an input line is terminated (for the example only EOT is considered). On detection of the terminate condition, the routine will disable the line. It is assumed that other routines will fill a lines output buffer and enable the device to request output or empty the input buffer and enable the device to input additional characters.

The routine begins operation by loading CX with a count of the number of lines in the system and DX with the I/O address of the first line. The I/O addresses are assigned as shown in Figure 4E1 with 8251A's as the I/O devices. The status of each line is read to determine if it needs service. If yes, the appropriate routine is called to input or output a character. After servicing the line or if no service is needed, CX is decremented and DX is incremented to test the next line. After all lines have been tested and serviced, the routine terminates. If all interrupts from the lines are OR'd together, only one interrupt is used for all lines. If the interrupt is input to the CPU through an 8259A interrupt controller, the 8259A should be programmed in the level triggered mode to guarantee all line interrupts are serviced.

To service either an input or output request, the called routine transfers DX to BX, and shifts BX to form the offset for this device into the table of input or output buffers. The first entry in the buffer is an index to the next character position in the buffer and is loaded into the SI register. By specifying the base address of the table of buffers as a displacement into the data segment, the base + index + displacement addressing mode allows direct access to the appropriate memory location. 8086 code for part of this example is shown in Figure 4E2.

2. As a second example, consider using memory mapped I/O and the 8086 string primative instructions to perform block transfers between memory and I/O. By assigning a block of the memory address space (equivalent in size to the maximum block to be transferred to the I/O device) and decoding this address space to generate the I/O device's chip select, the block transfer capability is easily implemented. Figure 4E3 gives an interconnect for 16-bit I/O devices while Figure 4E4 incorporates the 16-bit bus to 8-bit bus multiplexing scheme to support 8-bit I/O devices. A code example to perform such a transfer is shown in Figure 4E5.



DEVICES ARE CONNECTED TO THE UPPER AND LOWER HALVES OF THE DATA BUS.

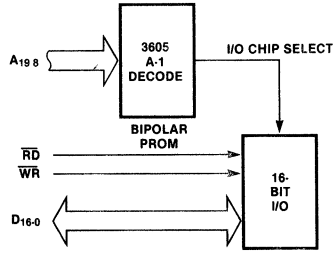
ADDRESS		
0	DEVICE 0	DATA
1	DEVICE 1	DATA
2	DEVICE 0	CONTROL/STATUS
3	DEVICE 1	CONTROL/STATUS
4	DEVICE 2	DATA
5	DEVICE 3	DATA
6	DEVICE 2	CONTROL/STATUS
7	DEVICE 3	CONTROL/STATUS
ETC.	"	"

Figure 4E1. Device Assignment

```
; THIS CODE DEMONSTRATES TESTING DEVICE
; STATUS FOR SERVICE, CONSTRUCTING THE
; APPROPRIATE LINE BUFFER ADDRESS FOR INPUT
, AND OUTPUT AND SERVICING AN INPUT
; REQUEST

                MASK EQU OFFFDH
CHECK_STATUS:   INPUT  AL, DX              ; GET 8251A STATUS.
                MOV    AH, AL
                TEST   AH, READ_OR_WRITE_STATUS
                JZ     NEXT_IO
                CALL   ADDRESS
                TEST   AH, READ STATUS
                JZ     WRITE_SERVICE
                CALL   READ
                TEST   AH, WRITE STATUS
                JZ     NEXT_IO
WRITE_SERVICE.  CALL   WRITE
NEXT_IO.        DEC    CX                  ; TEST IF DONE.
                JNC    EXIT                ; YES, RESTORE & RETURN.
                AND    DX, MASK            ; REMOVE A1 AND
                ADD    DX, 3               ; INCREMENT ADDRESS.
                OR     DX, 2               ; SELECT STATUS FOR
                JMP    CHECK_STATUS        ; NEXT INPUT.
ADDRESS·        AND    DX, MASK            ; SELECT DATA.
                MOV    BH, DL              ; CONSTRUCT BUFFER
                INC    BH                  ; DISPLACEMENT FOR
                SHR    BH                  ; THIS DEVICE.
                XOR    BL, BL              ; BX IS THE DISPLACEMENT.
                RET
READ·           INPUT AL, DX               ; READ CHARACTER.
                MOV SI, READ_BUFFERS [BX]  ; GET CHARACTER POINTER.
                MOV READ_BUFFERS [BX + SI], AL ; STORE CHARACTER.
                INC READ_BUFFERS [BX]      ; INCR CHARACTER POINTER.
                CMP AL, EOT                ; END OF TRANSMISSION?
                JNZ CONT_READ
                CALL DISABLE READ          ; YES, DISABLE RECEIVER.
                CONT_READ:  RET            ; SEND MESSAGE THAT INPUT
                                           ; IS READY.
```

Figure 4E2.



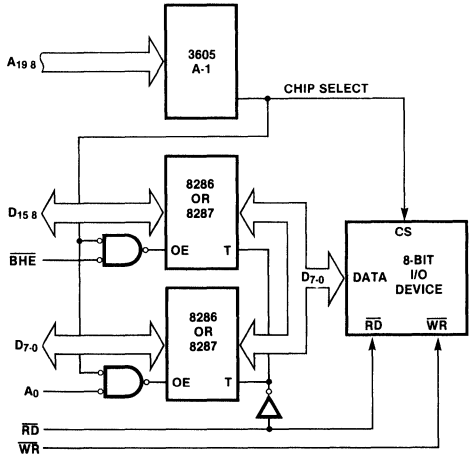
TRANSFER 256 BYTE BLOCKS TO THE I/O DEVICE

THE ADDRESS SPACE ASSIGNED TO THE I/O DEVICE IS

	A19		A8 | A7	A0
FROM	← BASE	ADDRESS →	← 0's	→
THRU	← BASE	ADDRESS →	← 1's	→

MEMORY DATA NEED NOT BE ALIGNED TO EVEN ADDRESS BOUNDARIES
I/O TRANSFERS MUST BE WORD TRANSFERS TO EVEN ADDRESS BOUNDARIES

Figure 4E3. Block Transfer to 16-Bit I/O Using 8086 String Primatives



ADDRESS ASSIGNMENT SAME AS PREVIOUS EXAMPLE. 16-BIT BUS IS MULTIPLEXED ONTO AN 8-BIT PERIPHERAL BUS.

Figure 4E4. Block Transfer to 8-Bit I/O Using 8086 String Primatives

```
                ; DEFINE THE I/O ADDRESS SPACE
                  I/O SEGMENT
                  ORG BLOCK_ADDRESS
I/O_BLOCK:        DW 128 DUP (?)
                  I/O ENDS

                ; ASSUME THE DATA IS FROM THE CURRENT
                ; DATA SEGMENT
                  CLD                     ; DF = FORWARD
                  LES DI, I/O_BLOCK_ADDRESS ; I/O BLOCK ADDRESS
                                          ; CONTAINS THE ADDRESS
                                          ; OF I/O BLOCK
                  MOV CX, BLOCK_LENGTH
                  MOV SI, SOURCE_ADDRESS
                  MOVS I/O BLOCK          ; PERFORM WORD TRANSFERS

                ; END CODE EXAMPLE
```

NOTE THE CODE IS CAPABLE OF PERFORMING BYTE TRANSFERS BY CHANGING THE I/O BLOCK DEFINITION FROM 128 WORD TO 256 BYTES

Figure 4E5. Code for Block Transfers

5. INTERFACING WITH MEMORIES

Figure 5.1 is a general block diagram of an 8086 memory. The basic characteristics of the diagram are the partitioning of the 16-bit word memory into high and low 8-bit banks on the upper and lower halves of the data bus and inclusion of $\overline{BHE}$ and A0 in the selection of the banks. Specific implementations depend on the type of memory and the system configuration.

5A. ROM and EPROM

The easiest devices to interface to the system are ROM and EPROM. Their byte format provides a simple bus interface and since they are read only devices, A0 and $\overline{BHE}$ need not be included in their chip enable/select decoding (chip enable is similar to chip select but additionally determines if the device is in active or standby power mode). The address lines connected to the devices start with A1 and continue up to the maximum number the device can accept, leaving the remaining address lines for chip enable/select decoding. To connect the devices directly to the multiplexed bus, they must have output enables. The output enable is also necessary to avoid bus contention in other configurations. Figure 5A1 shows the bus connections for ROM and EPROM memories. No special decode techniques are required for generating chip enables/selects. Each valid decode selects one device on the upper and lower halves of bus to allow byte and word access. Byte access is achieved by reading the full word onto the bus with the 8086 only accepting the desired byte. For the minimum mode 8086, if $\overline{RD}$, $\overline{WR}$ and M/$\overline{IO}$ are not decoded to form separate commands for memory and I/O, and the I/O space overlaps the memory space assigned to the EPROM/ROM then M/$\overline{IO}$ (high active) must be a condition of chip enable/select decode. The output enable is controlled by the system memory read signal.

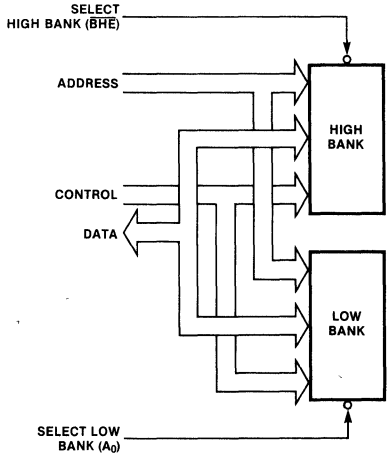
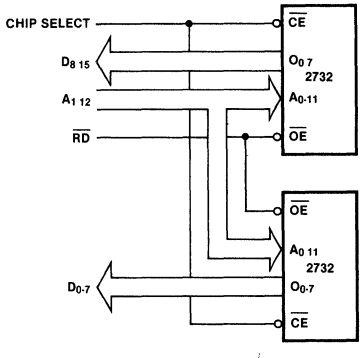


Figure 5.1. 8086 Memory Array



NOTE A_0 AND $\overline{BHE}$ ARE NOT USED.

Figure 5A1. EPROM/ROM Bus Interface

Static ROM's and EPROM's have only four parameters to evaluate when determining their compatibility to the system. The parameters, equations and evaluation techniques given in the I/O section are also applicable to these devices. The relationship of parameters is given in Table 5A1. TACC and TCE are related to the same equation and differ only by the delay associated with the chip enable/select decoder. As an example, consider a 2716 EPROM memory residing on the multiplexed bus of a minimum mode configuration:

TACC = 3TCLCL – 140 – address buffer delay = 430 ns
(8282 = 30 ns max delay)

TCE = TACC – decoder delay = 412 ns
(8205 decoder delay = 18 ns)

TOE = 2TCLCL – 195 = 205 ns

TDF = = 155 ns

TABLE 5A1. EPROM/ROM PARAMETERS

TOE — Output Enable to Valid Data ≡ TRLDV
TACC — Address to Valid Data ≡ TAVDV
TCE — Chip Enable to Valid Data ≡ TSLDV
TDF — Output Enable High to Output Float ≡ TRHDZ

The results are the times the system configuration requires of the component for full speed compatibility with the system. Comparing these times with 2716 parameter limits indicates the 2716-2 will work with no wait states while the 2716 will require one wait state. Table 5A2 demonstrates EPROM/ROM compatibility for the configurations presented in the I/O section. Before designing a ROM or EPROM memory system, refer to AP-30 for additional information on design techniques that give the system an upgrade path from 16K to 32K and 64K devices.

TABLE 5A2. COMPATIBLE EPROM/ROM (5 MHz 8086)

	Configuration			
	Minimum Mode		Maximum Mode	
	Unbuffered	Buffered	Buffered	Fully Buffered
2716-1	✓	✓	✓	✓
2716-2	✓	1W	1W	1W
2732	1W	1W	1W	1W
2332	✓	✓	✓	✓
2364	✓	✓	✓	✓

5B. Static RAM

Interfacing static RAM to the system introduces several new requirements to the memory design. A0 and $\overline{BHE}$ must be included in the chip select/chip enable decoding of the devices and write timing must be considered in the compatibility analysis.

For each device, the data bus connections must be restricted to either the upper or lower half of the data bus. Devices like the 2114 or 2142 must not straddle the upper and lower halves of the data bus (Fig. 5B1). To allow selecting either the upper byte, lower byte or full 16-bit word for a write operation, $\overline{BHE}$ must be a condition of decode for selecting the upper byte and A0 must be a condition of decode for selecting the lower byte. Figure 5B2 gives several selection techniques for devices with single chip selects and no output enables (2114, 2141, 2147). Figure 5B3 gives selection techniques for devices with chip selects and output enables.

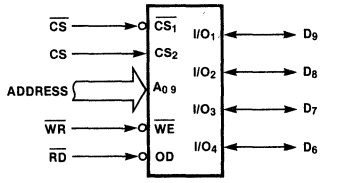


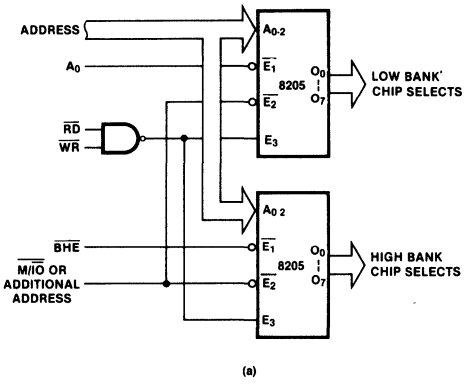
Figure 5B1. Incorrect Connection of 2142 Across Byte Boundaries

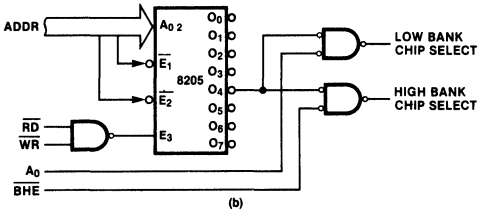
The first group requires inclusion of A0 and $\overline{BHE}$ to decode or enable the chip selects. Since these memories do not have output enables, read and write are used as enabies for chip select generation to prevent bus contention. If read and write are not used to enable the chip selects, devices with common input/output pins (like the 2114) will be subjected to severe bus contention between chip select and write active. For devices with separate input/output lines (like 2141, 2147), the outputs can be externally buffered with the buffer enable controlled by read. This solution will only allow bus contention between memory devices in the array during chip select transition periods. These techniques are considered in more detail in Section 2C.

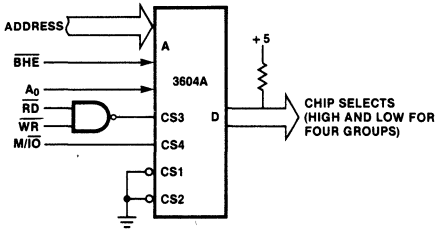
For devices with output enables (2142), write may be gated with $\overline{BHE}$ and A0 to provide upper and lower bank write strobes. This simplifies chip select decoding by eliminating $\overline{BHE}$ and A0 as a condition of decode. Although both devices are selected during a byte write operation, only one will receive a write strobe. No bus contention will exist during the write since a read command must be issued to enable the memory output drivers.

If multiple chip selects are available at the device, $\overline{BHE}$ and A0 may directly control device selection. This allows normal chip select decoding of the address space and direct connection of the read and write commands to the devices. Alternately, the multiple chip select inputs of the device could directly decode the address space (linear select) and be combined with the separate write strobe technique to minimize the control circuitry needed to generate chip selects.

As with the EPROM's and ROM's, if separate commands are not provided for memory and I/O in the minimum mode 8086 and the address spaces overlap, M/$\overline{IO}$ (high active) must be a condition of chip select decode. Also, the address lines connected to the memory devices must start with A1 rather than A0.







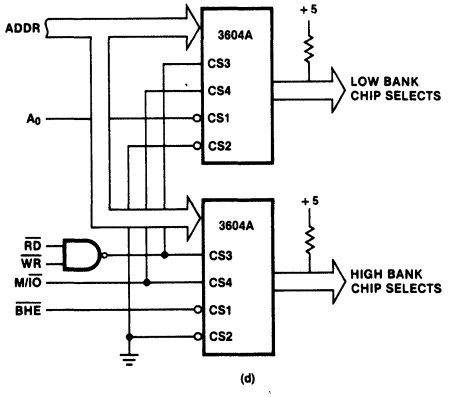
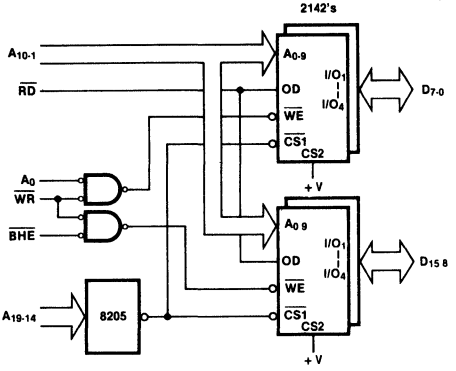
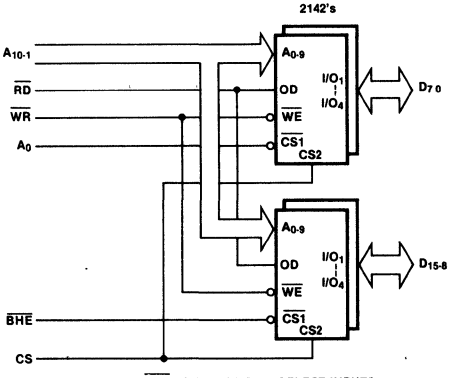


Figure 5B2. Generating Chip Selects for Devices without Output Enables





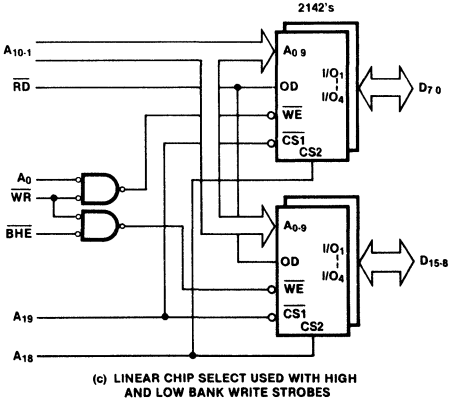


Figure 5B3. Chip Selection for Devices with Output Enables

For analysis of RAM compatibility, the write timing parameters listed in Table 5B1 may also need to be considered (depending on the RAM device being considered). The CPU clock relative timing is given in Table 5B2. The equations specify the device requirements at the CPU and provide a base for determining device requirements in other configurations. As an example consider the write timing requirements of a 2142 in a maximum mode buffered 8086 system (Figure 5B4). The 2142 write parameters that must be analyzed are TWA advanced write pulse width, TWR write release time, TDWA data to write time overlap and TDH data hold from write time.

TWA = 2TCLCL – TCLMLmax + TCLMHmin = 375 ns.
TWR = 2TCLCL – TCLMHmax + TCLLHmin + TSHOVmin = 170 ns.
TDWA = 2TCLCL – TCLDVmax + TCLMHmin – TIVOVmax = 265 ns.
TDH = TCLCH – TCLMHmax + TCHDXmin + TIVOVmin = 95 ns.

TABLE 5B1. TYPICAL WRITE TIMING PARAMETERS

TW — Write Pulse Width
TWR — Write Release (Address Hold From End of Write)
TDW — Data and Write Pulse Overlap
TDH — Data Hold From End of Write
TAW — Address Valid to End of Write
TCW — Chip Select to End of Write
TASW — Address Valid to Beginning of Write

TABLE 5B2. CYCLE DEPENDENT WRITE PARAMETERS FOR RAM MEMORIES

(a) Minimum Mode
TW = TWLWH = 2TCLCL – 60 = 340 ns
TWR = TCLCL – TCVCTXmax + TCLLHmin = 90 ns
TDW = 2TCLCL – TCLDVmax + TCVCTXmin = 300 ns
TDH = TWHDX = 88 ns
TAW = 3TCLCL – TCLAVmax + TCVCTXmin = 500 ns
TCW = TAW – Chip Select Decode
TASW = TCLCL – TCLAVmax + TCVCTXmin = 100 ns
(b) Maximum Mode
TW = TCLCL – TCLMLmax + TCLMHmin = 175 ns
TWR = TCLCL – TCLMHmax + TCLLHmin = 165 ns
TDW = TW = 175 ns
TDH = TCLCHmin – TCLMHmax + TCHDXmin = 93 ns
TAW = 3TCLCL – TCLAVmax + TCLMHmin = 500 ns
TCW = TAW – Chip Select Decode
TASW = 2TCLCL – TCLAVmax + TCLMLmin = 300 ns
TWA* = TW + TCLCL = 375 ns
TDWA* = 2TCLCL – TCLDVmax + TCLMHmin = 300 ns
TASWA* = TASW – TCLCL = 100 ns
*Relative to Advanced Write.

Comparing these results with the 2142 family indicates the standard 2142 write timing is fully compatible with this 8086 configuration. Read timing analysis is also necessary to completely determine compatibility of the devices.

5C. Dynamic RAM

Dynamic RAM is perhaps the most complex device to design into a system. To relieve the engineer of most of this burden, Intel provides the 8202 dynamic RAM controller as part of the 8086 family of peripheral devices. This section will discuss using the 8202 with the 8086 to build a dynamic memory system for an 8086 system. For additional information on the 8202, refer to the 8202 data sheet (9800873) and application note AP-45 Using the 8202 Dynamic RAM Controller (9800809A).

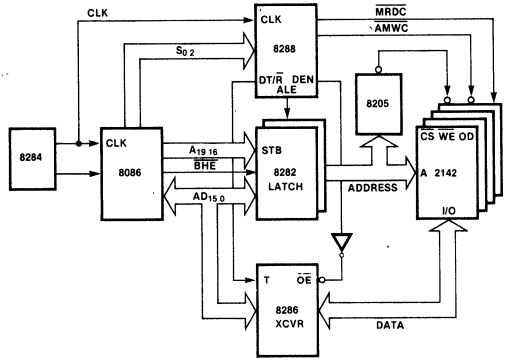


Figure 5B4. Sample Configuration for Compatibility Analysis Example

5.C.1 Standard 8086-8202 Interconnect

Figure 5.C.1.1 shows a standard interconnection for an 8202 into an 8086 system. The configuration accommodates 64K words (128K bytes) of dynamic RAM addressable as words or bytes. To access the RAM, the 8086 initiates a bus cycle with an address that selects the 8202 (via $\overline{PCS}$) and the appropriate transfer command ($\overline{MRDC}$ or $\overline{MWTC}$). If the 8202 is not performing a refresh cycle, the access starts immediately, otherwise, the 8086 must wait for completion of the refresh. $\overline{XACK}$ from the 8202 is connected to the 8284 RDY input to force the CPU to wait until the RAM cycle is completed before the CPU can terminate the bus cycle. This effectively synchronizes the asynchronous events of refresh and CPU bus cycles. The normal write command ($\overline{MWTC}$) is used rather than the advanced command ($\overline{AMWC}$) to guarantee the data is valid at the dynamic RAMS before the write command is issued. The gating of $\overline{WE}$ with A0 and $\overline{BHE}$ provides selective write strobes to the upper and lower banks of memory to allow byte and word write operations. The logic which generates the strobe for the data latches allows read data to propagate to the system as soon as the data is available and latches the data on the trailing edge of $\overline{CAS}$.

DETAILED TIMING

Read Cycle

For no wait state operation, the 8086 requires data to be valid from $\overline{MRDC}$ in:

2TCLCL – TCLML – TDVCL – buffer delays = 291 ns.

Since the 8202 is $\overline{CAS}$ access limited, we need only examine $\overline{CAS}$ access time. The 8202/2118 guarantees data valid from 8202 $\overline{RD}$ low to be:

(tph + 3tp + 100 ns) 8202 TCC delay + TCAC for the 2118

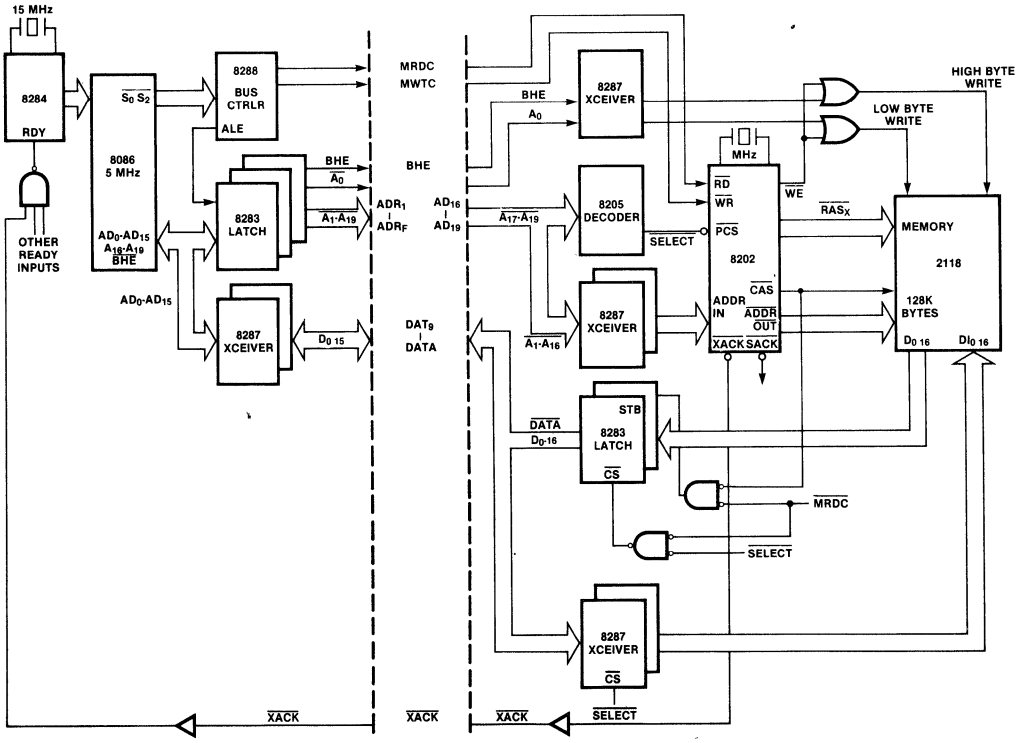


Figure 5C1.1. 5 MHz 8086/8202/128K Byte System — Double Data, Control and Address Buffering (Note: Bus driver on 8202 is not needed if less than 64K bytes are used)

For a 25 MHz 8202 and 2118-3, we get 297 ns which is insufficient for no wait state operation. If only 64K bytes are accessed, the 8202 requires only (tph + 3tp + 85 ns) giving 282 ns access and no wait states required. Refer to Figure 5.C.1.2 and 5C.1.3 for timing information on the 8202 and 2118.

Write Cycle

An important consideration for dynamic RAM write cycles is to guarantee data to the RAM is valid when both $\overline{CAS}$ and $\overline{WE}$ are active. For the 2118, if $\overline{WE}$ is valid prior to $\overline{CAS}$, the data setup is to $\overline{CAS}$ and if $\overline{CAS}$ is valid before $\overline{WE}$ (as would occur during a read modify write cycle) the data setup time is to $\overline{WE}$. For the 8202, the $\overline{WR}$ to $\overline{CAS}$ delay is analyzed to determine the data setup time to $\overline{CAS}$ inherently provided by the 8202 command to $\overline{RAS}/\overline{CAS}$ timing. The minimum delay from $\overline{WR}$ to $\overline{CAS}$ is:

TCCmin = tph + 2tp + 25 = 127 ns @ 25 MHz

Subtracting buffer delays and data setup at the 2118, we have 83 ns to generate valid data after the write command is issued by the CPU (in this case the 8288). Since the 8086 will not guarantee valid data until TCLAVmax – TCLMLmin = 100 ns from the advanced write signal, the normal write signal is used. The normal write $\overline{MWTC}$ guarantees data is valid 100 ns before it is active. The worst case write pulse width is approximately 175 ns which is sufficient for all 2118's.

Synchronization

To force the 8086 to wait during refresh the $\overline{XACK}$ or $\overline{SACK}$ lines must be returned to the 8284 ready input. The maximum delay from $\overline{RD}$ to $\overline{SACK}$ (if the 8202 is not performing refresh) is TAC = tp + 40 = 80 ns. To prevent a wait state at the 8086, RDY must be valid at the 8284 TCLCHmin – TCLMLmax – TR1VCLmax = 48 ns after the command is active. This implies that under worst case conditions, one wait state will be inserted for every read cycle. Since $\overline{MWTC}$ does not occur until one clock later, two wait states may be inserted for writes.

The $\overline{XACK}$ from command delay will assert RDY TCC + TCX = (tph + 3tp + 100) + (5tp + 20) = 460 ns after the command. This will typically insert one or two wait states.

Unless 2118-3's are used in 64K byte or less memories, $\overline{SACK}$ must not be used since it does not guarantee a wait state. From the previous access time analysis we saw that other configurations required a wait state.

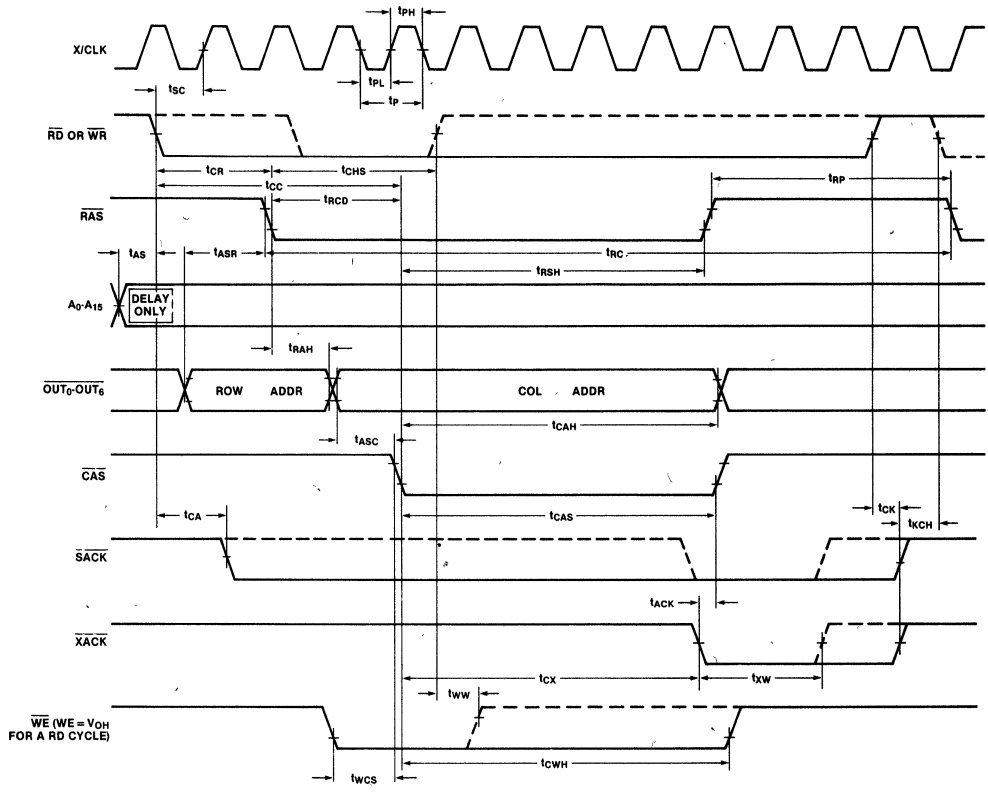


Figure 5C1.2. 8202 Timing Information

A.C. CHARACTERISTICS

$T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$

Loading:
64 Devices

$\overline{SACK}$, $\overline{XACK}$	CL = 30 pF
$\overline{OUT_0} - \overline{OUT_6}$	CL = 320 pF
$\overline{RAS_1} - \overline{RAS_4}$	CL = 230 pF
$\overline{WE}$	CL = 450 pF
$\overline{CAS}$	CL = 640 pF

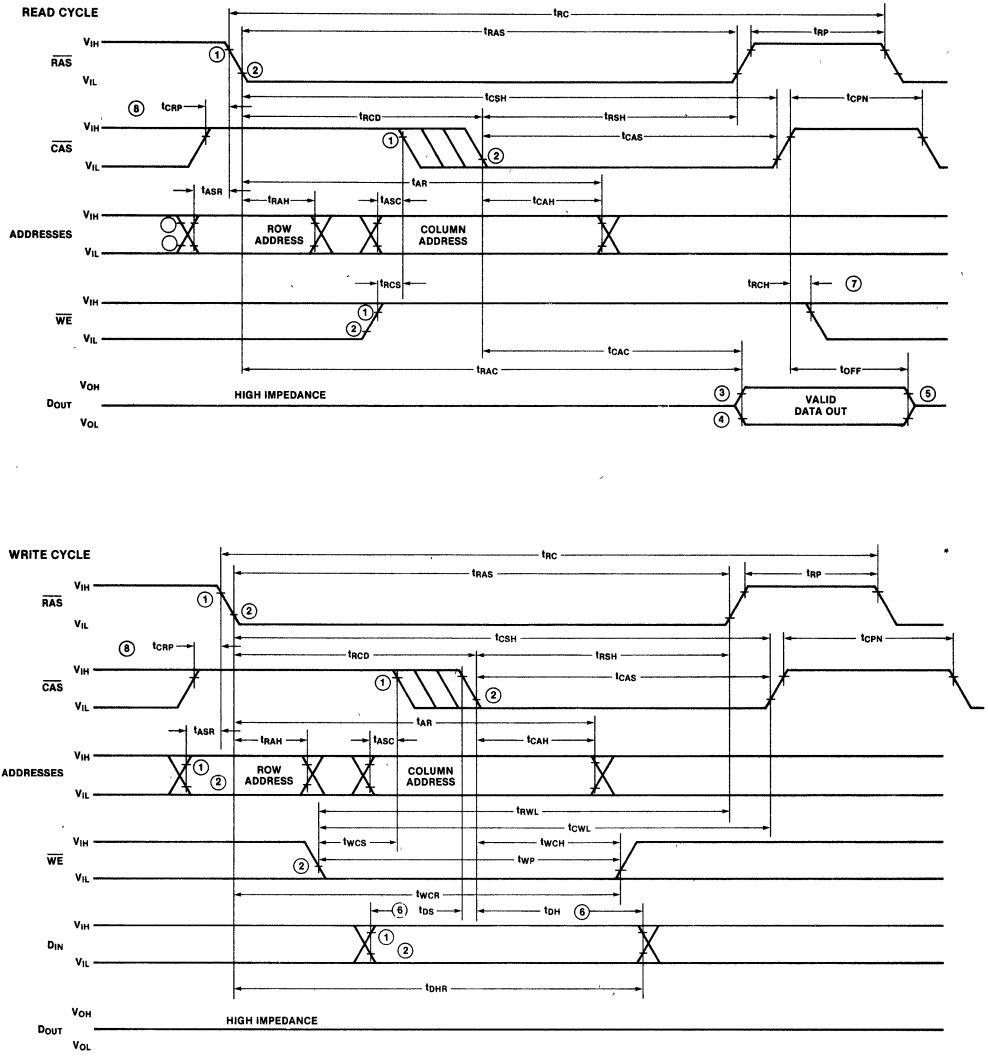
Measurements made with respect to $RAS_1 - RAS_4$, CAS, WE, $OUT_0 - OUT_6$ are at 2.4V and 0.8V. All other pins are measured at 1.5V.

Symbol	Parameter	Min	Max	Units
t_P	Clock (Internal/External) Period (See Note 1)	40	54	ns
t_{RC}	Memory Cycle Time	$10\ t_P - 30$	$12\ t_P$	ns
t_{RAH}	Row Address Hold Time	$t_P - 10$		ns
t_{ASR}	Row Address Setup Time	t_{PH}		ns
t_{CAH}	Column Address Hold Time	$5\ t_P$		ns
t_{ASC}	Column Address Setup Time	$t_P - 35$		ns
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	$2\ t_P - 10$	$2\ t_P + 45$	ns
t_{WCS}	$\overline{WE}$ Setup to $\overline{CAS}$	$t_P - 40$		ns
t_{RSH}	$\overline{RAS}$ Hold Time	$5\ t_P - 30$		ns
t_{CAS}	$\overline{CAS}$ Pulse Width	$5\ t_P - 30$		ns
t_{RP}	$\overline{RAS}$ Precharge Time (See Note 2)	$4\ t_P - 30$		ns
t_{WCH}	$\overline{WE}$ Hold Time to $\overline{CAS}$	$5\ t_P - 35$		ns
t_{REF}	Internally Generated Refresh to Refresh Time 64 Cycle 128 Cycle	 $548\ t_P$ $264\ t_P$	 $576\ t_P$ $288\ t_P$	 ns ns
t_{CR}	$\overline{RD}$, $\overline{WR}$ to $\overline{RAS}$ Delay	$t_{PH} + 30$	$t_{PH} + t_P + 75$	ns
t_{CC}	$\overline{RD}$, $\overline{WR}$ to $\overline{CAS}$ Delay	$t_{PH} + 2\ t_P + 25$	$t_{PH} + 3\ t_P + 100$	ns
t_{RFR}	REFRQ to $\overline{RAS}$ Delay	$1.5\ t_P + 30$	$2.5\ t_P + 100$	ns
t_{AS}	A_0—A_{15} to $\overline{RD}$, $\overline{WR}$ Setup Time (See Note 4)	0		ns
t_{CA}	$\overline{RD}$, $\overline{WR}$ to $\overline{SACK}$ Leading Edge		$t_P + 40$	ns
t_{CK}	$\overline{RD}$, $\overline{WR}$ to $\overline{XACK}$, $\overline{SACK}$ Trailing Edge Delay		30	ns
t_{KCH}	$\overline{RD}$, $\overline{WR}$ Inactive Hold to $\overline{SACK}$ Trailing Edge	10		ns
t_{SC}	$\overline{RD}$, $\overline{WR}$, $\overline{PCS}$ to X/CLK Setup Time (See Note 3)	15		ns
t_{CX}	$\overline{CAS}$ to $\overline{XACK}$ Time	$5\ t_P - 40$	$5\ t_P + 20$	ns
t_{ACK}	$\overline{XACK}$ Leading Edge to $\overline{CAS}$ Trailing Edge Time	10		ns
t_{XW}	$\overline{XACK}$ Pulse Width	$2\ t_P - 25$		ns
t_{LL}	REFRQ Pulse Width	20		ns
t_{CHS}	$\overline{RD}$, $\overline{WR}$, $\overline{PCS}$ Active Hold to $\overline{RAS}$	0		ns
t_{WW}	$\overline{WR}$ to $\overline{WE}$ Propagation Delay	8	50	ns
t_{AL}	S_1 to ALE Setup Time	40		ns
t_{LA}	S_1 to ALE Hold Time	$2\ t_P + 40$		ns
t_{PL}	External Clock Low Time	15		ns
t_{PH}	External Clock High Time	22		ns
t_{PH}	External Clock High Time for $V_{CC} = 5V \pm 5\%$	18		ns

Notes:

1. t_P minimum determines maximum oscillator frequency.
   t_P maximum determines minimum frequency to maintain 2 ms refresh rate and t_{RP} minimum.
2. To achieve the minimum time between the $\overline{RAS}$ of a memory cycle and the $\overline{RAS}$ of a refresh cycle, such as a transparent refresh, REFRQ should be pulsed in the previous memory cycle.
3. t_{SC} is not required for proper operation which is in agreement with the other specs, but can be used to synchronize external signals with X/CLK if it is desired.
4. If t_{AS} is less than 0 then the only impact is that t_{ASR} decreases by a corresponding amount.

Figure 5C1.2. 8202 Timing Information (Con't)



NOTES 1,2 $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS

3,4 $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}

5. t_{OFF} IS MEASURED TO $|I_{OUT}| < |I_{OL}|$

6. t_{DS} AND t_{DH} ARE REFERENCED TO $\overline{CAS}$ OR $\overline{WE}$, WHICHEVER OCCURS LAST

‡ t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST

8 t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAS}$/$\overline{CAS}$ CYCLES PRECEDED BY A $\overline{CAS}$ ONLY CYCLE (i e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).

Figure 5C1.3. 2118 Family Timing

A.C. CHARACTERISTICS[1,2,3]

$T_A = 0°C$ to $70°C$, $V_{DD} = 5V \pm 10\%$, $V_{SS} = 0V$, unless otherwise noted.

READ, WRITE, READ-MODIFY-WRITE AND REFRESH CYCLES

Symbol	Parameter	2118-3		2118-4		2118-7		Unit	Notes
		Min.	Max.	Min.	Max.	Min.	Max.		
t_{RAC}	Access Time From $\overline{RAS}$		100		120		150	ns	4,5
t_{CAC}	Access Time from $\overline{CAS}$		55		65		80	ns	4,5,6
t_{REF}	Time Between Refresh		2		2		2	ms	
t_{RP}	$\overline{RAS}$ Precharge Time	110		120		135		ns	
t_{CPN}	$\overline{CAS}$ Precharge Time (non-page cycles)	50		55		70		ns	
t_{CRP}	$\overline{CAS}$ to $\overline{RAS}$ Precharge Time	0		0		0		ns	
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	25	45	25	55	25	70	ns	7
t_{RSH}	$\overline{RAS}$ Hold Time	70		85		105		ns	
t_{CSH}	$\overline{CAS}$ Hold Time	100		120		165		ns	
t_{ASR}	Row Address Set-Up Time	0		0		0		ns	
t_{RAH}	Row Address Hold Time	15		15		15		ns	
t_{ASC}	Column Address Set-Up Time	0		0		0		ns	
t_{CAH}	Column Address Hold Time	15		15		20		ns	
t_{AR}	Column Address Hold Time to $\overline{RAS}$	60		70		90		ns	
t_T	Transition Time (Rise and Fall)	3	50	3	50	3	50	ns	8
t_{OFF}	Output Buffer Turn Off Delay	0	45	0	50	0	60	ns	
READ AND REFRESH CYCLES									
T_{RC}	Random Read Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{RCS}	Read Command Set-Up Time	0		0		0		ns	
t_{RCH}	Read Command Hold Time	0		0		0		ns	
WRITE CYCLE									
t_{RC}	Random Write Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{WCS}	Write Command Set-Up Time	0		0		0		ns	9
t_{WCH}	Write Command Hold Time	25		30		45		ns	
t_{WCR}	Write Command Hold Time, to $\overline{RAS}$	70		85		115		ns	
t_{WP}	Write Command Pulse Width	25		30		50		ns	
t_{RWL}	Write Command to $\overline{RAS}$ Lead Time	60		65		110		ns	
t_{CWL}	Write Command to $\overline{CAS}$ Lead Time	45		50		100		ns	
t_{DS}	Data-In Set-Up Time	0		0		0		ns	
t_{DH}	Data-In Hold Time	25		30		45		ns	
t_{DHR}	Data-In Hold Time, to $\overline{RAS}$	70		85		115		ns	
READ-MODIFY-WRITE CYCLE									
t_{RWC}	Read-Modify-Write Cycle Time	285		320		410		ns	
t_{RRW}	RMW Cycle $\overline{RAS}$ Pulse Width	165	10000	190	10000	265	10000	ns	
t_{CRW}	RMW Cycle $\overline{CAS}$ Pulse Width	105	10000	120	10000	185	10000	ns	
t_{RWD}	$\overline{RAS}$ to $\overline{WE}$ Delay	100		120		150		ns	9
t_{CWD}	$\overline{CAS}$ to $\overline{WE}$ Delay	55		65		80		ns	9

NOTES
1 All voltages referenced to V_{SS}
2 Eight cycles are required after power-up or prolonged periods (greater than 2 ms) of $\overline{RAS}$ inactivity before proper device operation is achieved Any 8 cycles which perform refresh are adequate for this purpose
3 A C Characteristics assume $t_T = 5$ ns
4 Assume that $t_{RCD} \leq t_{RCD}$ (max) If t_{RCD} is greater than t_{RCD} (max) then t_{RAC} will increase by the amount that t_{RCD} exceeds t_{RCD} (max)
5 Load = 2 TTL loads and 100 pF
6 Assumes $t_{RCD} \geq t_{RCD}$ (max)
7 t_{RCD} (max) is specified as a reference point only, if t_{RCD} is less than t_{RCD} (max) access time is t_{RAC}, if t_{RCD} is greater than t_{RCD} (max) access time is $t_{RCD} + t_{CAC}$
8 t_T is measured between V_{IH} (min) and V_{IL} (max)
9 t_{WCS}, t_{CWD} and t_{RWD} are specified as reference points only If $t_{WCS} \geq t_{WCS}$ (min) the cycle is an early write cycle and the data out pin will remain high impedance throughout the entire cycle If $t_{CWD} \geq t_{CWD}$ (min.) and $t_{RWD} \geq t_{RWD}$ (min.), the cycle is a read-modify-write cycle and the data out will contain the data read from the selected address If neither of the above conditions is satisfied, the condition of the data out is indeterminate

Figure 5C1.3. 2118 Family Timing (Con't)

5.C.2 Enhanced Operation

Two problems are evident from the previous investigation:

1) $\overline{SACK}$ timing from command will not allow reliable operation while $\overline{XACK}$ is not active early enough to prevent wait states.

2) The normal write command required to guarantee data setup is not enabled until the CPU has sampled READY thereby forcing multiple wait states during write operations.

The first problem could be resolved if an early command could be generated that would guarantee $\overline{SACK}$ was valid when READY was sampled and $\overline{SACK}$ to data valid satisfied the CPU requirements. Figure 5.C.2.1 is a circuit which provides an early read command derived from the maximum mode status. The early command is enabled from the trailing edge of ALE and disabled on the trailing edge of the normal command. The command provides an additional TCHCLmin – TCHLLmax + TCLMLmax – circuit delays = 53 ns of access time and time to generate RDY from the early command. If we go back to our previous equations, early command to valid data at the CPU is now:

TCHCLmin – TCHLLmax + 2TCLCL – TDVCLmax – buffer and circuit delays = 333 ns

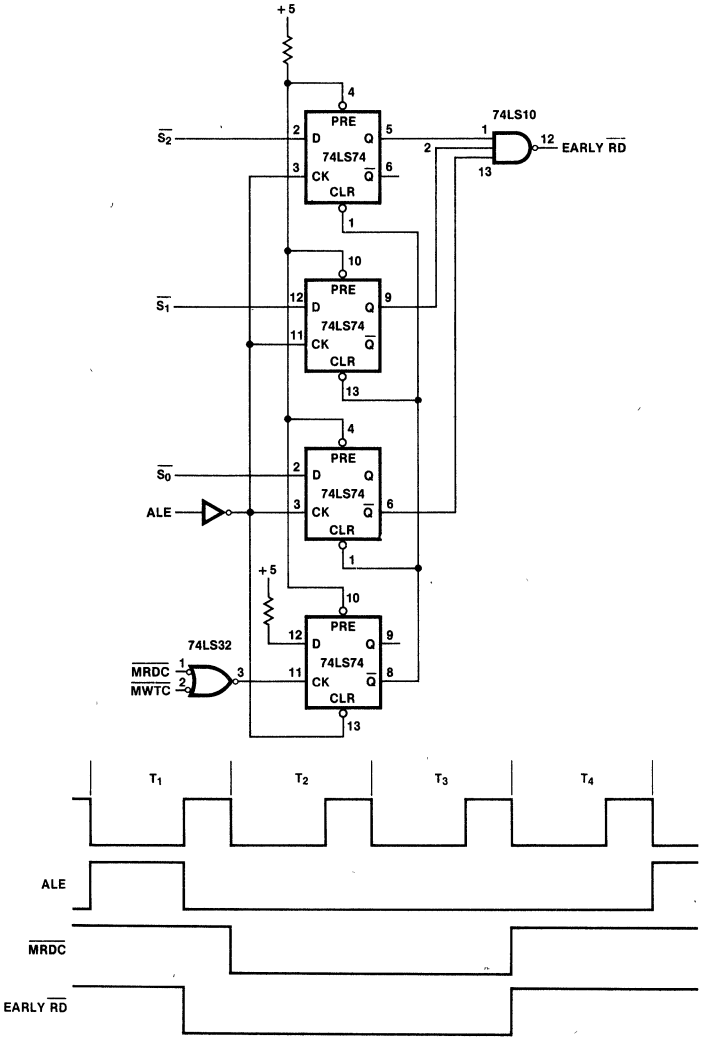


Figure 5C2.1. Early Read and Write Command Generation

We can now use the slowest 2118 which gives 8202 and 2118 access of 320 ns. Early command to RDY timing is TCLCL – TCHLLmax – circuit delays – TR1VCLmax = 115 ns and provides 35 ns of margin beyond the 8202 command to $\overline{SACK}$ delay.

The write timing of the 8202 and write data valid timing of the 8086 do not allow use of an early write command. However, if the 8202 clock is reduced from 25 MHz to 20 MHz and $\overline{WE}$ to the RAM's is gated with $\overline{CAS}$, the advanced write command ($\overline{AMWC}$) may be used. At 20 MHz the minimum command to $\overline{CAS}$ delay is 148 ns while the maximum data valid delay is 144 ns.

The reduced 8202 clock frequency still satisfies no wait state read operation from early read and will insert no more than one wait state for write (assuming no conflict with refresh). 20 MHz 8202 operation will however require using the 2118-4 to satisfy read access time.

Note that slowing the 8202 to 22.2 MHz guarantees valid data within 10 ns after $\overline{CAS}$ and allows using the 2118-7. Since this analysis is totally based on worst case minimum and maximum delays, the designer should evaluate the timing requirements of his specific implementation.

It should be noted that the 8202 $\overline{SACK}$ is equivalent to $\overline{XACK}$ timing if the cycle being executed was delayed by refresh. Delaying $\overline{SACK}$ until $\overline{XACK}$ time causes the CPU to enter wait states until the cycle is completed. If the cycle is a read cycle, the $\overline{XACK}$ timing guarantees data is valid at the CPU before RDY is issued to the CPU.

The use of the early command signals also solves a problem not mentioned previously. The cycle rate of the 8202 @ 20 MHz requires that commands (from leading edge to leading edge) be separated by a minimum of 695 ns. The maximum mode 8086 however may issue a read command 600 ns after the normal write command. For the early read command and advanced write command, 725 ns are guaranteed between commands.

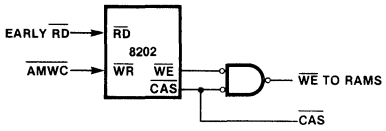


Figure 5C2.2. Delayed Write to Dynamic RAMs

APPENDIX I
BUS CONTENTION AND ITS EFFECT ON SYSTEM INTEGRITY

SYSTEM ARCHITECTURE

As higher performance microprocessors have become available, the architecture of microprocessor systems has been evolving, again placing demands on memory. For many years, system designers have been plagued with the problem of bus contention when connecting multiple memories to a common data bus. There have been various schemes for avoiding the problem, but device manufacturers have been unable to design internal circuits that would guarantee that one memory device would be "off" the bus before another device was selected. With small memories (512x8 and 1Kx8), it has been traditional to connect all the system address lines together and utilize the difference between t_{ACC} and t_{CO} to perform a decode to select the correct device (as shown in Figure 1).

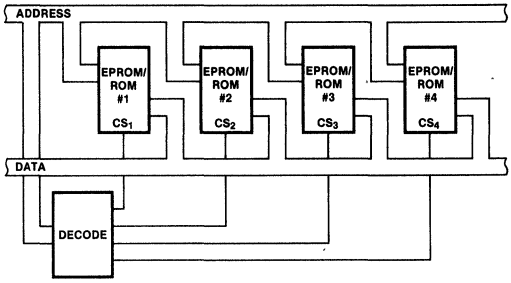


Figure 1. Single Control Line Architecture

With the 1702A, the chip select to output delay was only 100 ns shorter than the address access time; or to state it another way, the t_{ACC} time was 1000 ns while the t_{CO} time was 900 ns. The 1702A t_{ACC} performance of 1000 ns was suitable for the 4004 series microprocessors, but the 8080 processor required that the corresponding numbers be reduced to $t_{ACC} = 450$ ns and $t_{CO} = 120$ ns. This allowed a substantial improvement in performance over the 4004 series of microprocessors, but placed a substantial burden on the memory. The 2708 was developed to be compatible with the 8080 both in access time and power supply requirements. A portion of each 8080 machine cycle time had to be devoted to the architecture of the system decoding scheme used. This devoted portion of the machine cycle included the time required for the system controller (8224) to perform its function before the actual decode process could begin.

Let's pause here and examine the actual decode scheme that was used so we can understand how the control functions that a memory device requires are related to system architecture.

The 2708 can be used to illustrate the problem of having a single control line. The 2708 has only one read control function, chip select ($\overline{CS}$), which is very fast ($t_{CO} = 120$ ns) with respect to the overall access time ($t_{ACC} = 450$ ns) of the 2708. It is this time difference (330 ns) that is used to perform the decode function, as illustrated in Figure 2. The scheme works well and does not limit system performance, but it does lead to the possibility of bus contention.

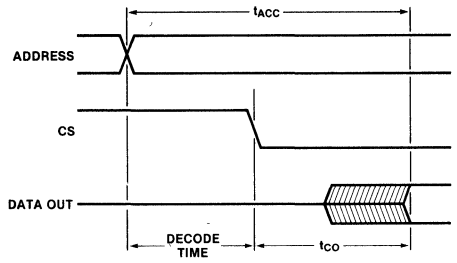


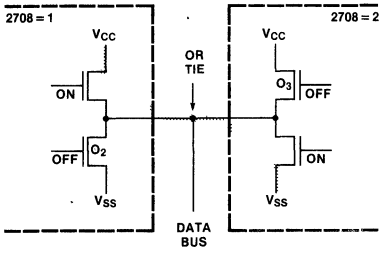
Figure 2. Single Line Control Architecture

BUS CONTENTION

There are actually two problems with the scheme described in the previous section. First, if one device in a multiple memory system has a relatively long deselect time, and a relatively fast decoder is used, it would be possible to have another device selected at the same time. If the two devices thus selected were reading opposite data; that is, device number one reading a HIGH and device number two reading a LOW, the output transistors of the two memory devices would effectively produce a short circuit, as Figure 3 illustrates. In this case, the current path is from V_{CC} on device number one to GND on device number two. This current is limited only by the "on" impedance of the MOS output transistors and can reach levels in excess of 200 mA per device. If the MOS transistors have a lot of "extra" margin, the current is usually not destructive; however, an instantaneous load of 400 mA can produce "glitches" on the V_{CC} supply—glitches large enough to cause standard TTL devices to drop bits or otherwise malfunction, thus causing incorrect address decode or generation.

The second problem with a single control line scheme is more subtle. As previously mentioned, there is only one control function available on the 2708 and any decoding scheme must use it out of necessity. In addition, any inadvertent changes in the state of the high order address lines that are inputs to the decoder will cause a change in the device that is selected. The result is the same as before—bus contention, only from a different source. The deselected device cannot get "off" the bus before the selected one is "on" the bus as the addresses rapidly change state. One approach to solving this problem would be to design (and specify as a maximum) devices

with t_{DF} time less than t_{CO} time, thereby assuring that if one device is selected while another is simultaneously being deselected, there would be some small (20 ns) margin. Even with this solution, the user would not be protected from devices which have very fast t_{CO} times (t_{CO} is specified as a maximum).



RESULTS OF IMPROPER TIMING WHEN OR TYING MULTIPLE MEMORIES.

Figure 3. Results of Improper Timing when OR Tying Multiple Memories

The only sure solution appears to be the use of an external bus driver/transceiver that has an independent enable function. Then that function, not the "device selecting function," or addresses, could control the flow of data "on" and "off" the bus, and any contention problems would be confined to a particular card or area of a large card. In fact, many systems are implemented that way—the use of bus drivers is not at all uncommon in large systems where the drive requirements of long, highly capacitive interconnecting lines must be taken into consideration—it also may be the reason why more system designers were not aware of the bus contention problem until they took a previously large (multicard) system and, using an advanced micorprocessor and higher density memory devices, combined them all on one card, thereby eliminating the requirement for the bus drivers, but experiencing the problem of bus contention as described above.

THE MICROPROCESSOR/MEMORY INTERFACE

From the foregoing discussion, it becomes clear that some new concepts, both with regard to architecture and performance are required. A new generation of two control line devices is called for with general requirements as listed below:

1. Capability to control the data "on" and "off" the system bus, independent of the device selecting function identified above.

2. Access time compatible with the high performance microprocessors that are currently available.

Now let's examine the system architecture that is required to implement the two line control and prevent bus contention. This is shown in the form of a timing diagram (Figure 4). As before, addresses are used to generate the unique device selecting function, but a separate and independent Output Enable (OE) control is now used to gate data "on" and "off" the system data bus. With this scheme, bus contention is completely eliminated as the processor determines the time during which data must be present on the bus and then releases the bus by way of the Output Enable line, thus freeing the bus for use by other devices, either memories or peripheral devices. This type of architecture can be easily accomplished if the memory devices have two control functions, and the system is implemented according to the block diagram shown in Figure 5. It differs from the previous block diagram (shown in Figure 1) in that the control bus, which is connected to all memory Output Enable pins, provides separate and independent control over the data bus. In this way, the microprocessor is always in control of the system; while in the previous system, the microprocessor passed control to the particular memory device and then waited for data to become available. Another way to look at it is, with a single control line the sytem is always asynchronous with respect to microprocessor/memory communications. By using two control lines, the memory is synchronized to the processor.

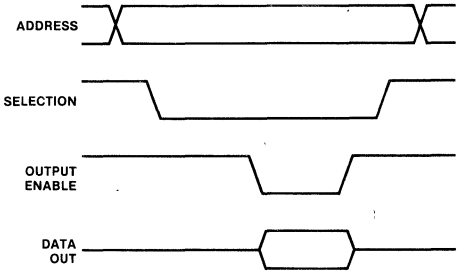


Figure 4. Two Control Line Architecture

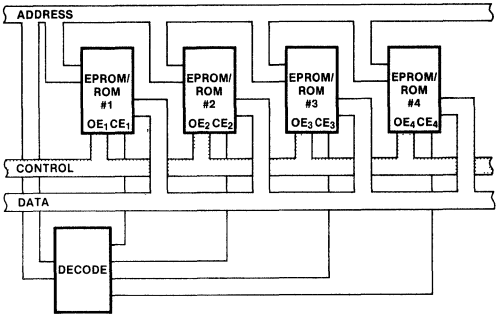


Figure 5. Two Control Line Architecture

intel

APPLICATION NOTE

AP-123

March 1982

Graphic CRT Design Using the Intel® 8089

Hal Kop
Microcomputer Applications

MARCH 1982
ORDER NUMBER: 210355-001

Graphic CRT Design Using the Intel 8089

Contents

INTRODUCTION

The purpose of this application note is to provide the reader with the conceptual tools and factual information needed to apply the Intel 8089 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 Intel 8089 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 Intel 8089 (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

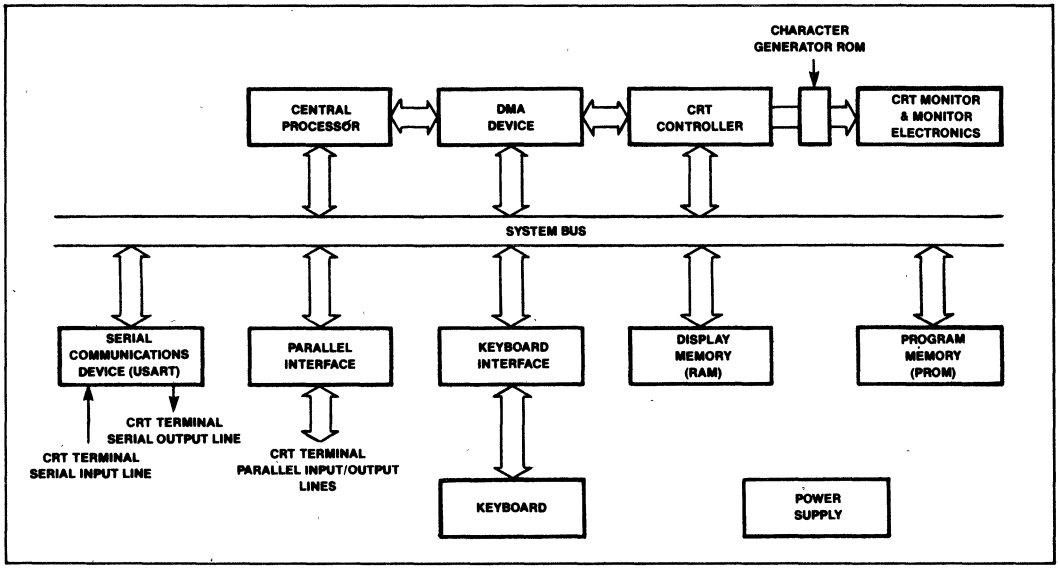


Figure 1. Typical CRT Terminal Block Diagram

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.

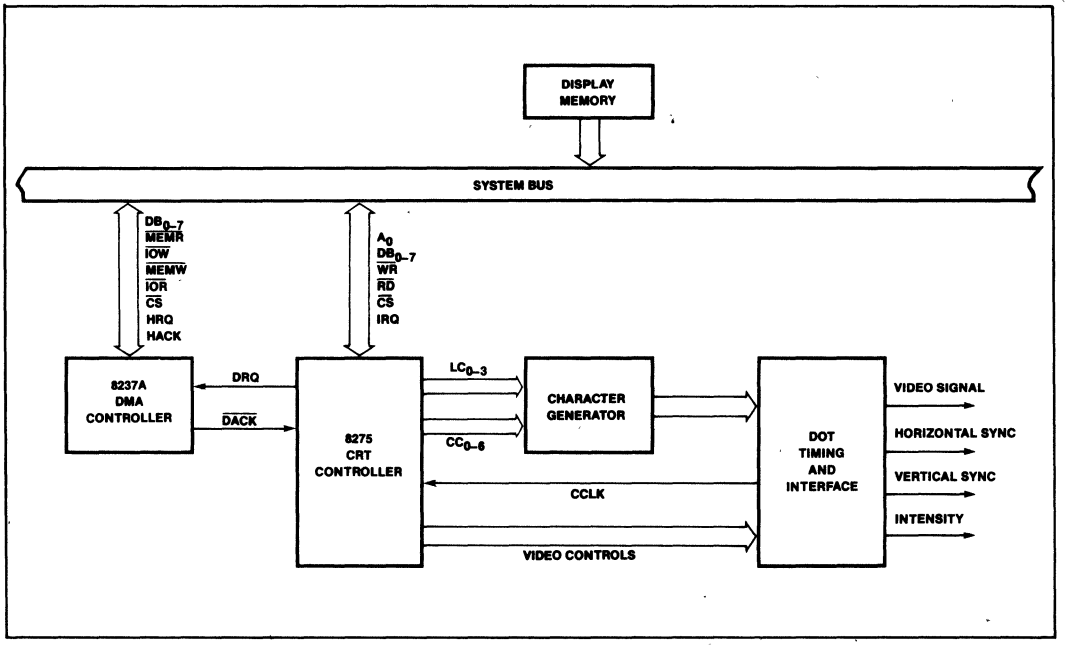


Figure 2. Components Involved in the CRT Refresh Function

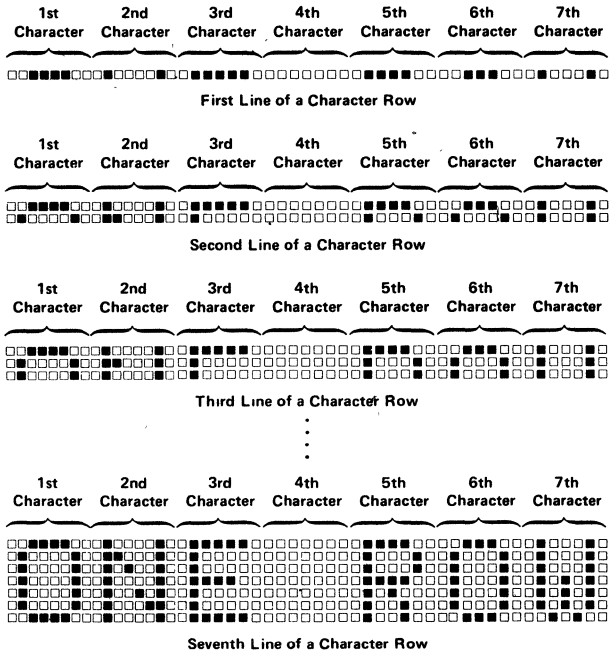


Figure 3. Character Row Display

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.

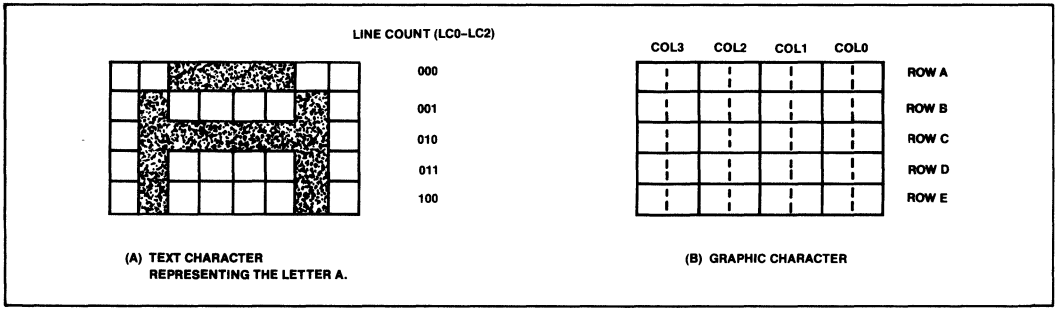


Figure 4. Character Structure

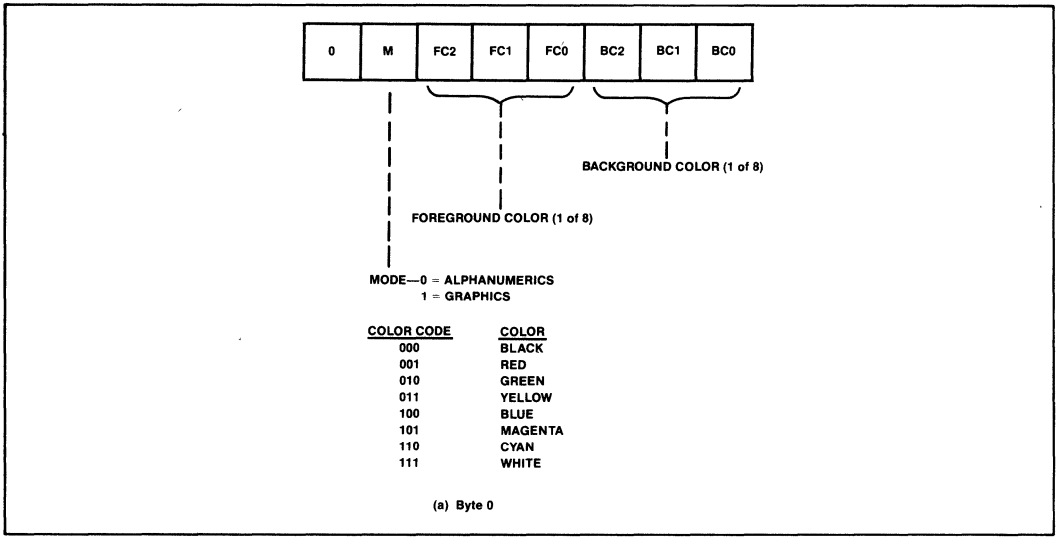
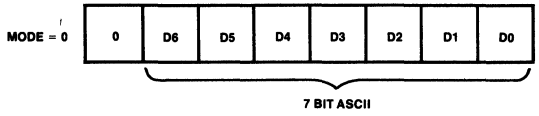
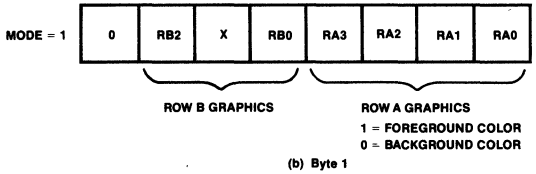


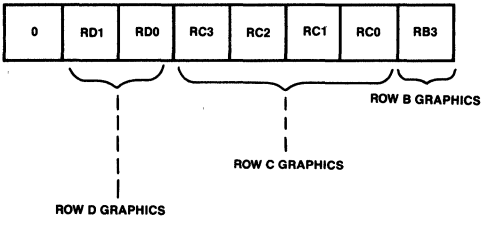
Figure 5. Display Character Specification



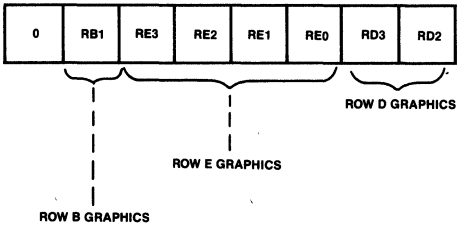


(b) Byte 1

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 Intel 8089 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

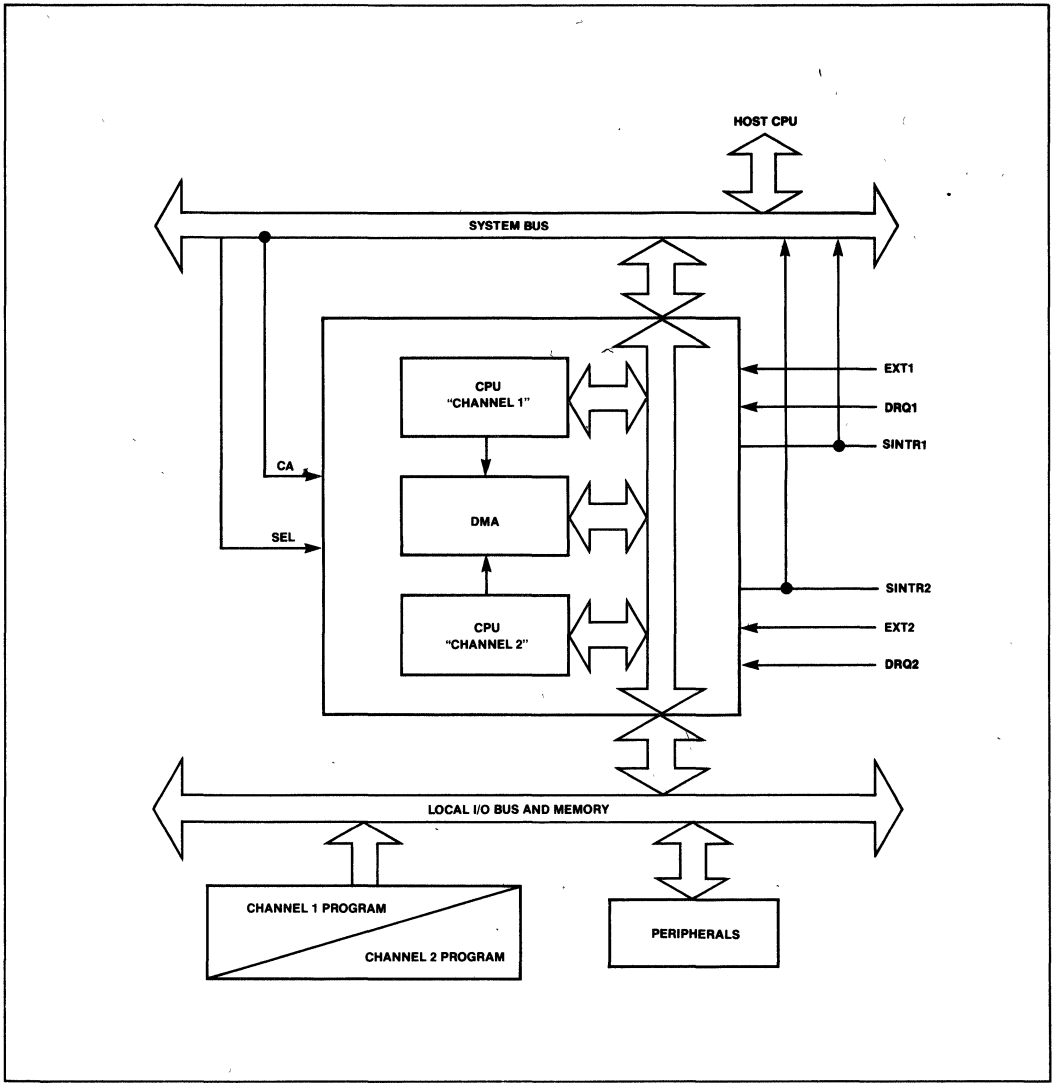


Figure 6. Simplified Block Diagram of the 8089

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.

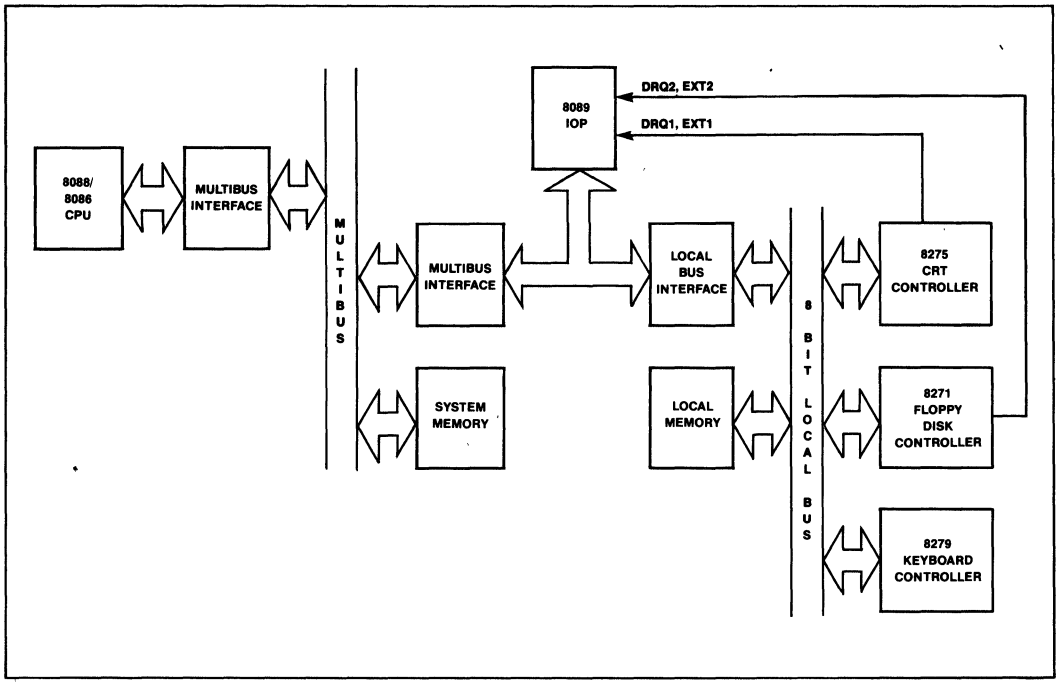


Figure 7. I/O System with Multiple Peripherals

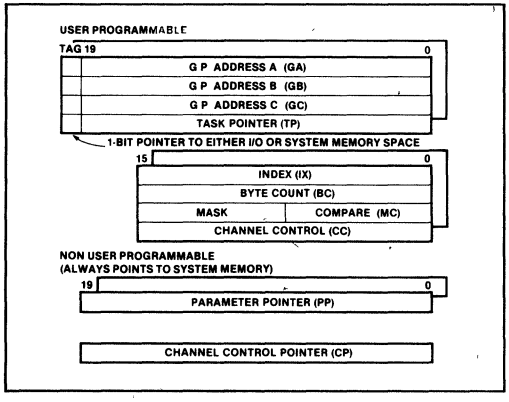


Figure 8. 8089 Register Set

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

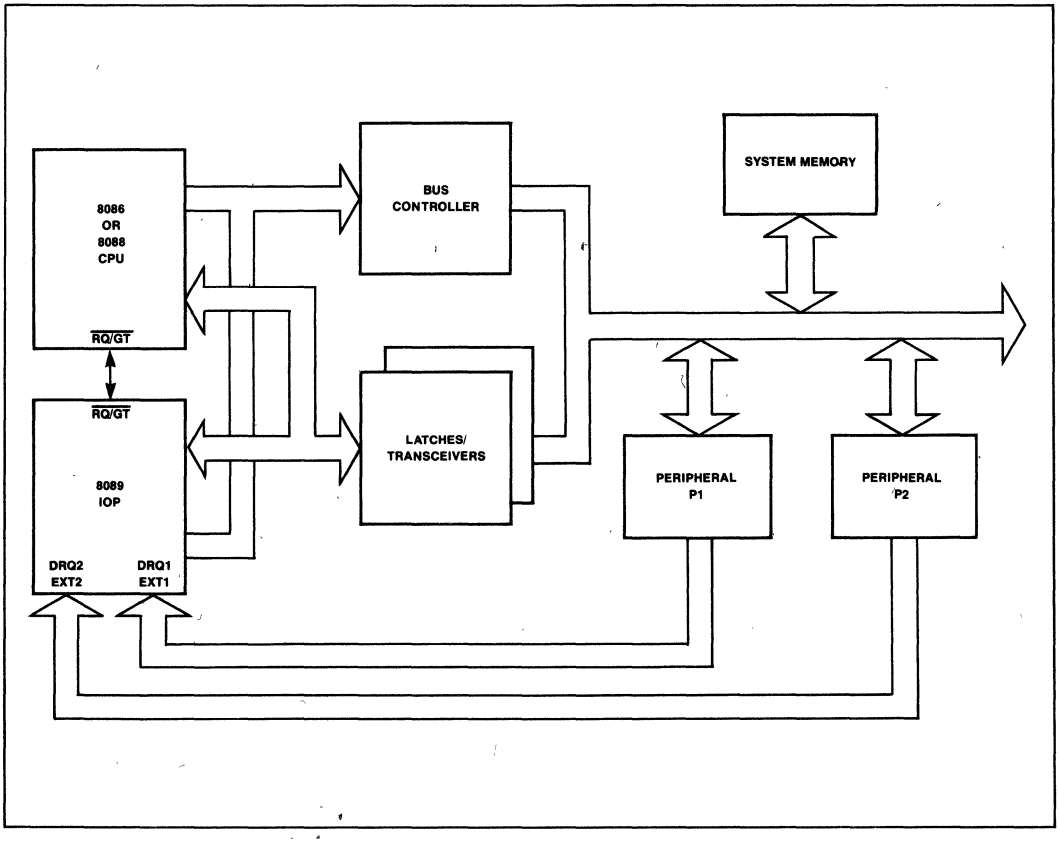


Figure 9. CPU and IOP in Local Configuration

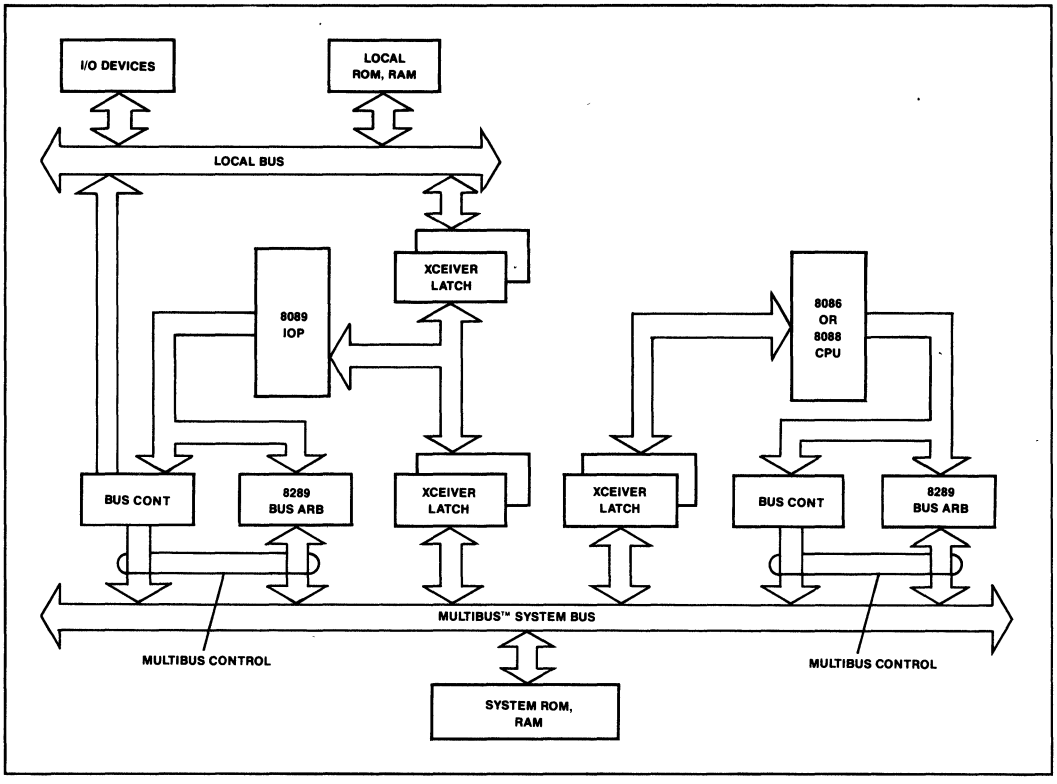


Figure 10. CPU and IOP in Remote Configuration

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

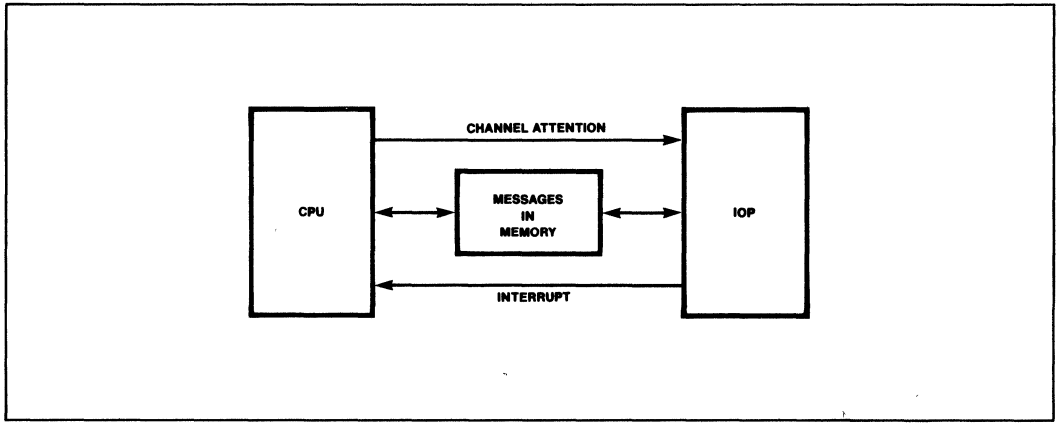


Figure 11. CPU/IOP Communication

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.

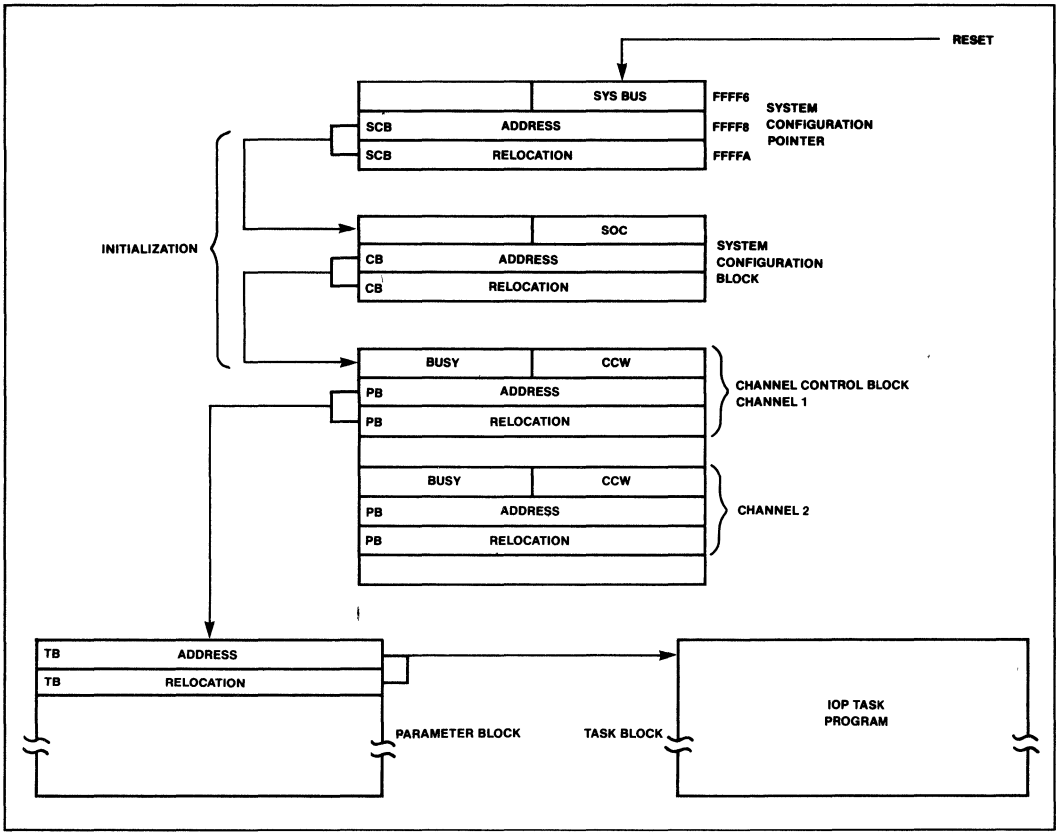


Figure 12. Linked Block Communication Structure

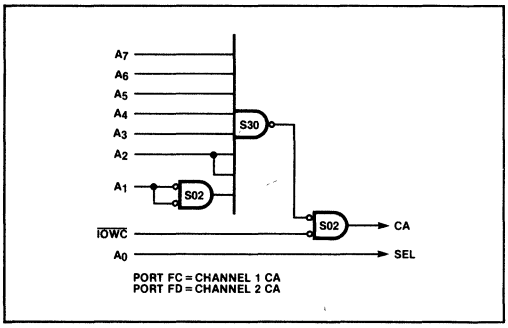
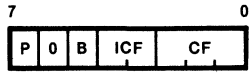


Figure 13. Channel Attention Decoding Circuit

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

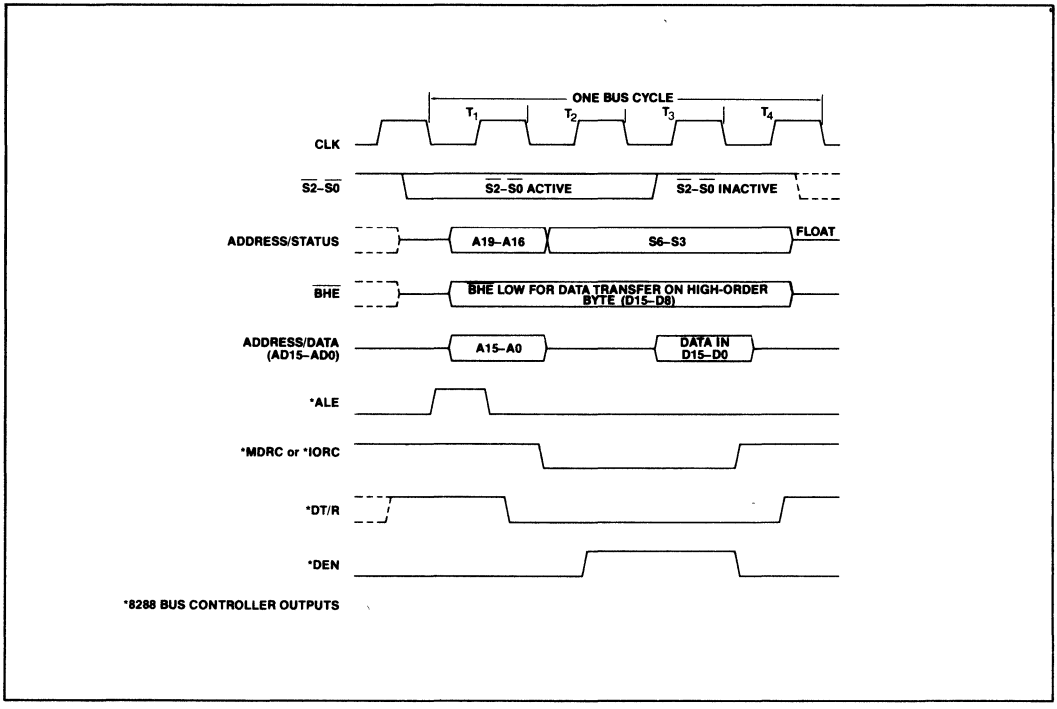


Figure 15. Read Bus Cycle

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 Intel 8089.

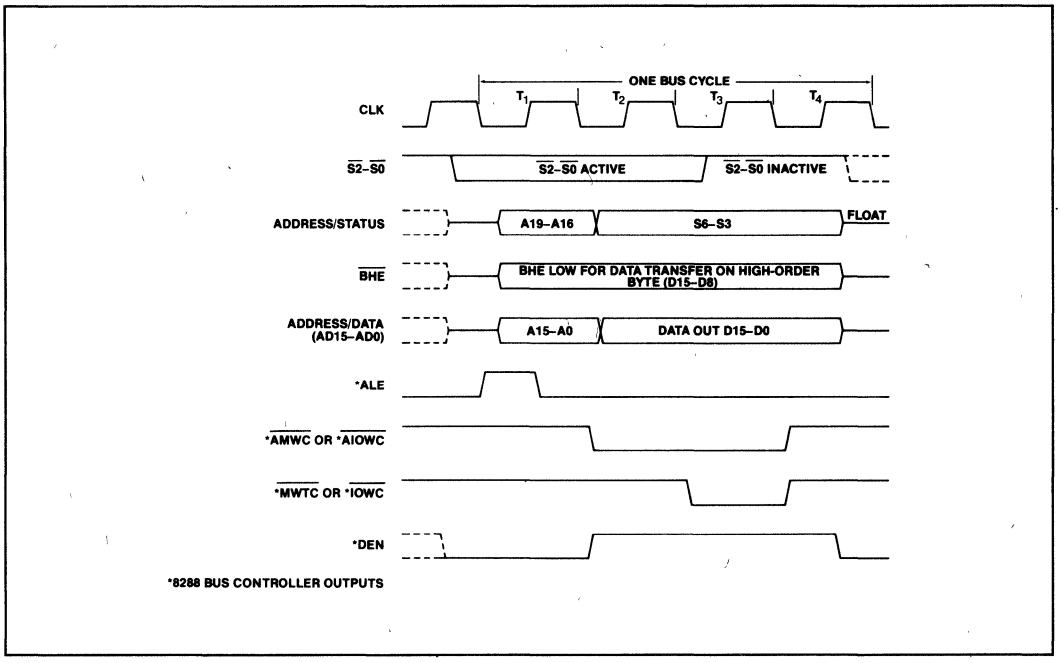


Figure 16. Write Bus Cycle

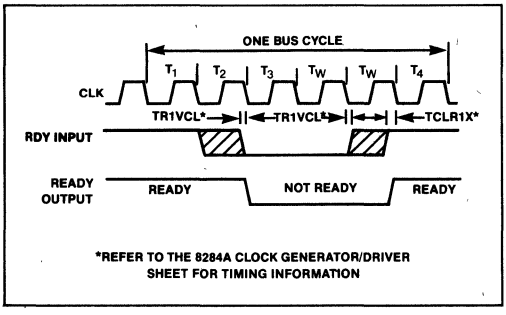


Figure 17. Wait-State Timing (Synchronous RDY Input)

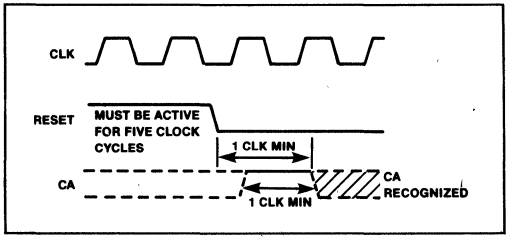


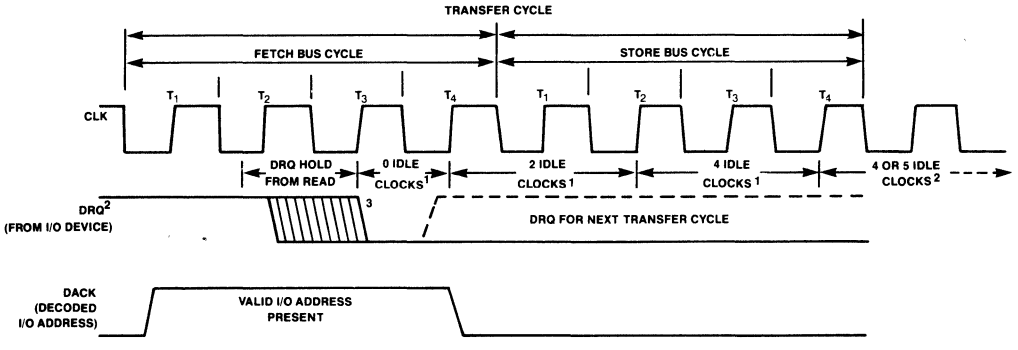
Figure 18. Reset and Channel Attention Timing

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.

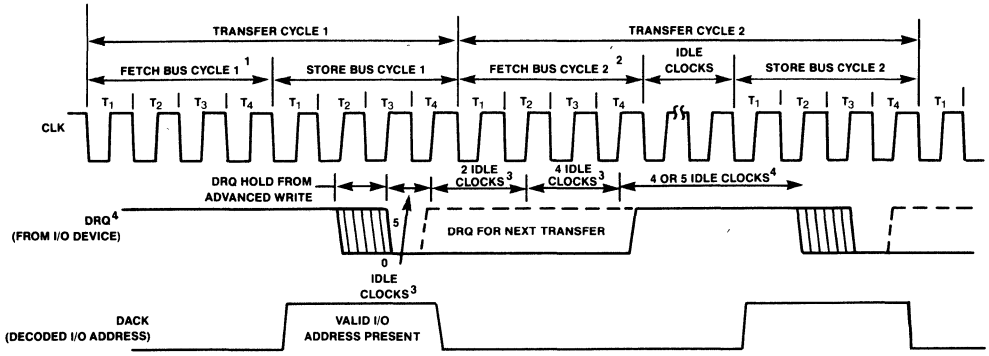
The system configuration then becomes that shown in Figure 21. The dual-port RAM becomes the only data path between the 8086 and the 8089. Access to this memory is time-shared between the 8086 and the 8089, with the 8089 taking less than 50% of the total time available. Since the 8089 does not access 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

210355-001

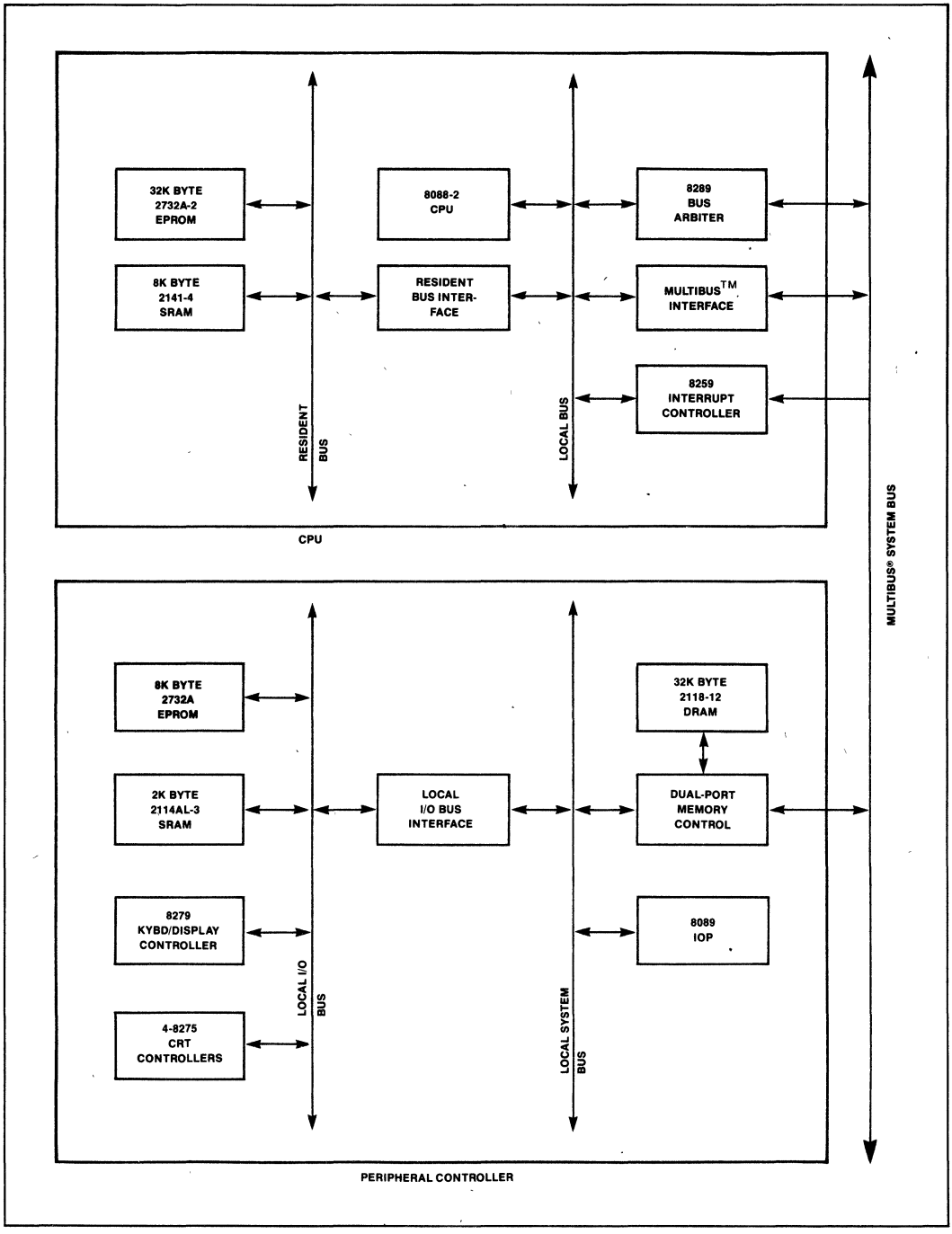


Figure 21. Remote Configuration with Dual-Port RAM

210355-001

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.

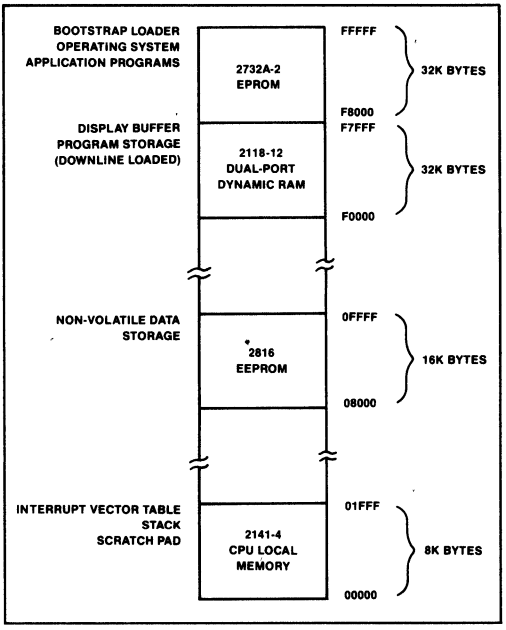


Figure 22. CPU Memory Organization

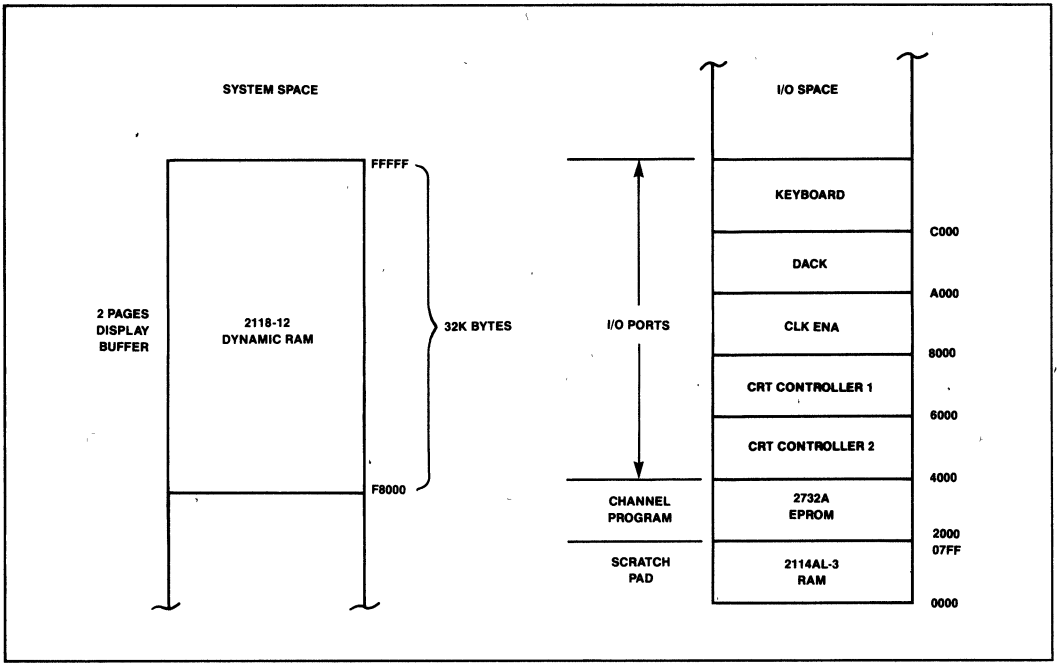


Figure 23. IOP Memory Organization

210355-001

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.

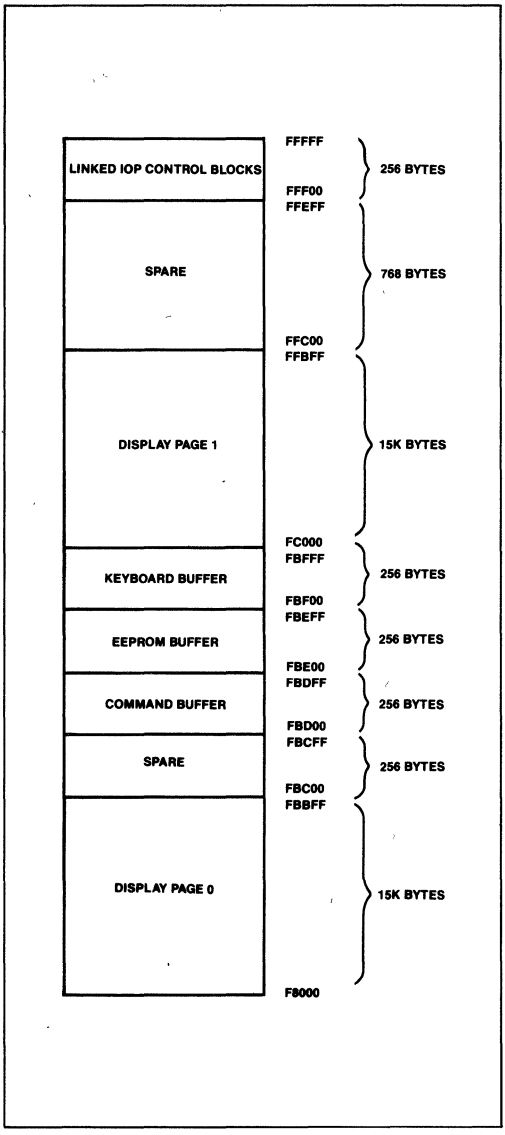


Figure 24. Organization of the Dual-Port RAM

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.

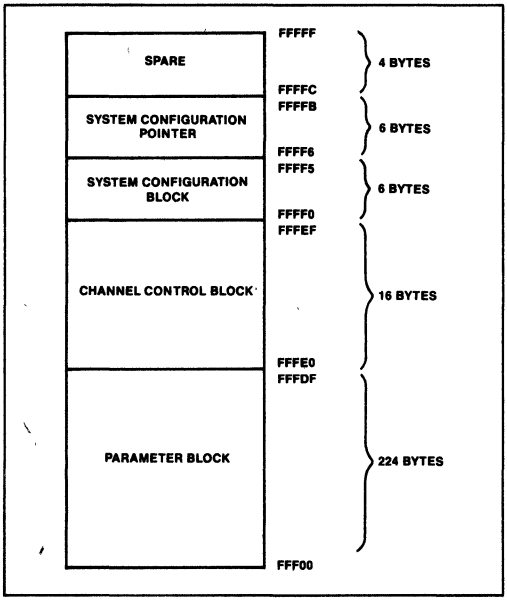


Figure 25. Organization of the Linked IOP Control Blocks Area

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

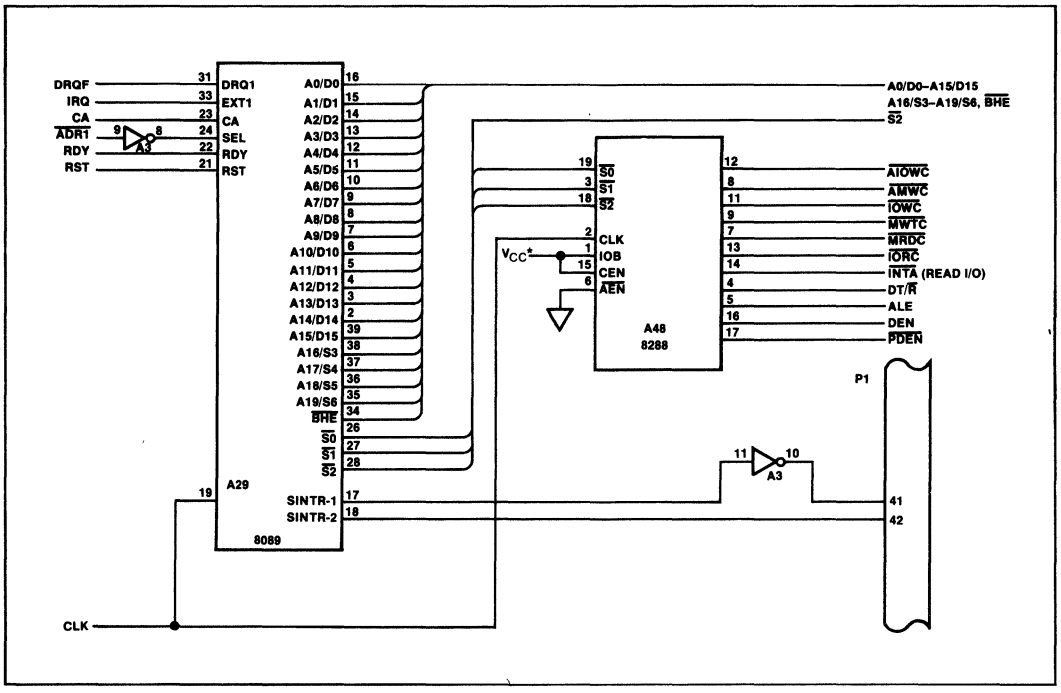


Figure 26. 8089 I/O Processor and 8288 Bus Controller

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 line S0, S1, and S2, and the system interrupt request lines, SINTR-l 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.

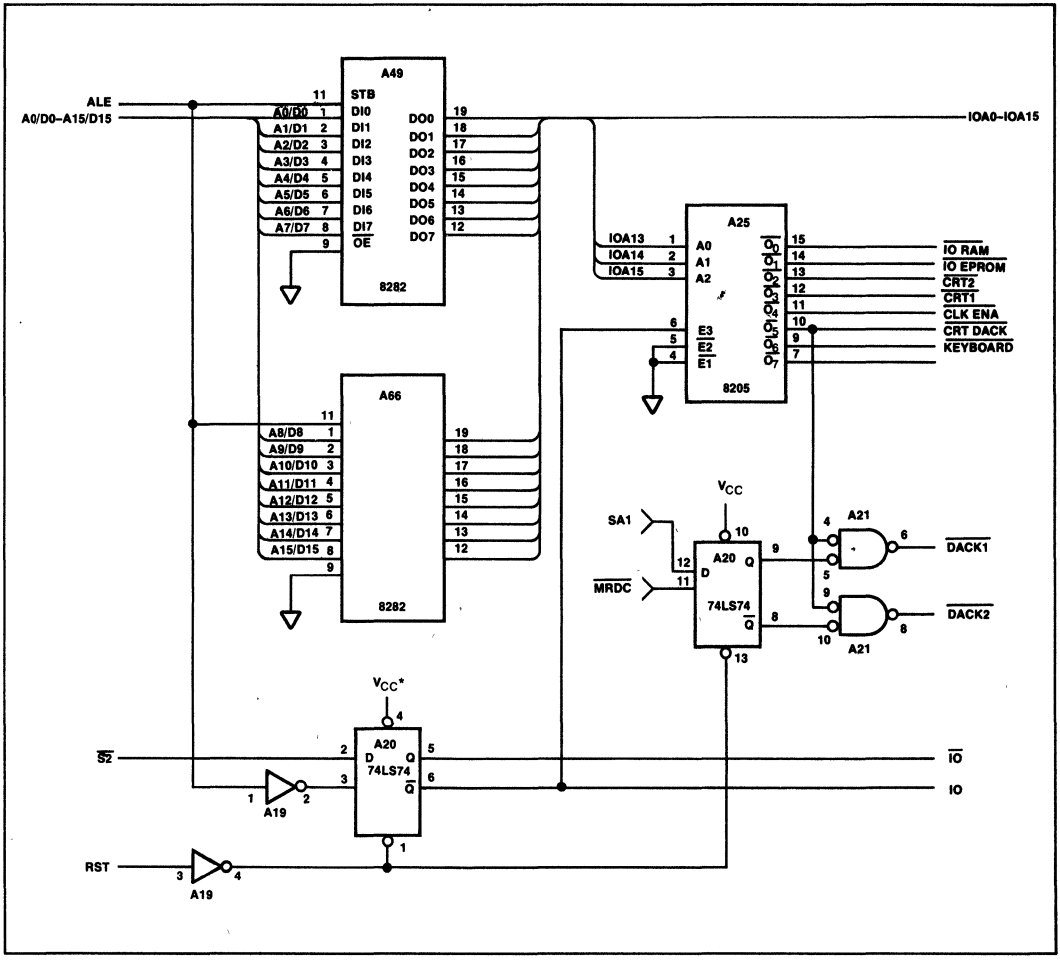


Figure 27. Address Latches, Decoders, and DACK Generator

210355-001

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.

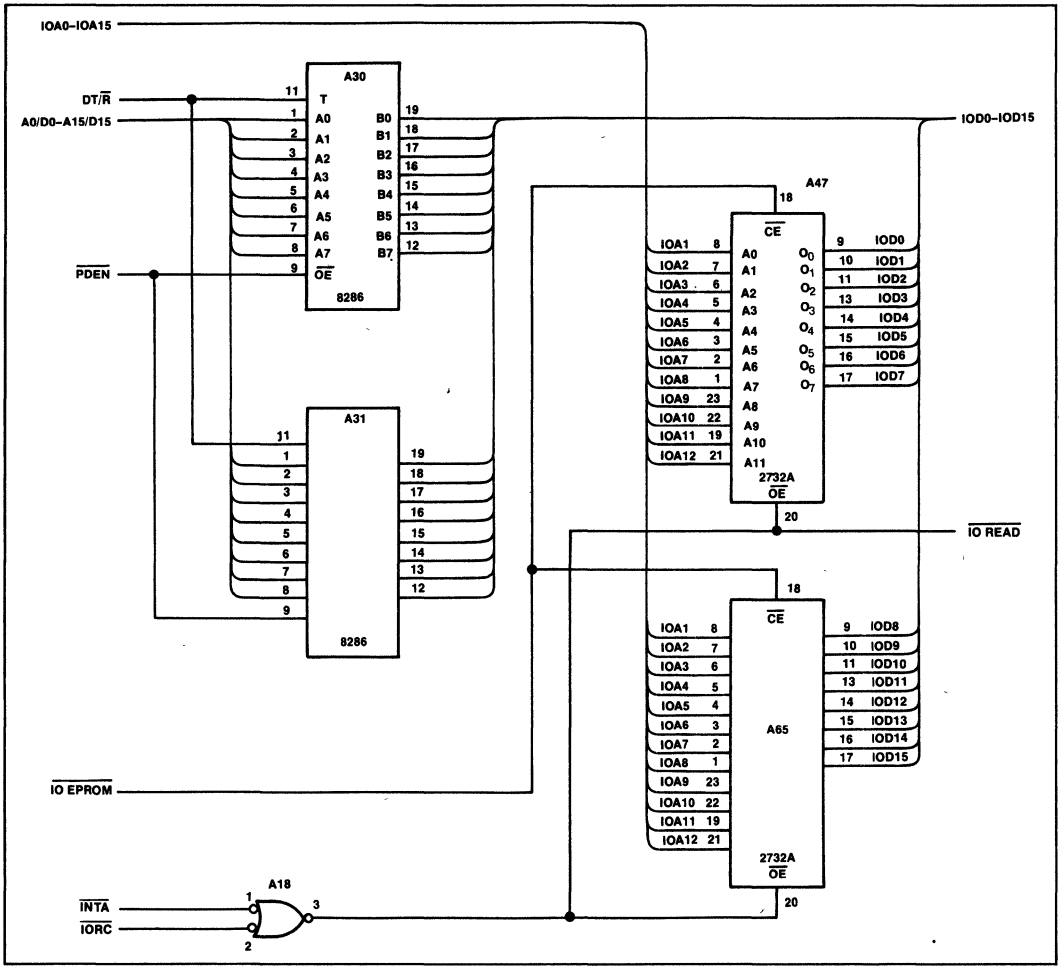


Figure 28. Bus Transceivers and EPROMs on I/O Bus

210355-001

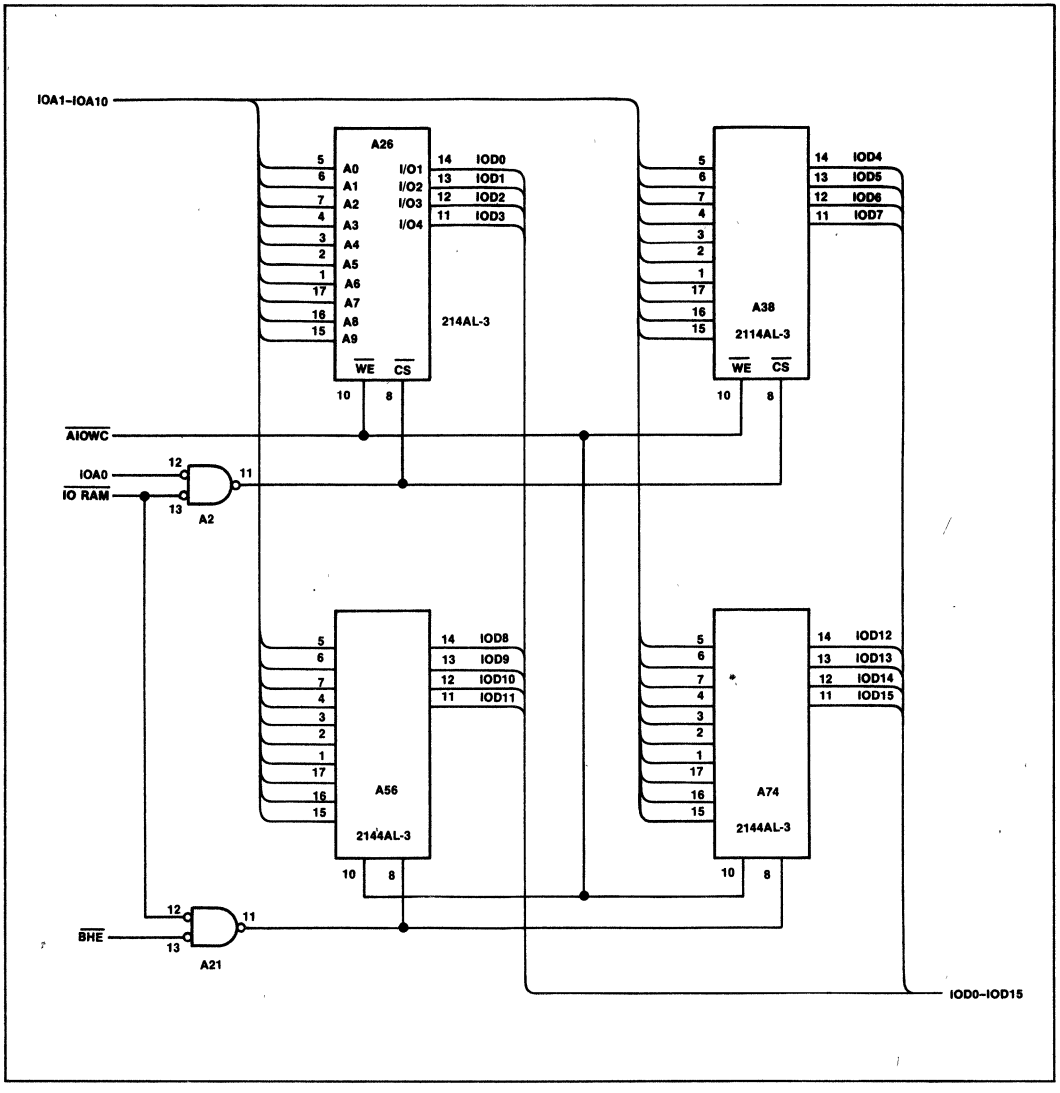


Figure 29. Static RAMs on I/O Bus

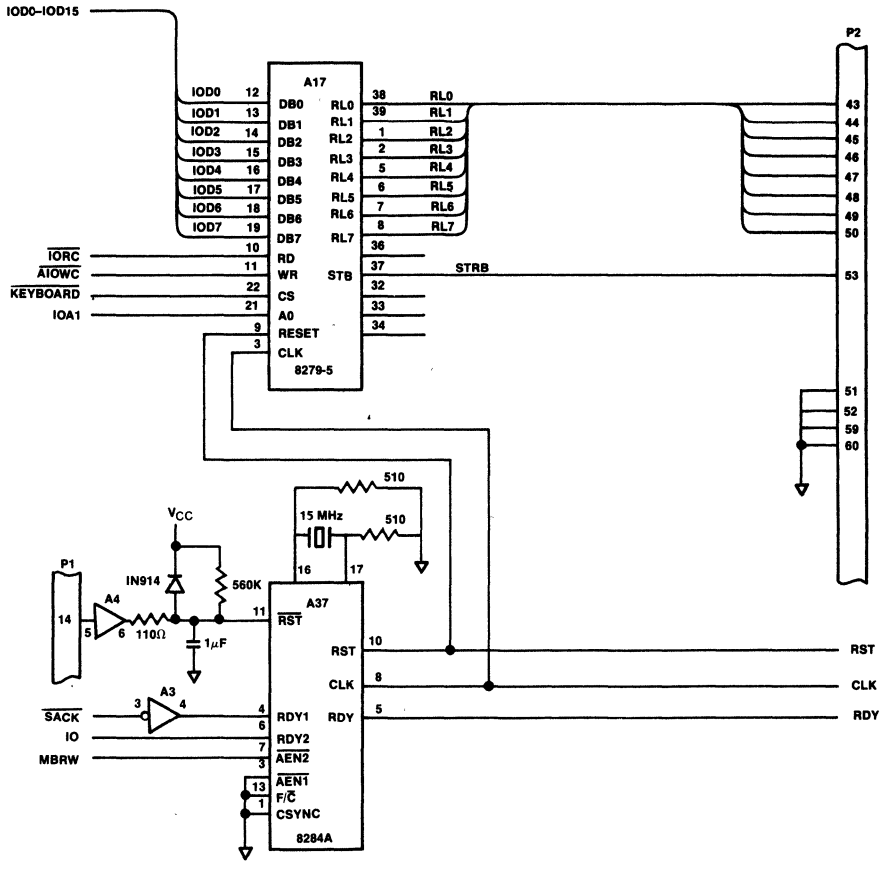


Figure 30. Keyboard Controller and Clock Generator

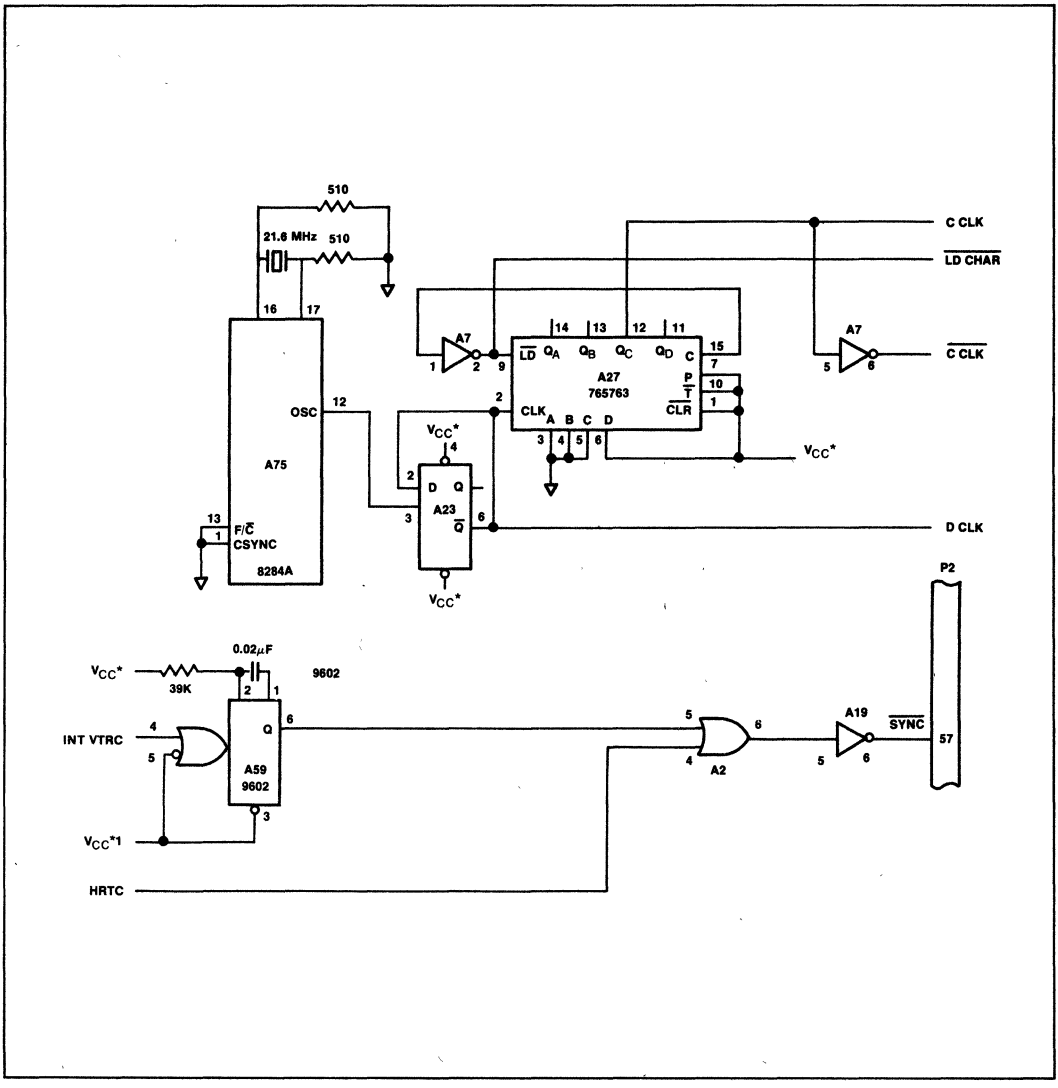


Figure 31. Character Clock Generator and Video Sync Pulse

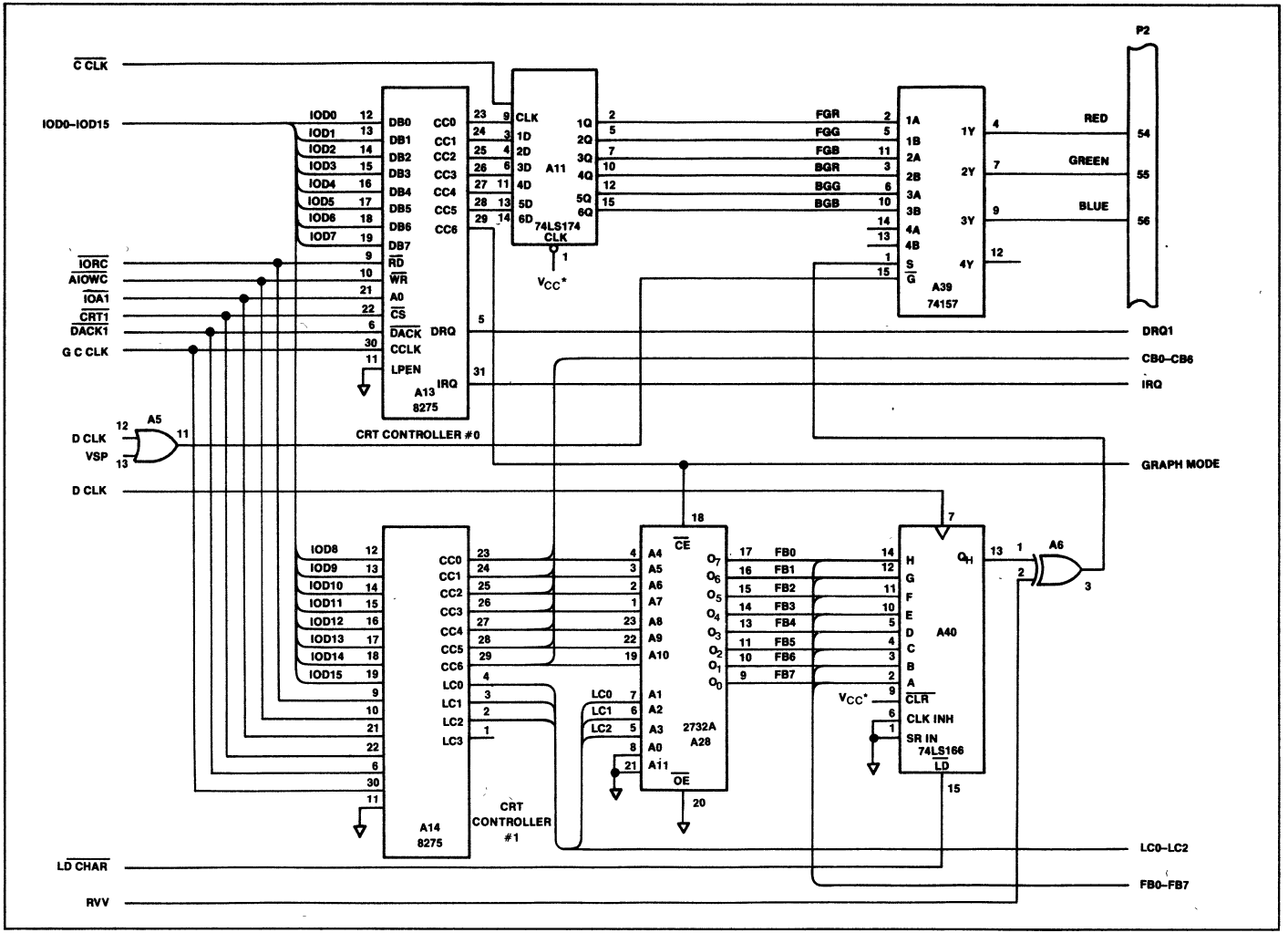


Figure 32. CRT Controllers, Color Multiplexer, and Character Generator

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).

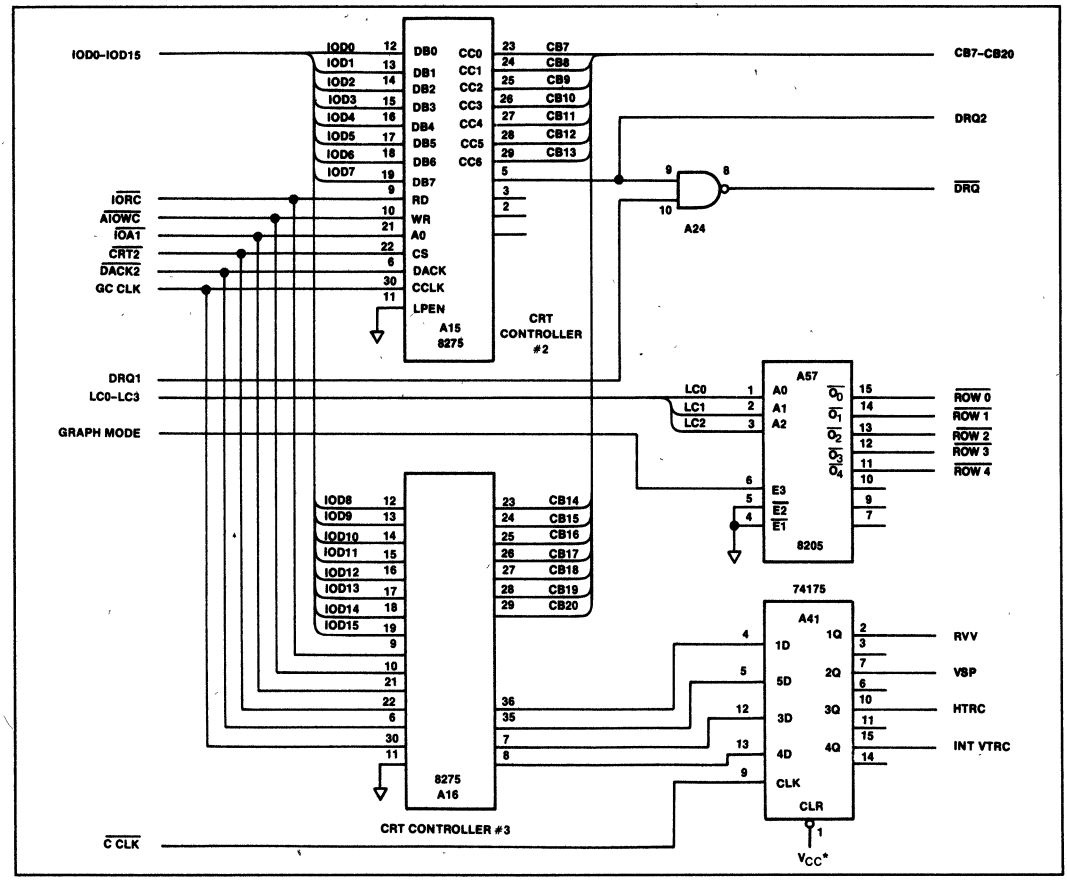


Figure 33. CRT Controllers, Line Decoder, and Video Control Signal Latch

210355-001

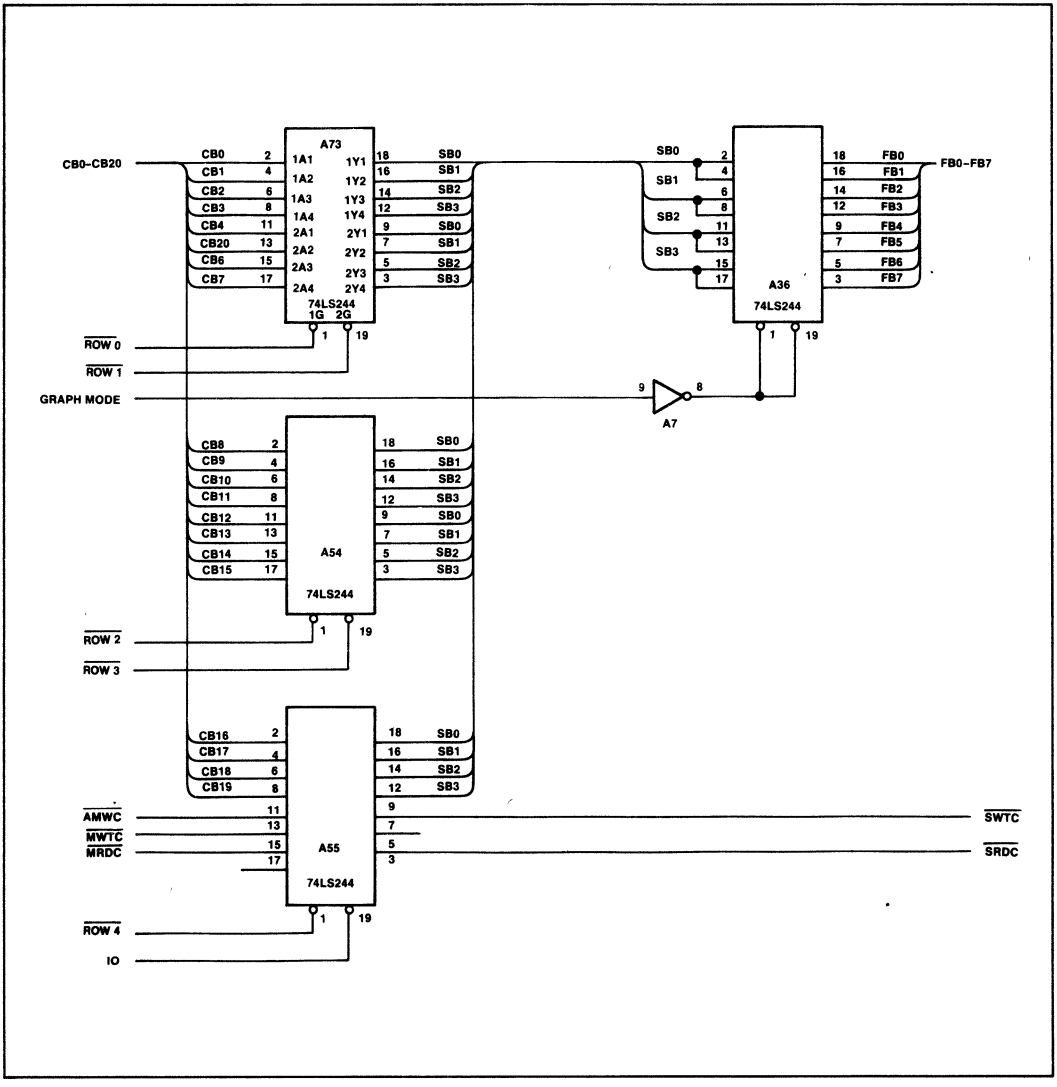


Figure 34. Tri-State Buffers for Graphic Color Information

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.

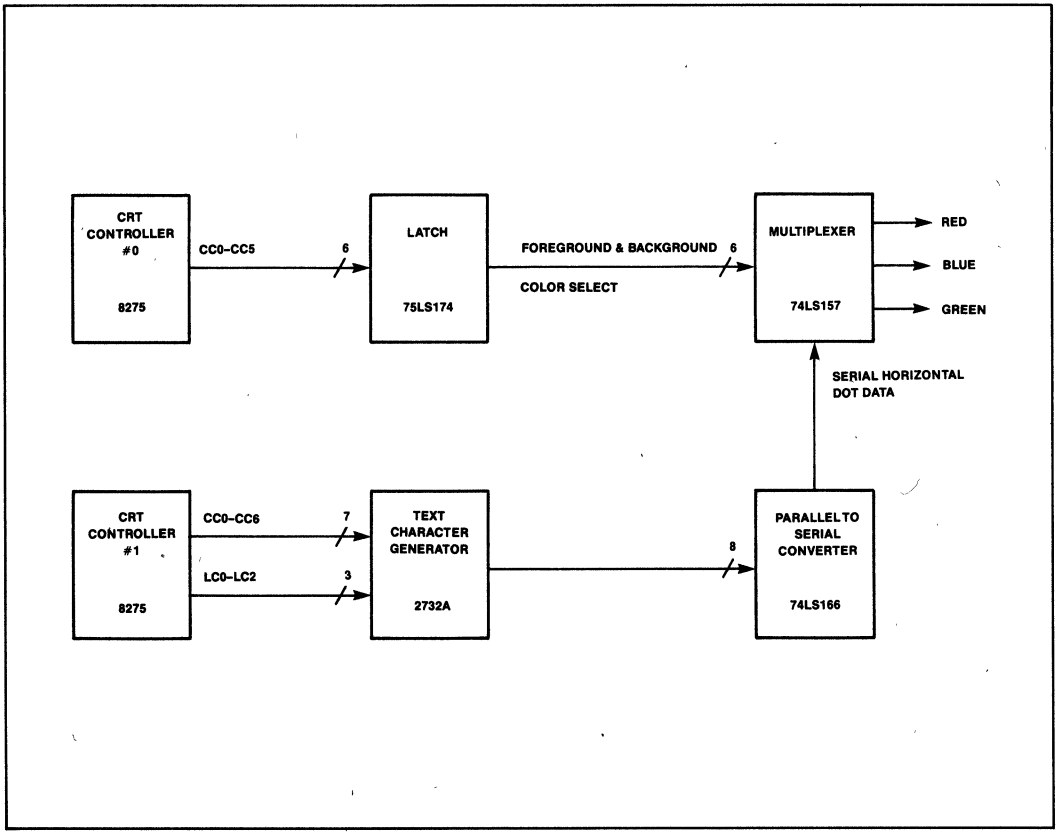


Figure 35. Processing of Text Characters

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.

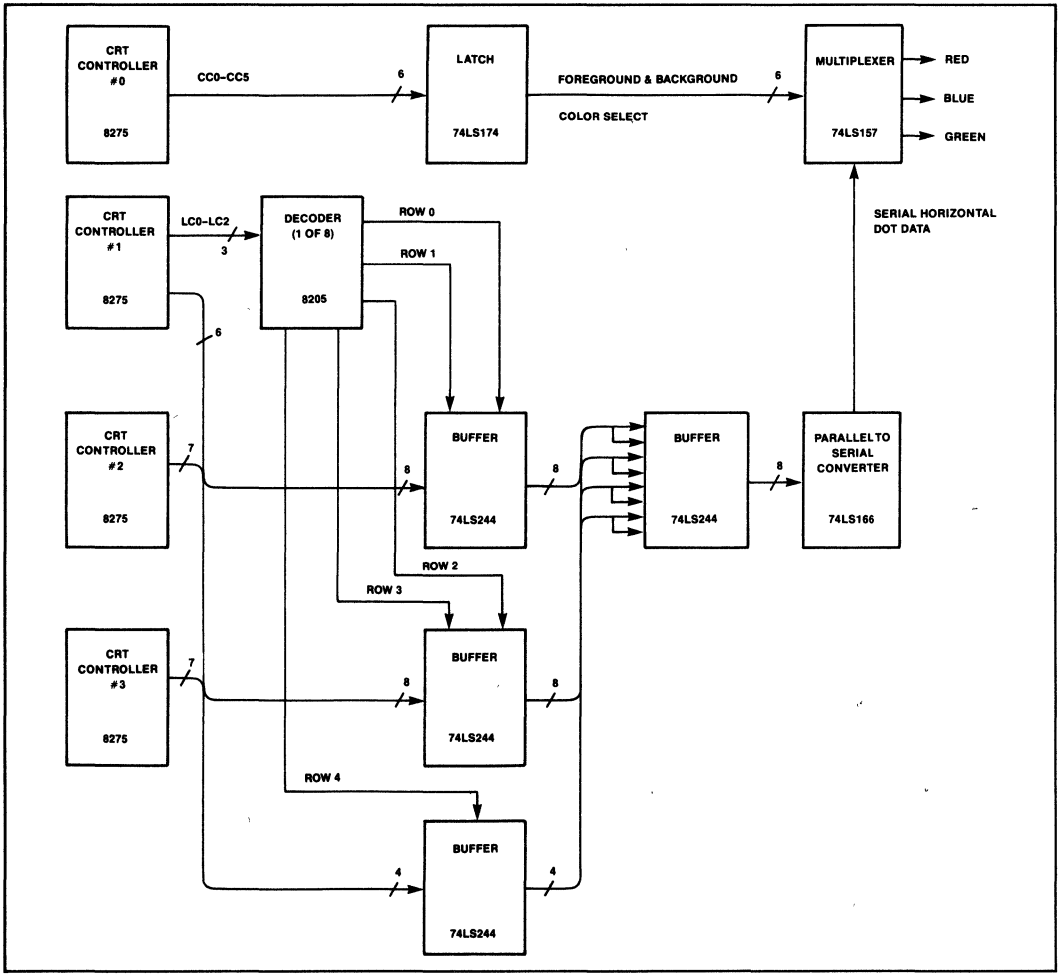


Figure 36. Processing of Graphic Characters

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.

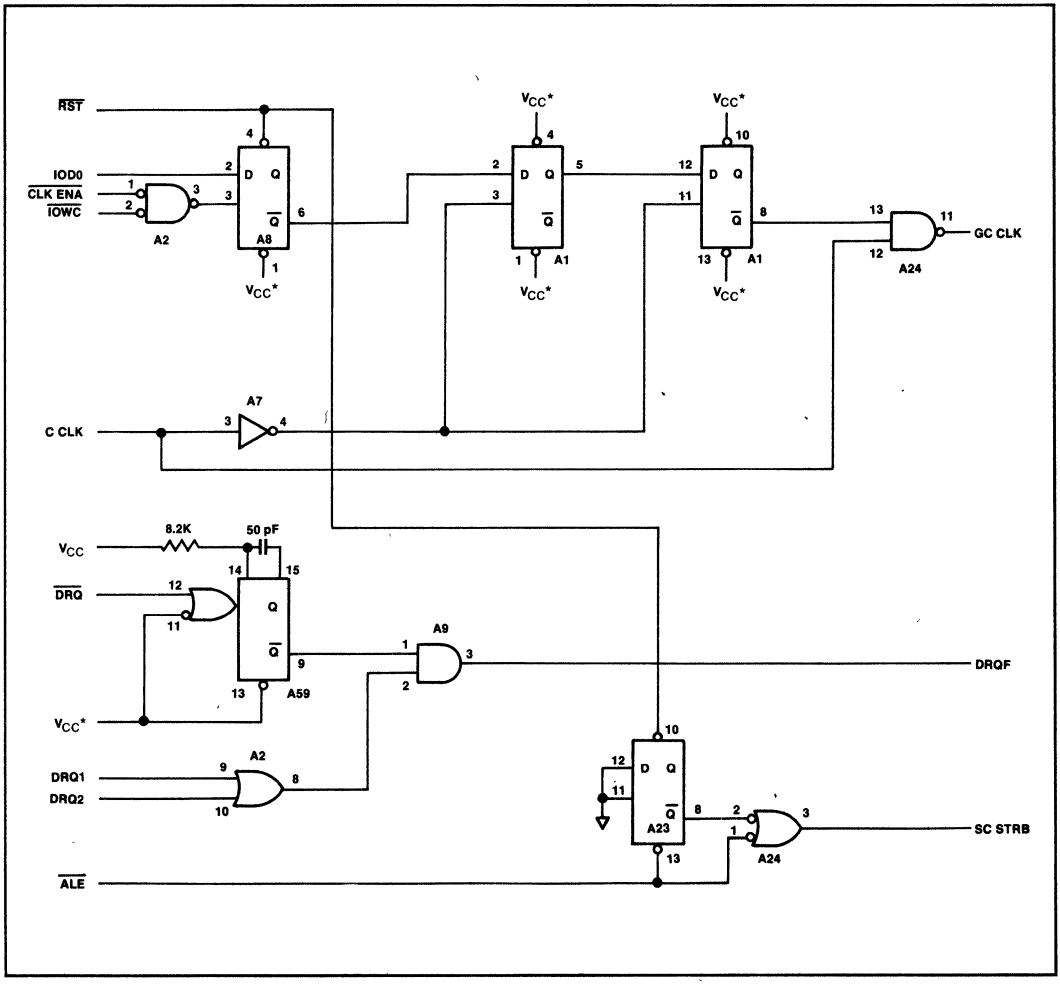


Figure 37. Circuits to Synchronize CRT Controllers and Generate DRQF

210355-001

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.

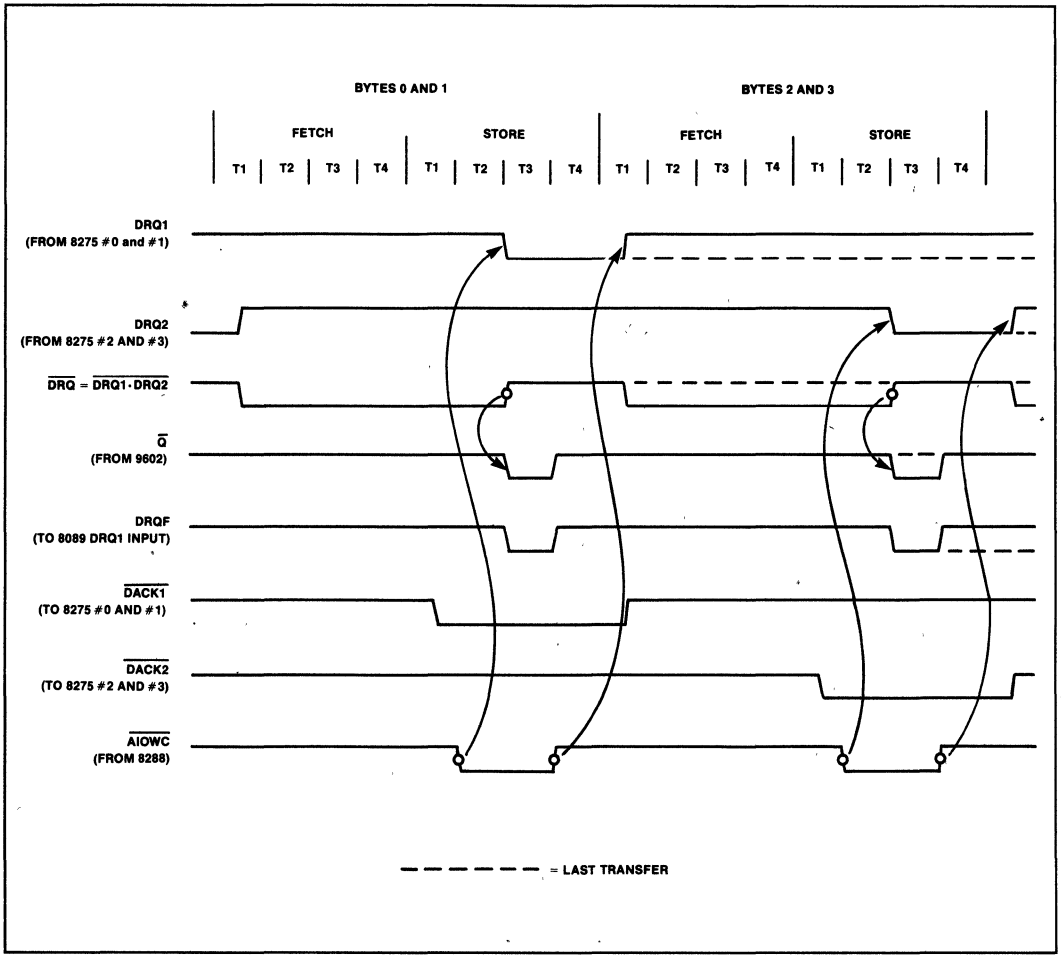


Figure 38. Derivation of DRQF Signal

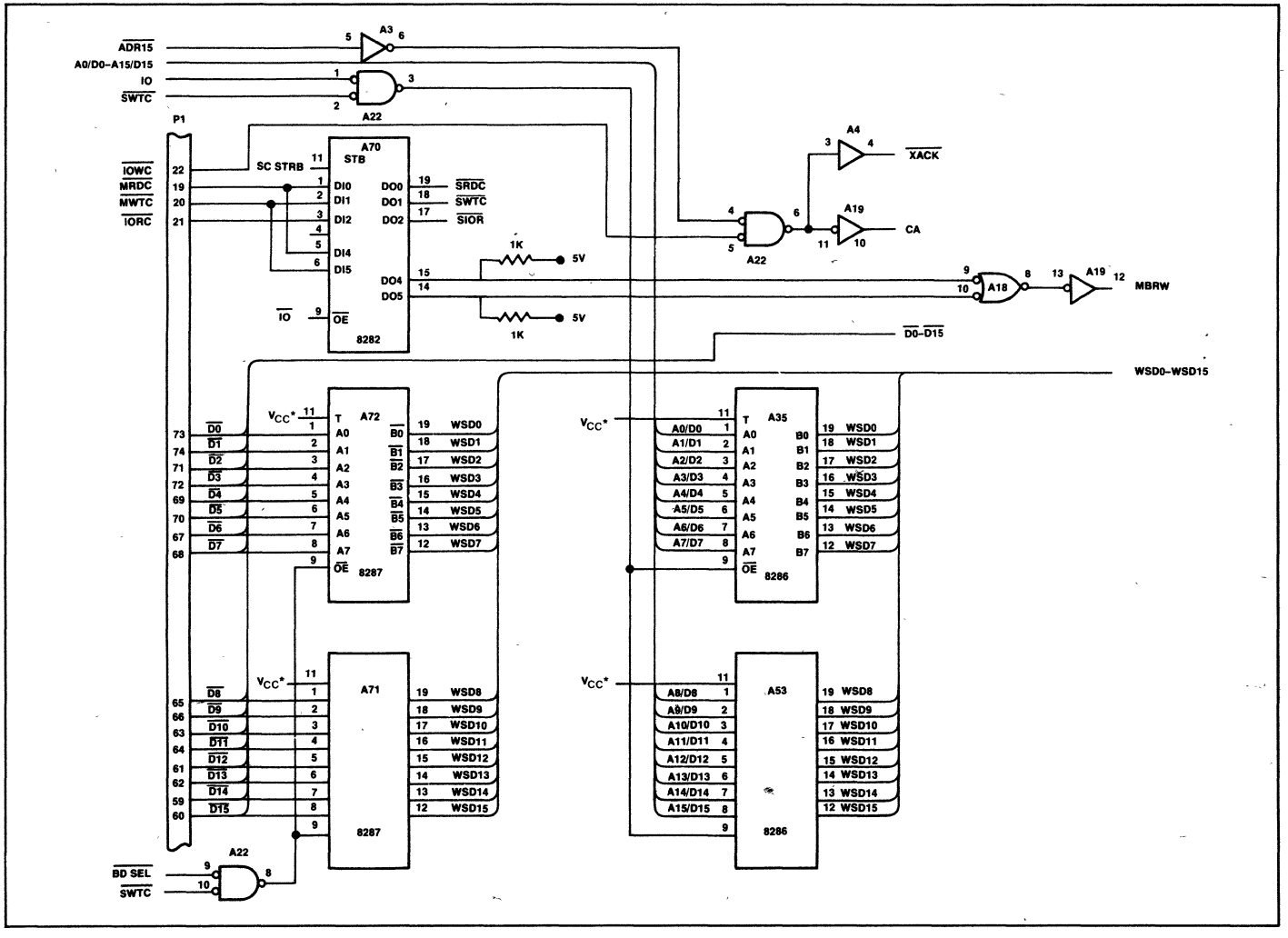


Figure 39. Multiplexer for Writing to Dual-Port RAM

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.

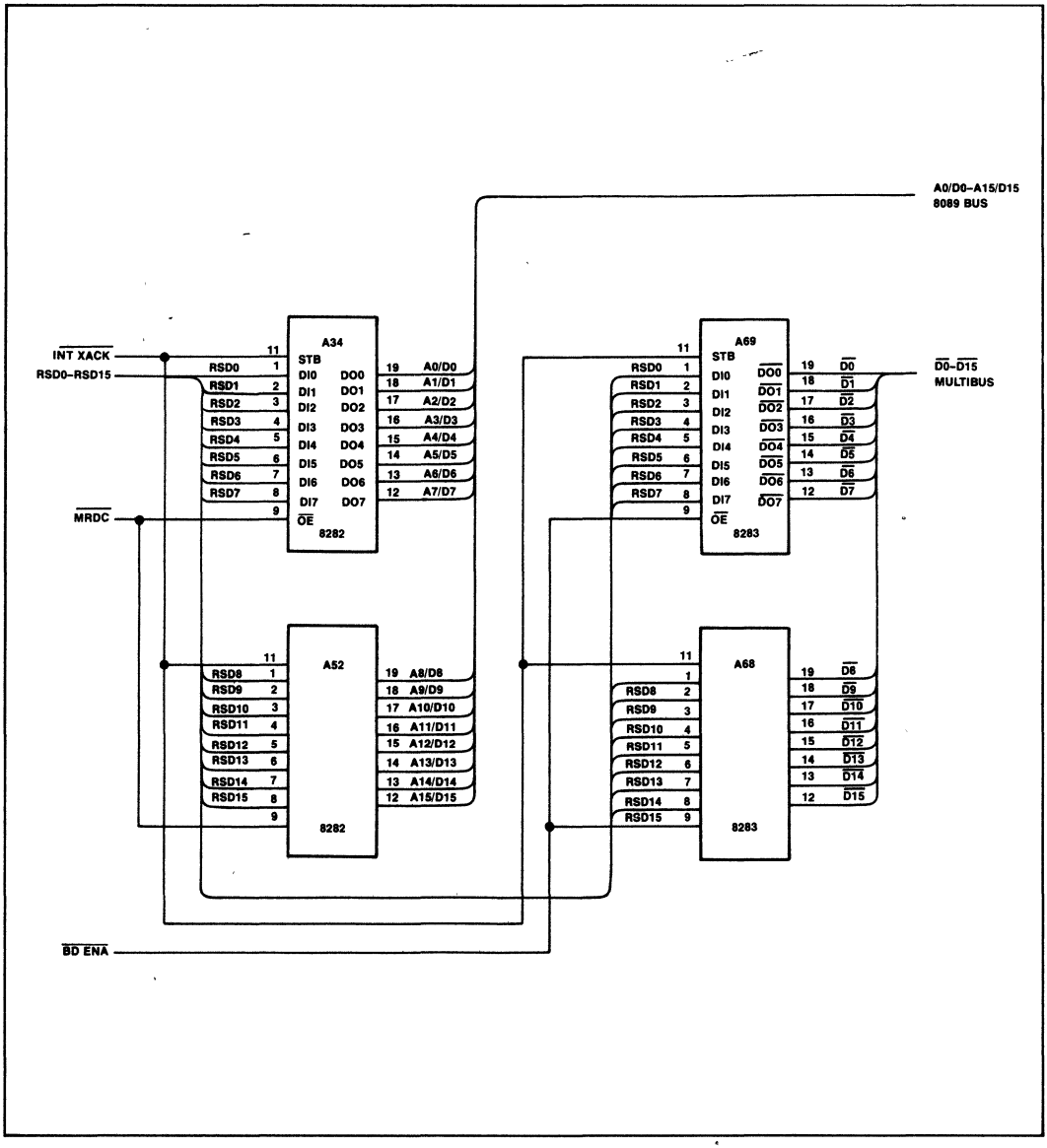


Figure 40. Demultiplexer for Reading from Dual-Port RAM

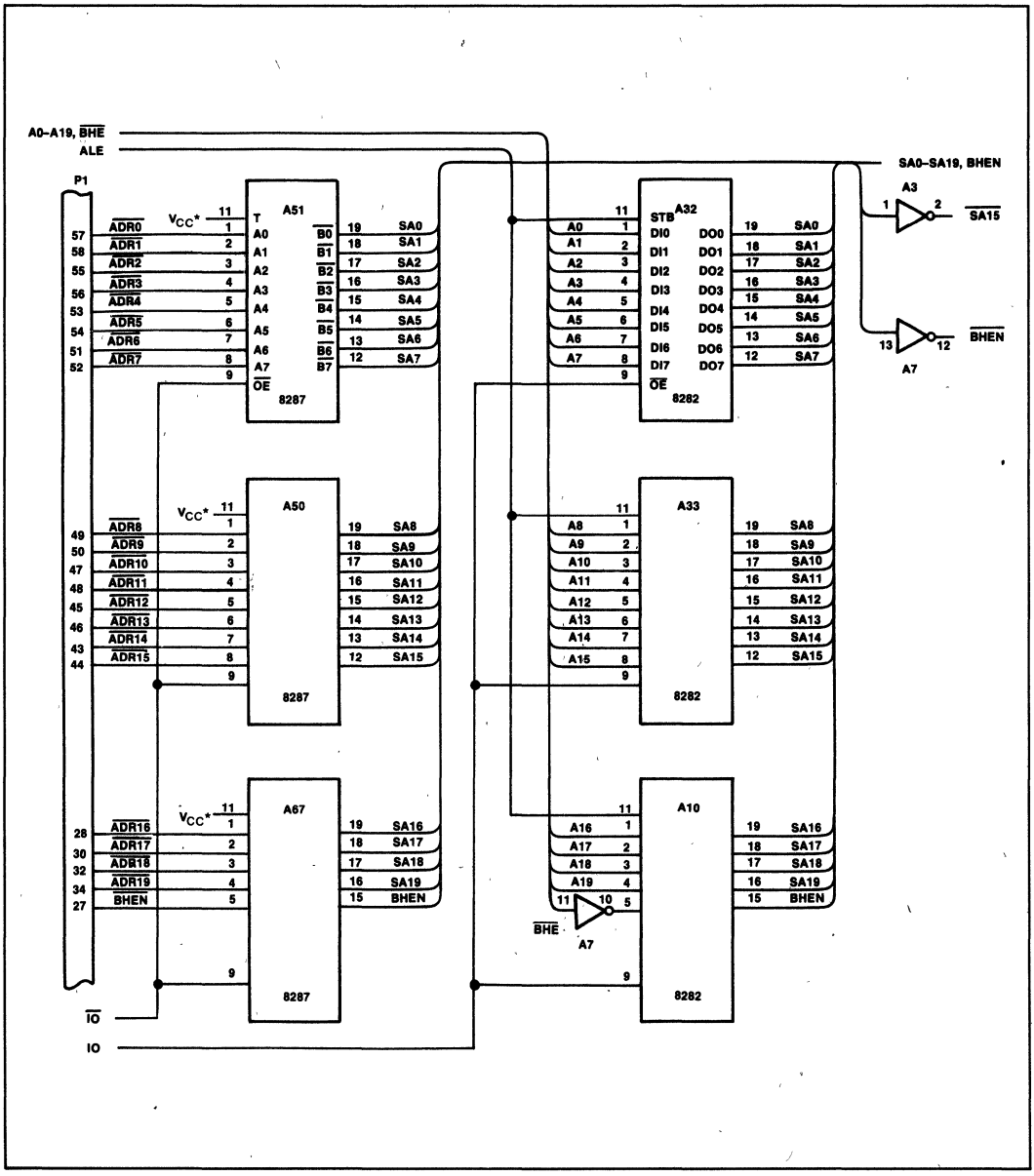


Figure 41. Multiplexer for Address Inputs to Dual-Port RAM

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.

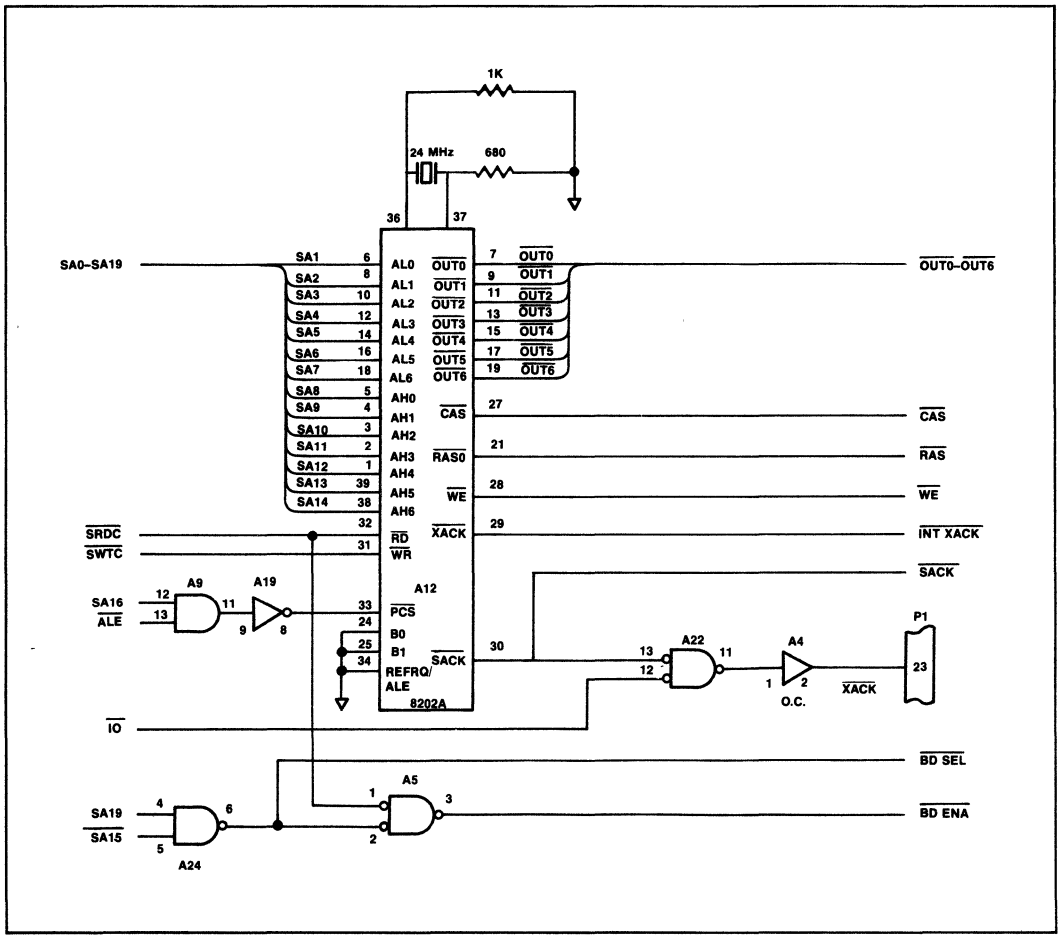


Figure 42. Dynamic RAM Controller

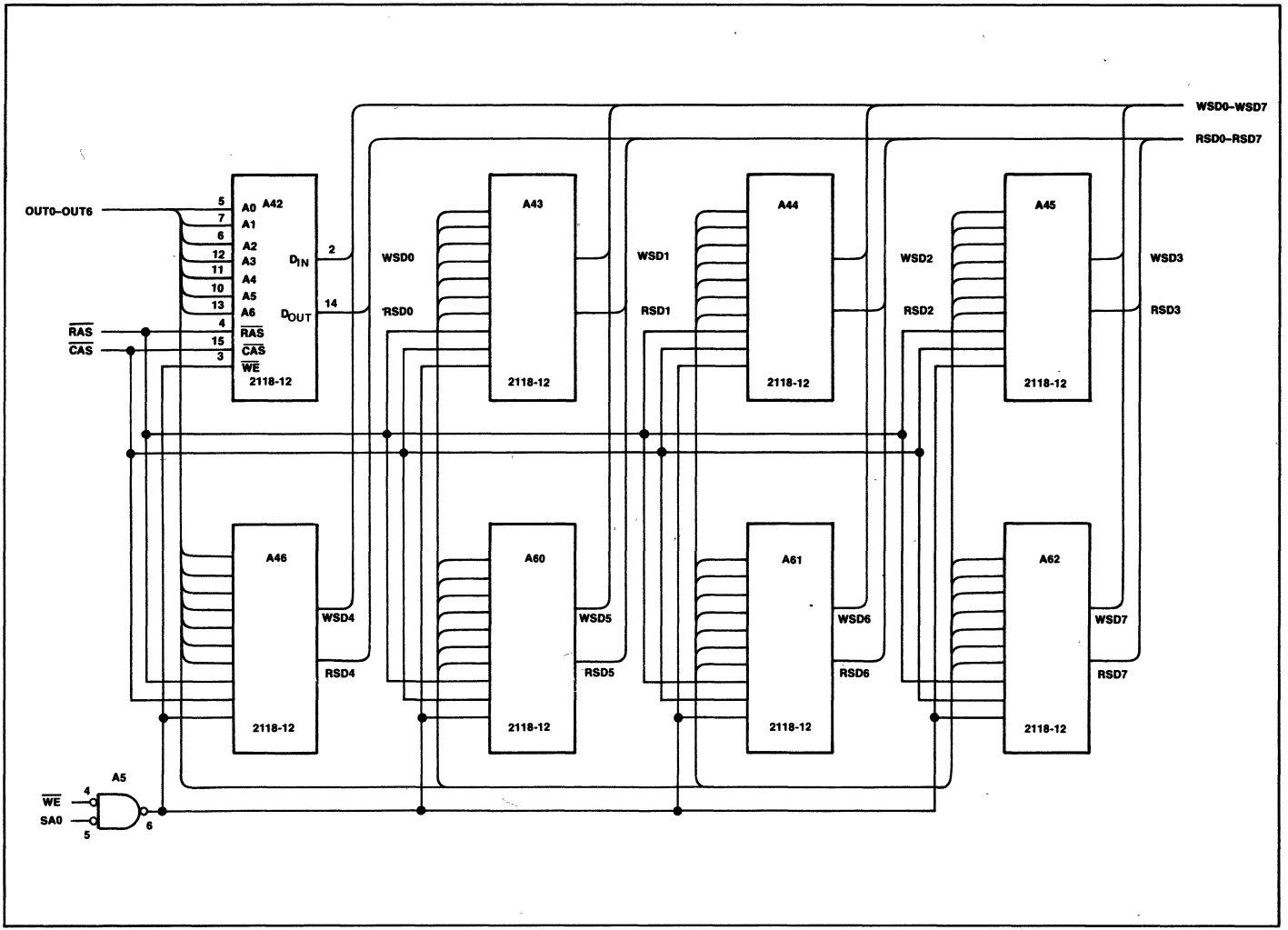


Figure 43. Dynamic RAM (Low Data Byte)

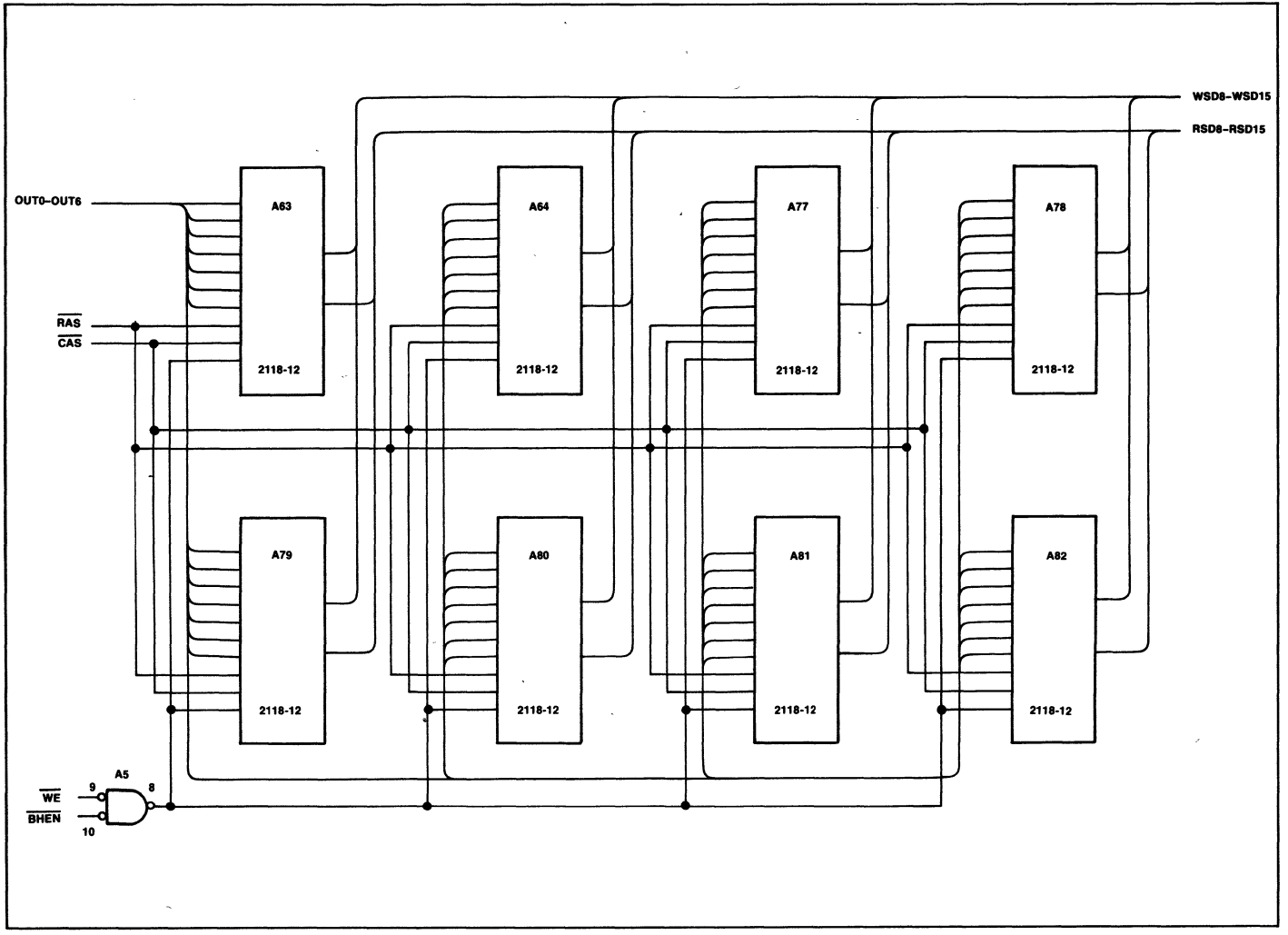


Figure 44. Dynamic RAM (High Data Byte)

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.

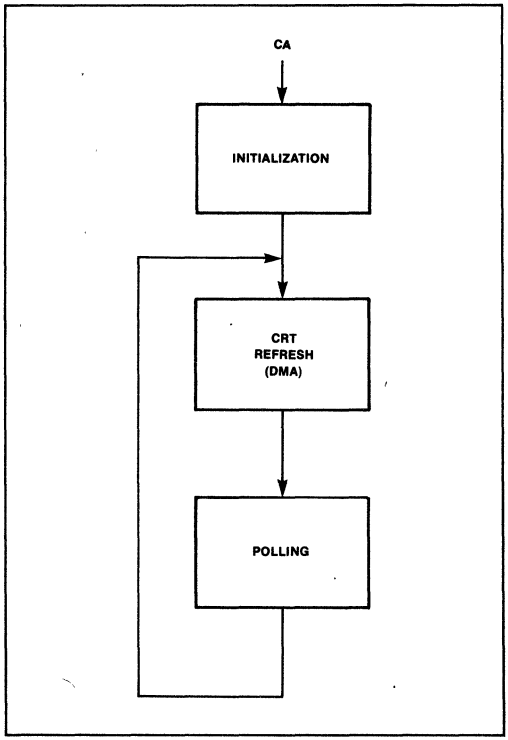


Figure 45. Channel Program Simplified Flowchart

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 8089 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).

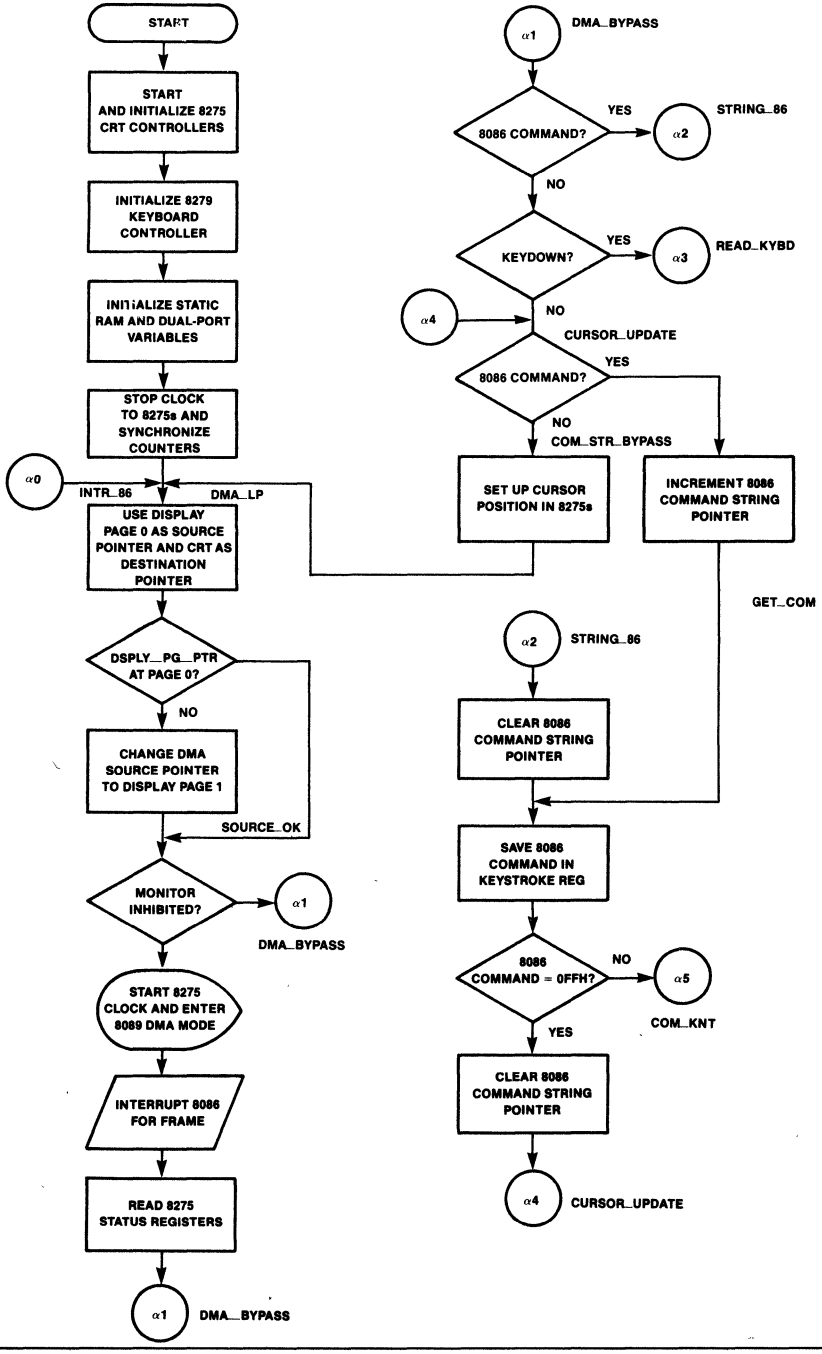
Termination on masked compare.

Word assembly/disassembly (8-bit word to/from 16-bit word).

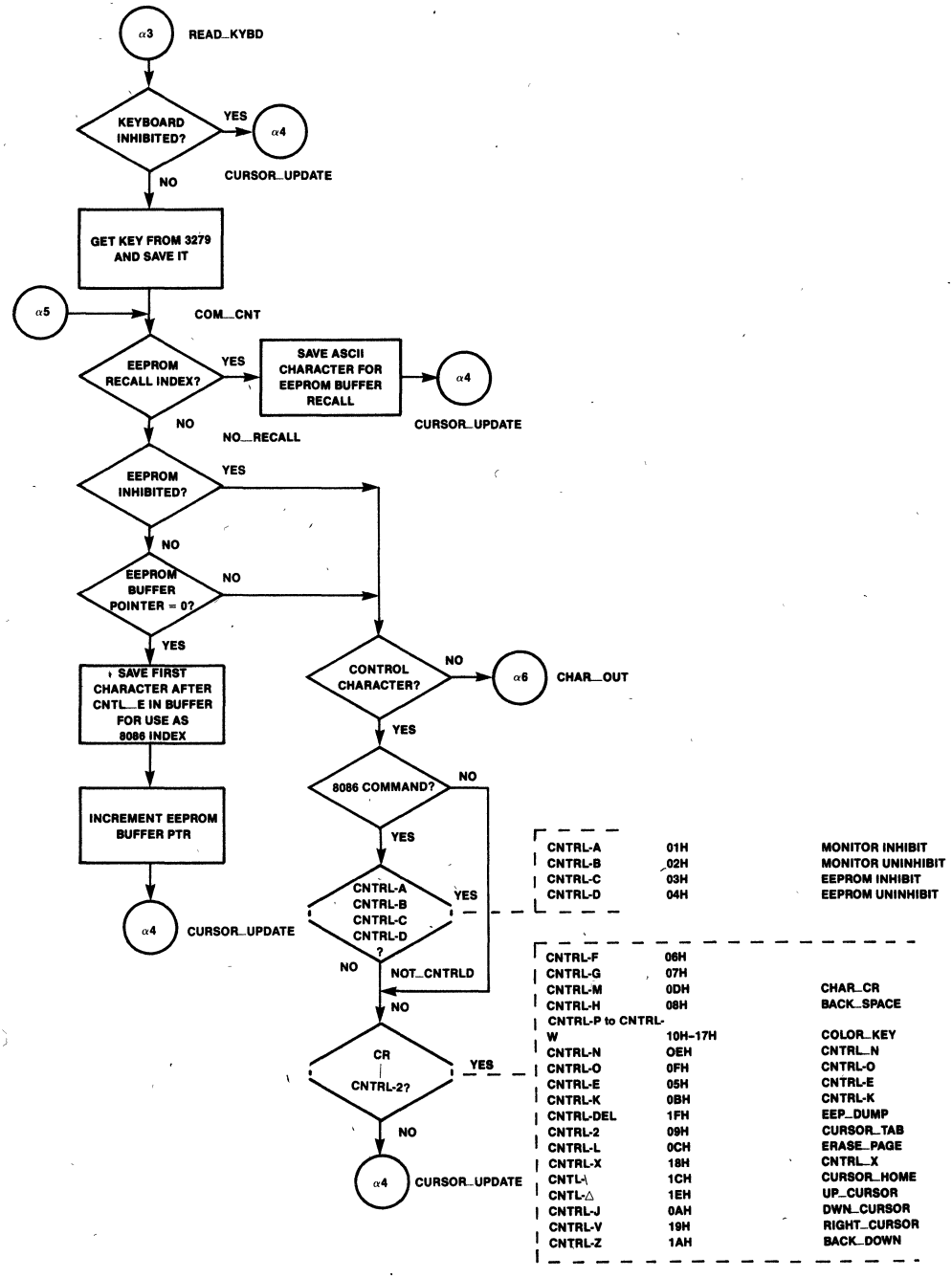
Memory-to-memory or I/O-to-I/O transfer.

Synchronization on source, destination, or neither.

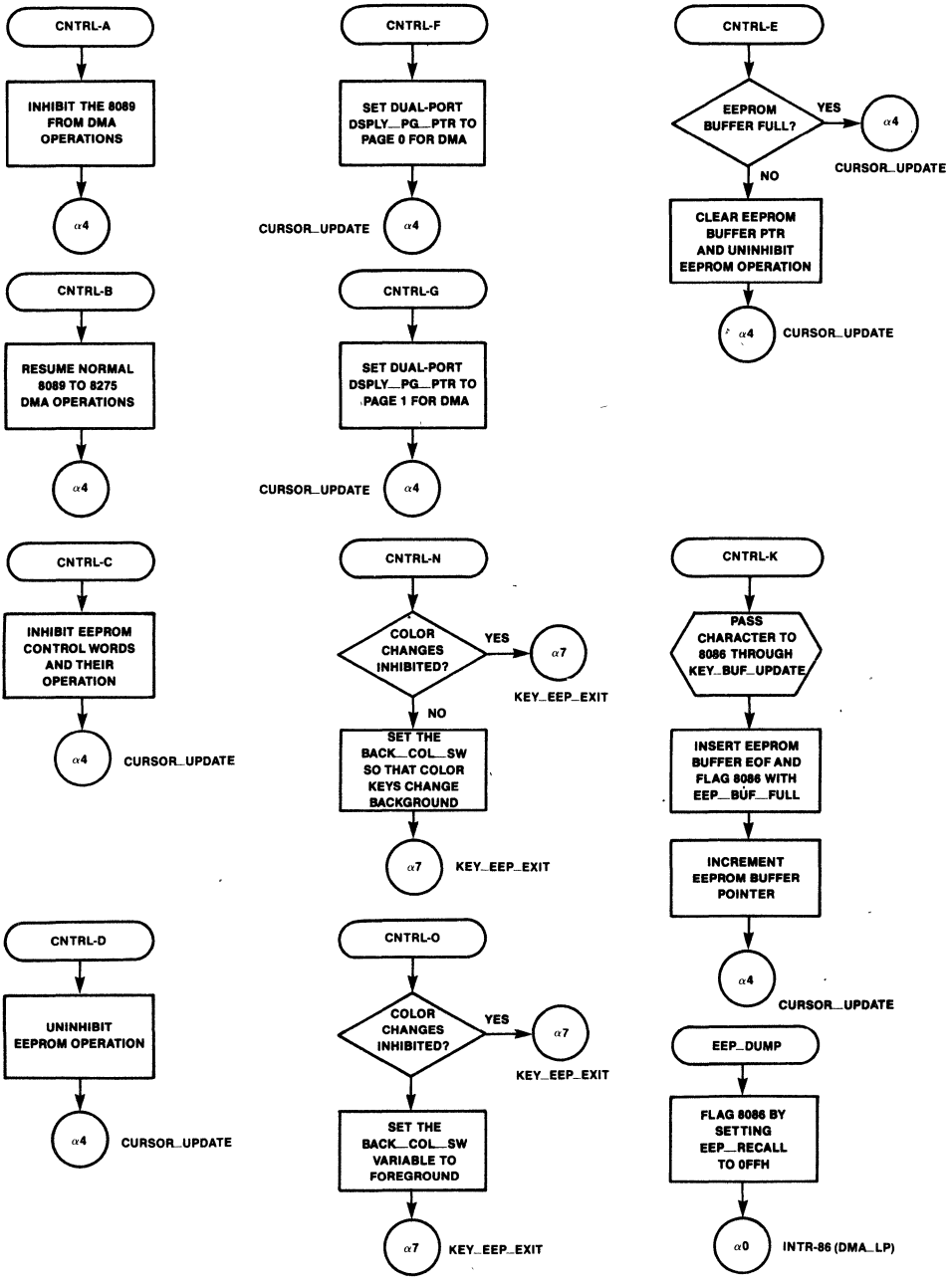
INITIALIZATION AND MAIN LOOP



KEY AND COMMAND DECODE

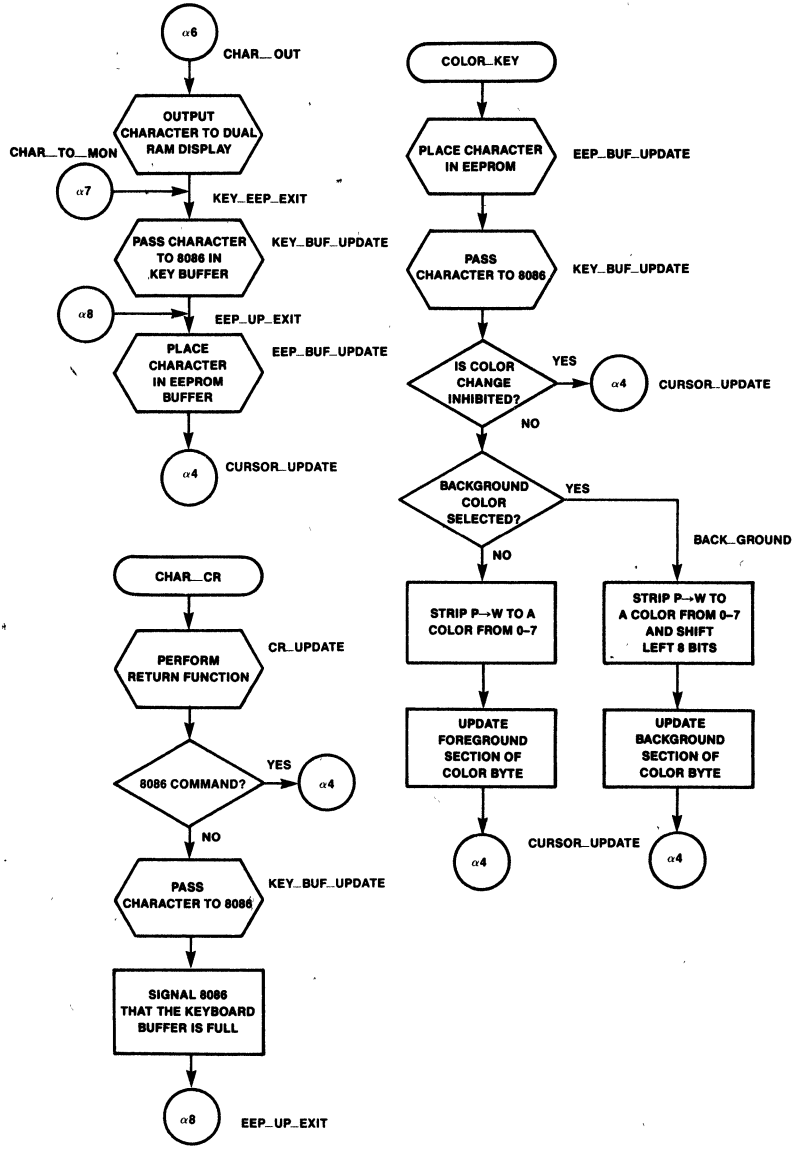


CONTROL KEY OPERATIONS

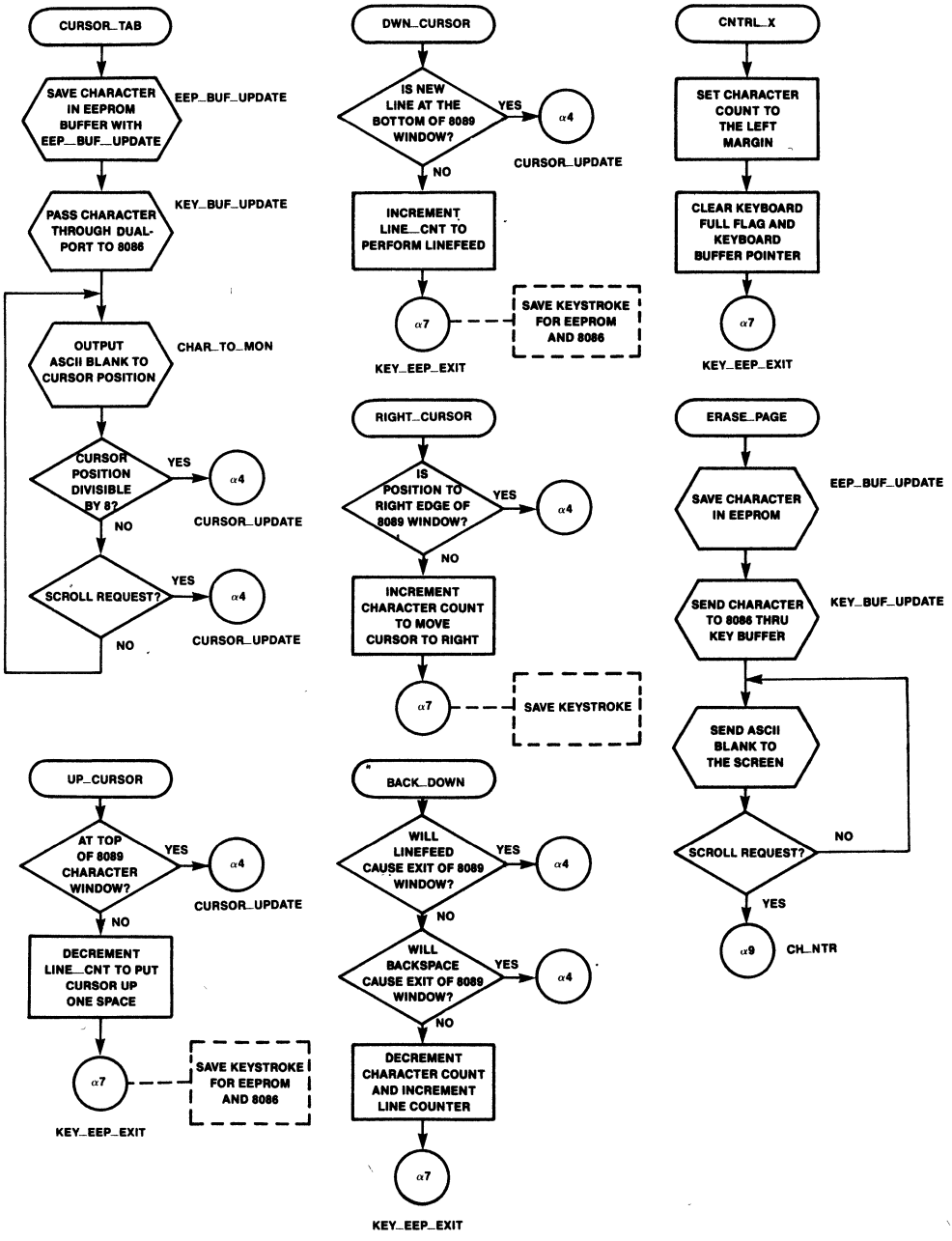


210355-001

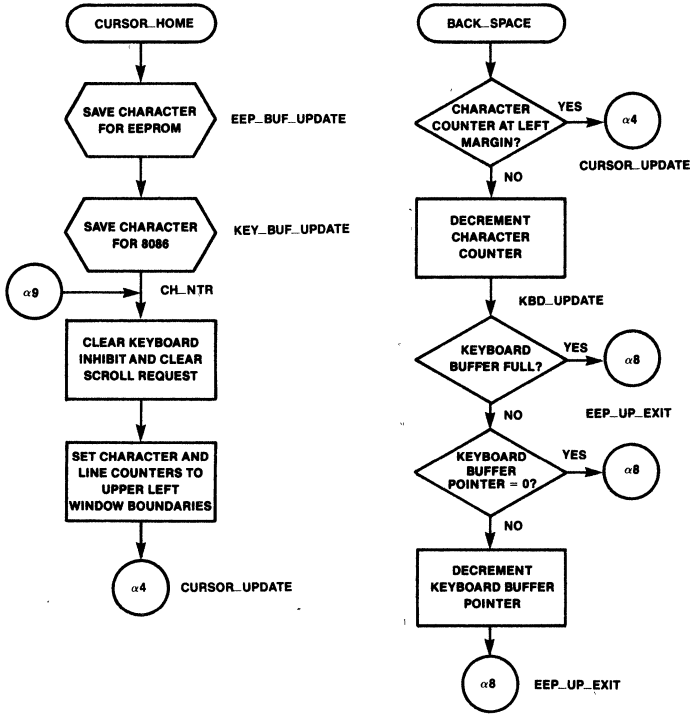
CONTROL KEY OPERATIONS



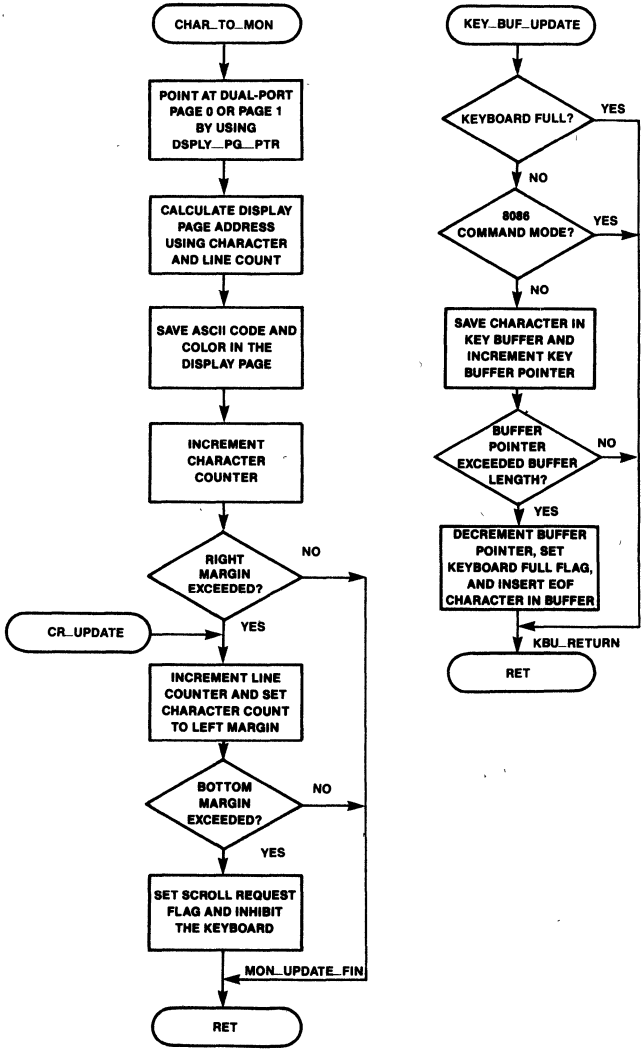
CONTROL KEY OPERATIONS



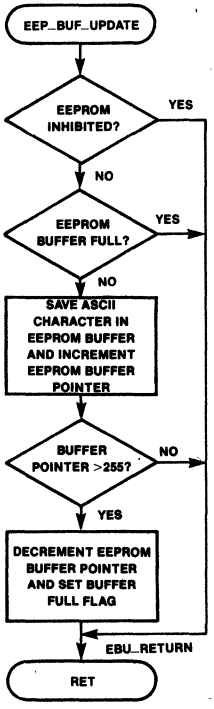
CONTROL KEY OPERATIONS



SUPPORT SUBROUTINES



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 GURSOR 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,OFFH
 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,OFFH
 642           MOVBI     [GC].EEP_INH,OFFH
 643           MOV       IX,[GB].EEP_BUF_PTR
 644           LPDI      GA,EEP_BUF
 645 ;
 646 ;   INSERT END OF FILE MARKER
 647 ;
 648           MOVBI     [GA+IX],OFFH
 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,OFFH      ;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     [GD].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-113

February 1981

Getting Started With the Numeric Data Processor

Bill Rash
Microcomputer Applications

207865-001

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 I/APX 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.

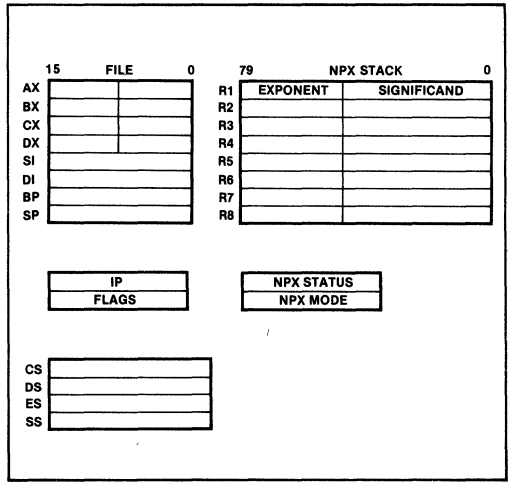


Figure 1. NDP Register Set for iAPX 86/20, 88/20

[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.

```
FILD        VALUE
FIADD       TABLE [BX]
FADD        ST,ST(1)
```

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.

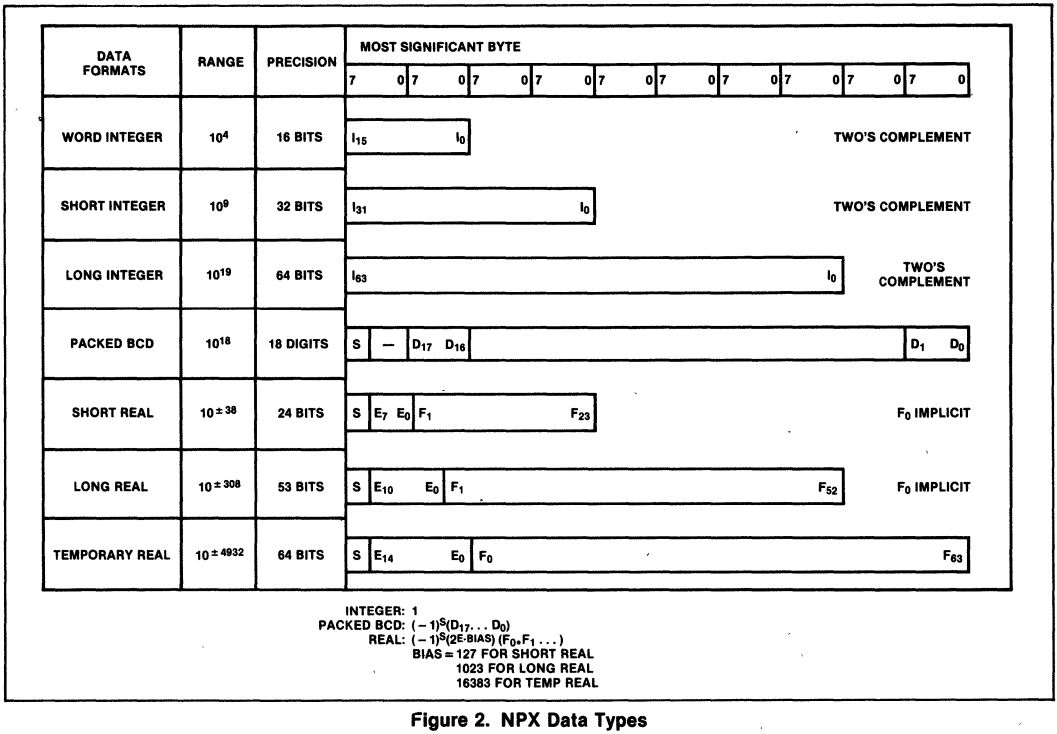


Figure 2. NPX Data Types

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.

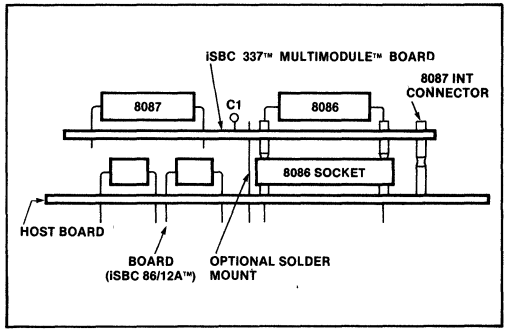


Figure 4. MULTIMODULE™ Math Mounting Scheme

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

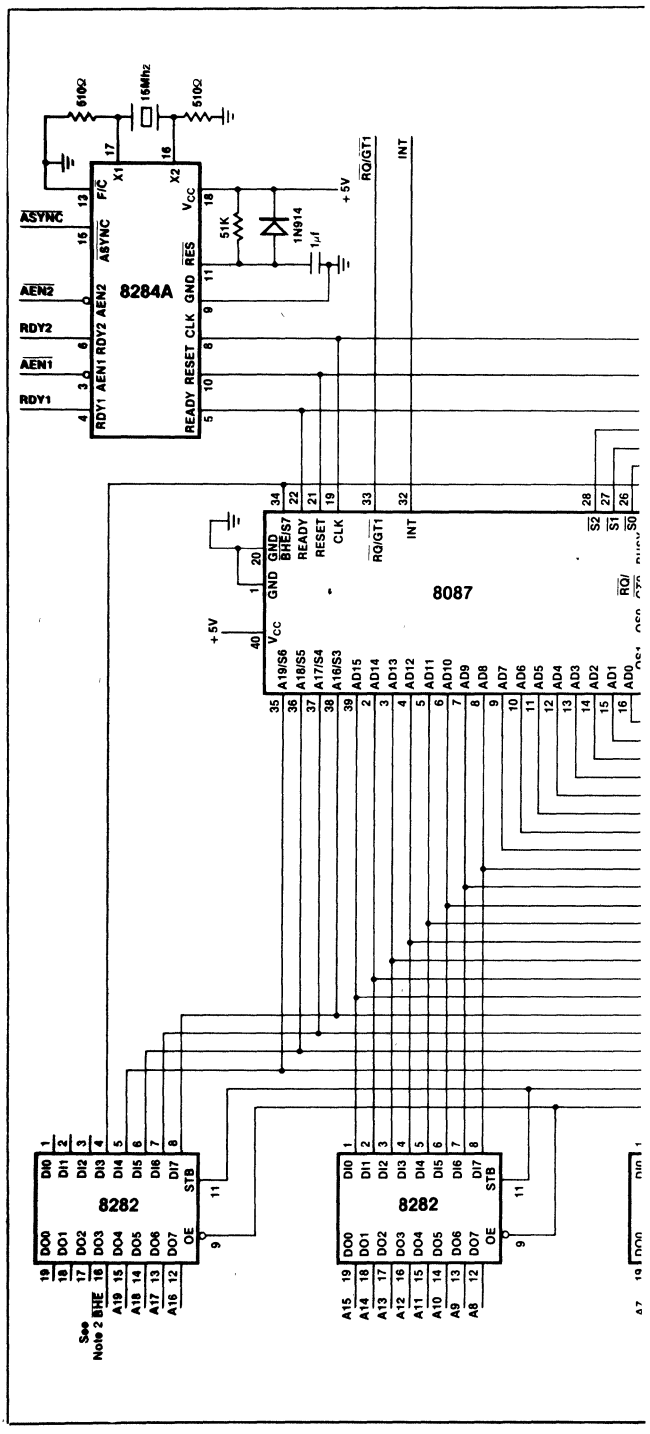
The 8087 can be designed into any 86/1X or 88/1X system operating in maximum mode. Such a system would be designated an 86/2X or 88/2X. Figure 5 shows the local bus interconnections for an iAPX 86/20 (or iAPX 88/20) system. The 8087 shares the maximum mode host's multiplexed address/data bus, status 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.

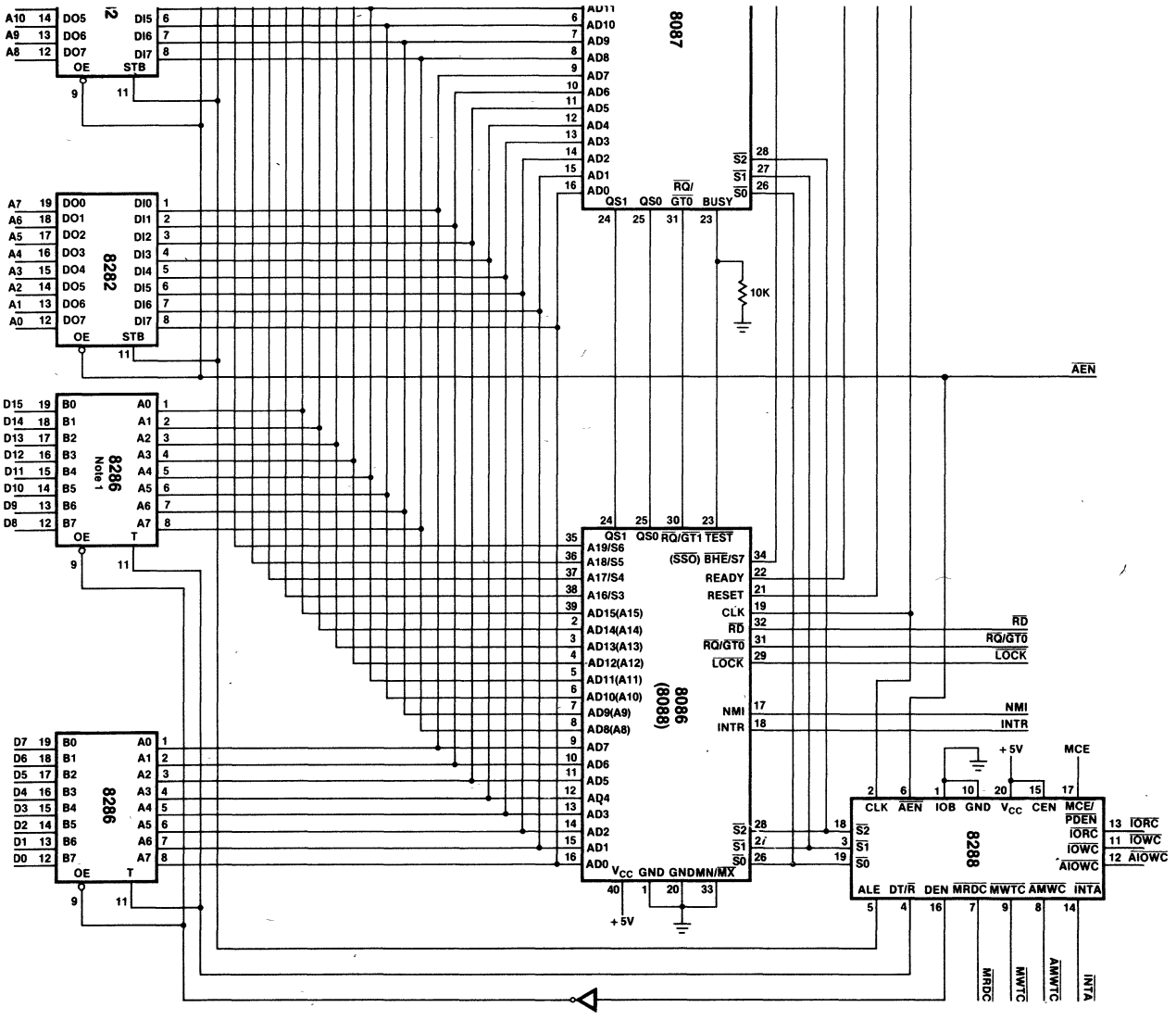
Adding the 8087 to your design has a minor effect on hardware timing. The 8087 has the exact same timing and equivalent DC and AC drive characteristics as a host or IOP on the local bus. All the local bus logic, such as clock, ready, and interface logic is shared.

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.





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.

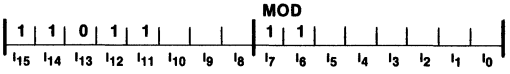
DECODING ESCAPE INSTRUCTIONS

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.

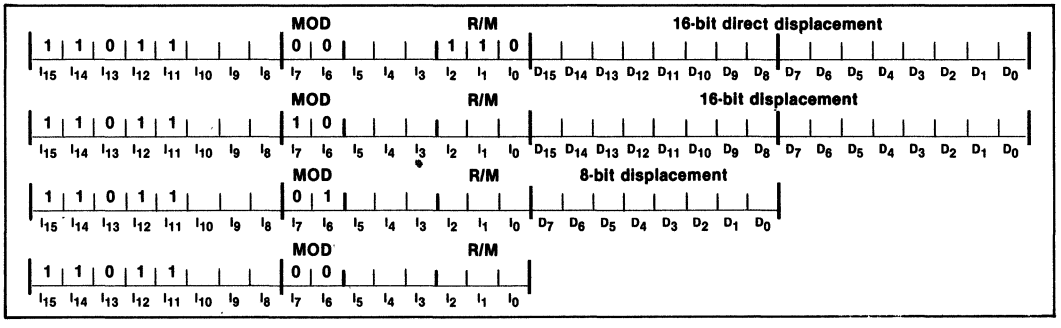


Figure 8. Memory Reference Escape Instruction Forms

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.

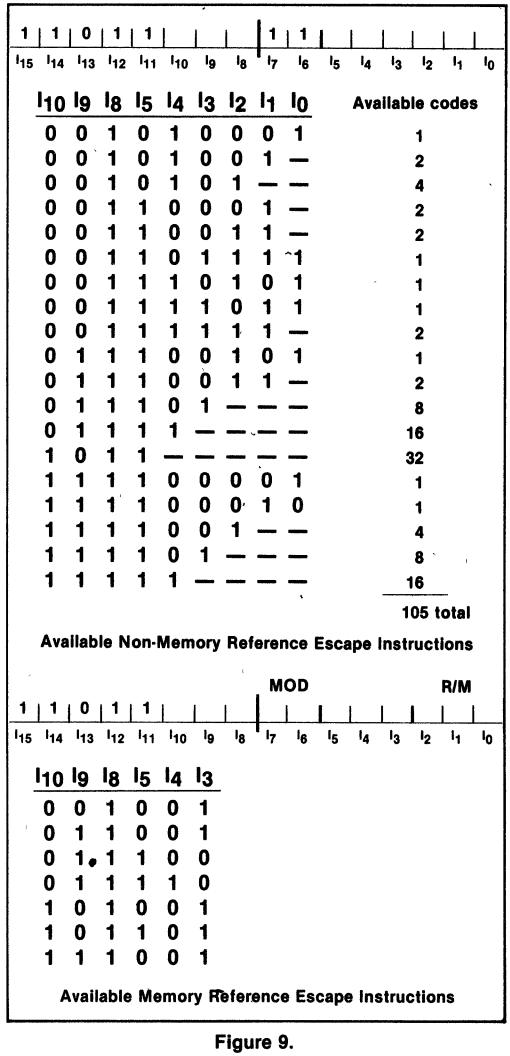


Figure 9.

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.

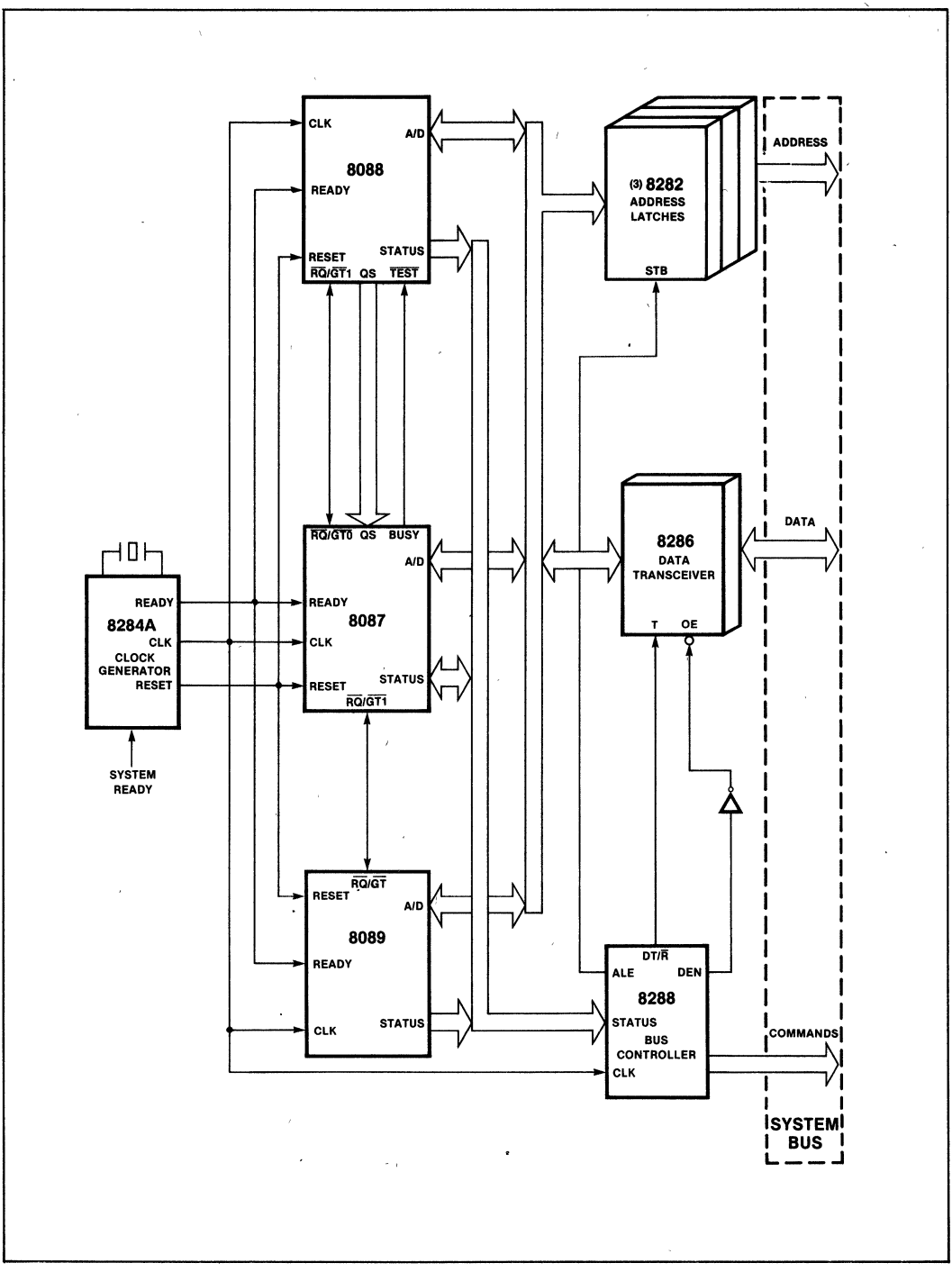


Figure 10. iAPX 88/21

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

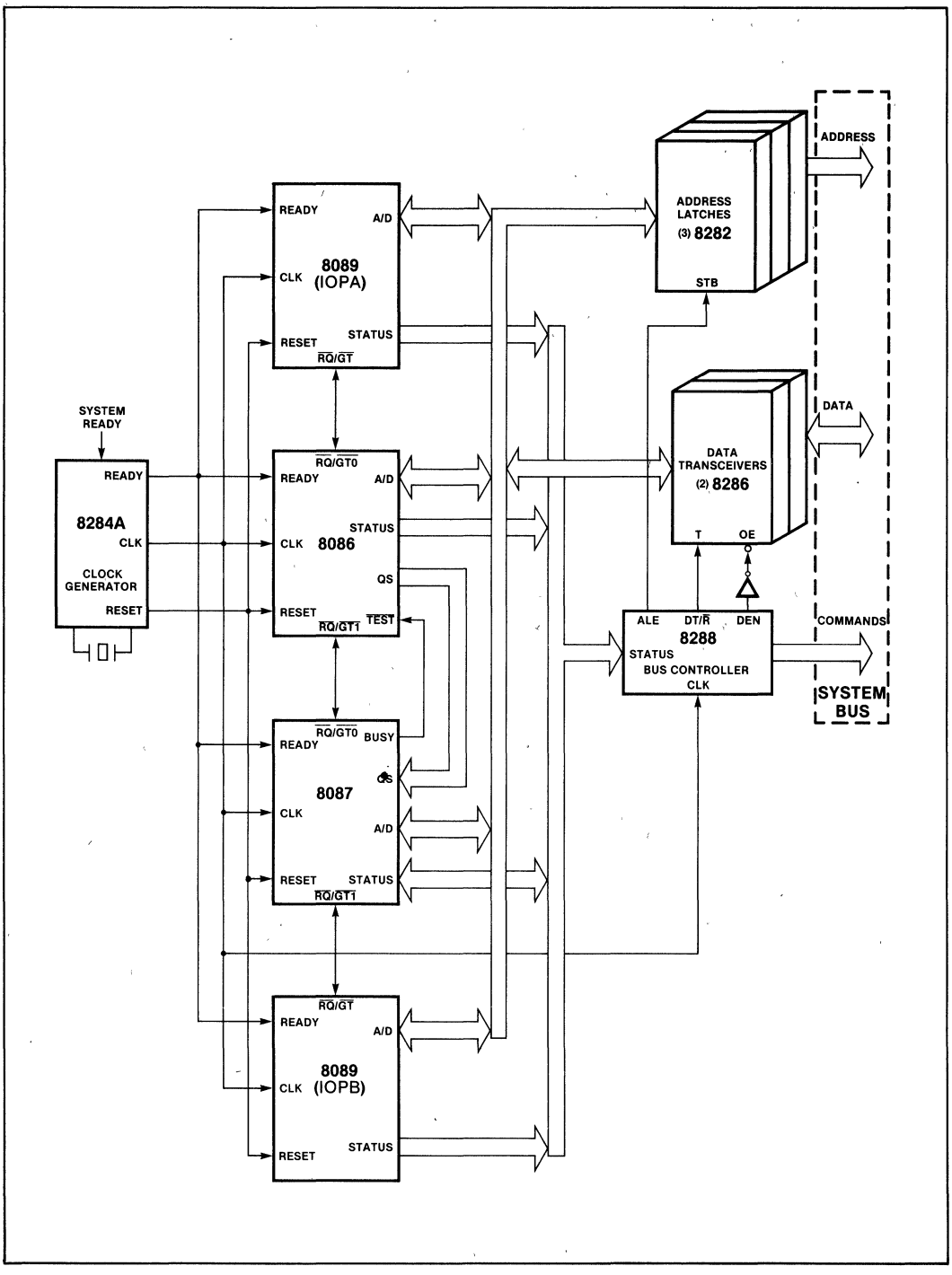


Figure 11. iAPX 86/22 System

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.

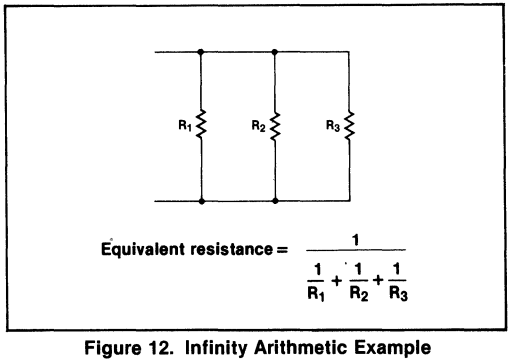


Figure 12. Infinity Arithmetic Example

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 page 19). 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.

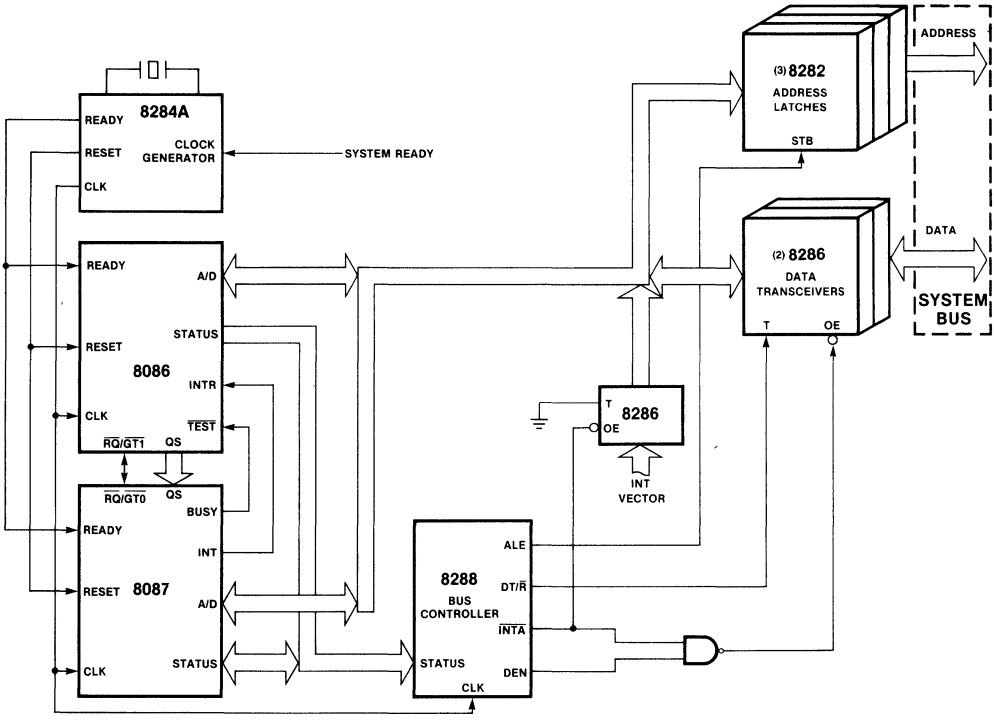


Figure 14. iAPX 86/20 With Numerics Interrupt Only

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:
    MOV   I, 1
    FILD  I

Case 2:
    MOV   AX, I
    FISTP I

Case 3:
    FILD  I
    FWAIT
    MOV   I,5

Case 4:
    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 (?)

119     stack           ends
```

```
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             fldl                            ; 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             fld1                            ; 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              f2xm1                           ; 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_overflow:
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 quarentee 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 <0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0>
  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
```

```
indefinite      dd      0FFC00000R              ; Indefinite special value

;
;         This subroutine calculates the sine or cosine of the angle, given in
;       radians.  The angle is in ST(0), the returned value will be in ST(0).
;       The result is accurate to within 7 units of the least significant three
;       bits of the NPX extended real format.  The PLM/86 definition is:
;
;       sine:   procedure (angle) real external;
;               declare angle real;
;               end sine;
;
;       cosine: procedure (angle) real external;
;               declare angle real;
;               end cosine;
;
;         Three stack registers are required.  The result of the function is
;       defined as follows for the following arguments:
;
;                       angle                                           result
;
;               valid or unnormal less than 2**62 in magnitude  correct value
;               zero                                            0 or 1
;               denormal                                        correct denormal
;               valid or unnormal greater than 2**62            indefinite
;               infinity                                        indefinite
;               NAN                                             NAN
;               empty                                           empty

;
;         This function is based on the NPX fptan instruction.  The fptan
;       instruction will only work with an angle of from 0 to PI/4.  With this
;       instruction, the sine or cosine of angles from 0 to PI/4 can be accurately
;       calculated.  The technique used by this routine can calculate a general
;       sine or cosine by using one of four possible operations:
;
;                       Let R = |angle mod PI/4|
;                           S = -1 or 1, according to the sign of the angle
;
;       1) sin(R)       2) cos(R)       3) sin(PI/4-R)  4) cos(PI/4-R)
;
;         The choice of the relation and the sign of the result follows the
;       decision table shown below based on the octant the angle falls in:
;
;               octant          sine            cosine
;
;                 0              S*1              2
;                 1              S*4              3
;                 2              S*2             -1*1
;                 3              S*3             -1*4
;                 4             -S*1             -1*2
;                 5             -S*4             -1*3
;                 6             -S*2                1
;                 7             -S*3                4
;


;
;         Angle to sine function is a zero or unnormal.
;
sine_zero_unnormal:

        fstp    st(1)                           ; Remove PI/4
        jnz     enter_sine_normalize            ; Jump if angle is unnormal
;
;         Angle is a zero.
;
        pop     bp                              ; Return the zero as the result
        ret
;
;         Angle is an unnormal.
;
enter_sine_normalize:

```

```
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-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.

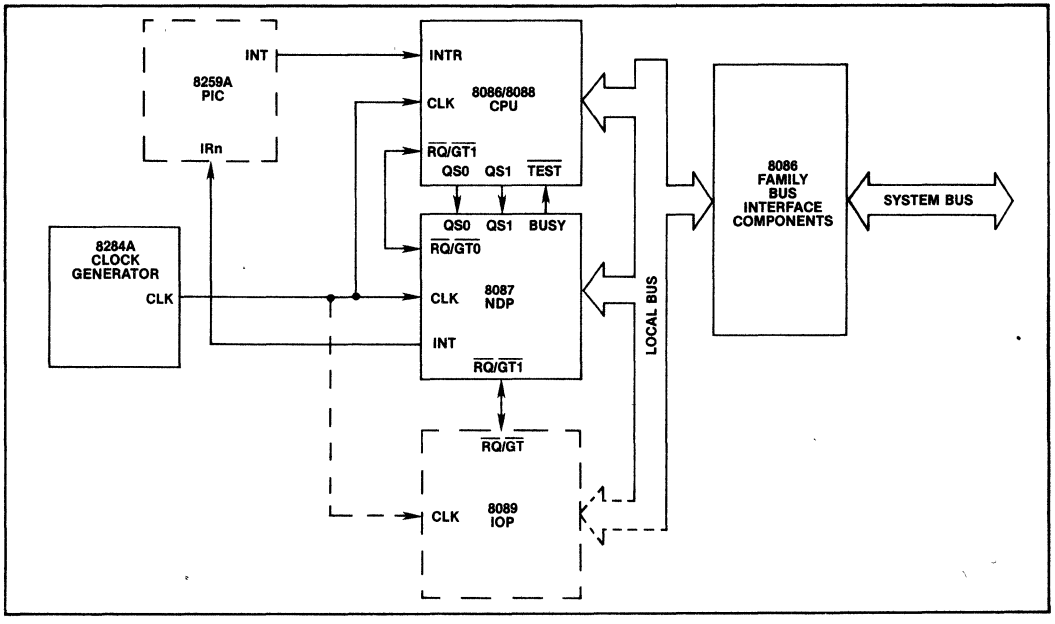


Figure 1. Typical 86/20 System

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

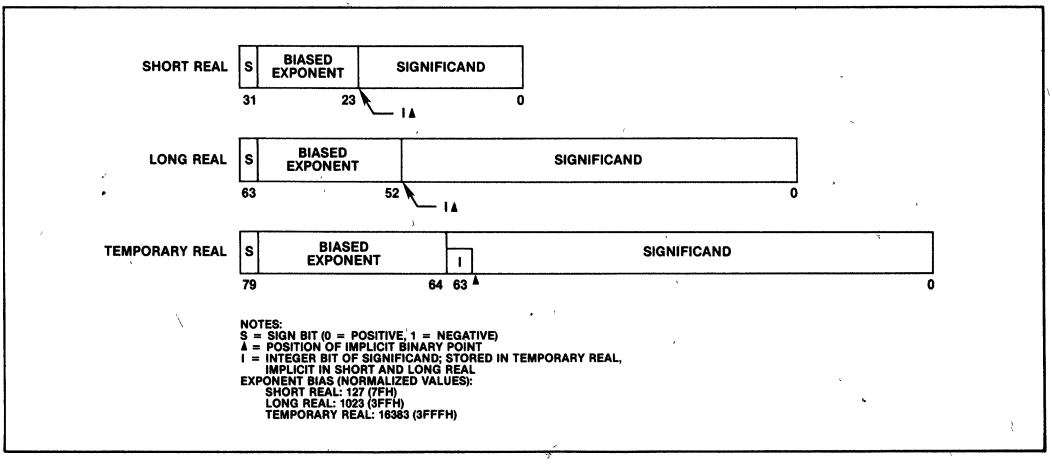


Figure 2. Data Formats

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

```
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 costl := 20.0 * totcnt
            else if totcnt < 5000 then costl := 17.5 * totcnt
            else costl := 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 + costl)/(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];
            costl := setupcost+optwz[j+1];
            if totcnt <= 500 then costl := costl + 20.0*totcnt
            else if totcnt <= 5000 then costl := costl + 17.5*totcnt
            else costl := costl + 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 := costl + 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.

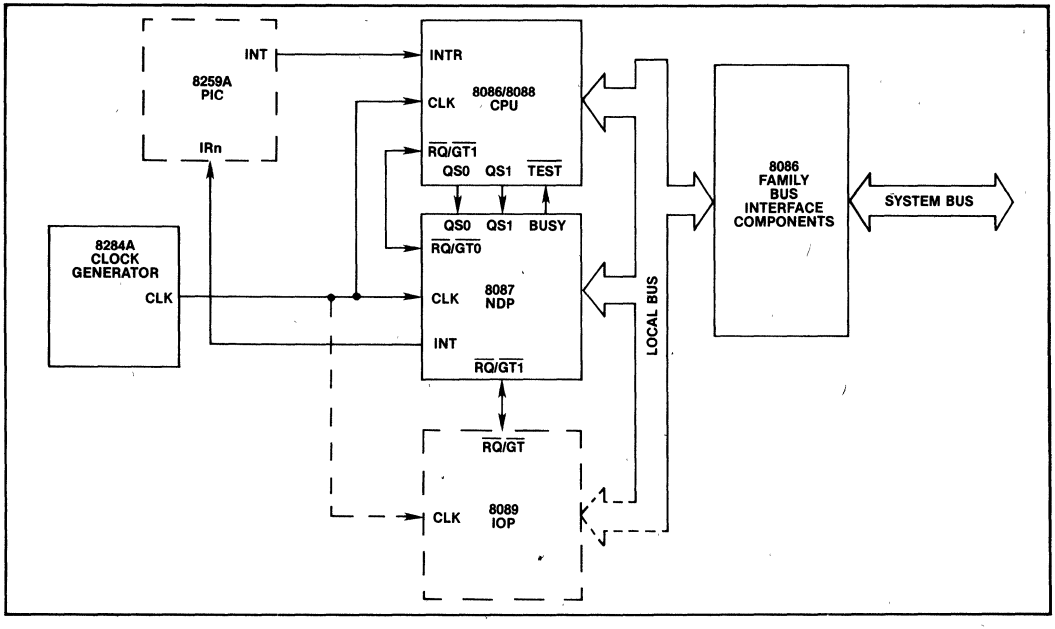


Figure 1. Example 86/20 System

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.

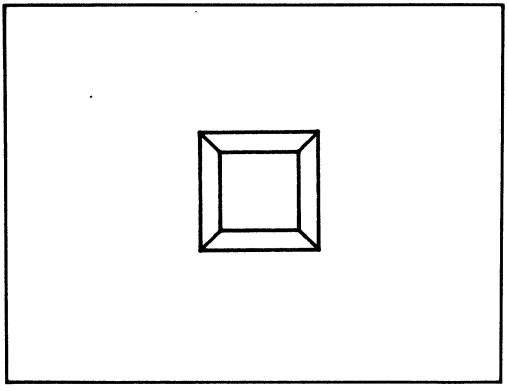


Figure 2. 2 × 2 × 2 Cube Centered at (0,0,0) Viewed from (0,0,10)

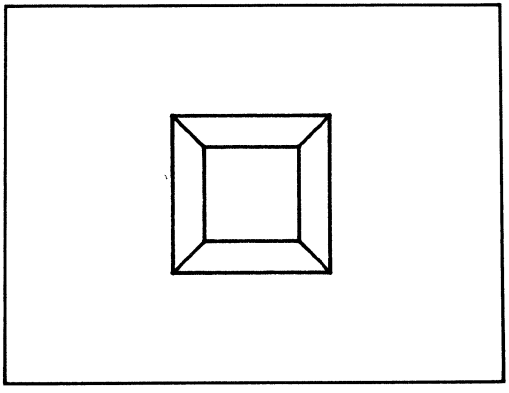


Figure 3. Same Cube and Viewpoint, +2 Z Translation

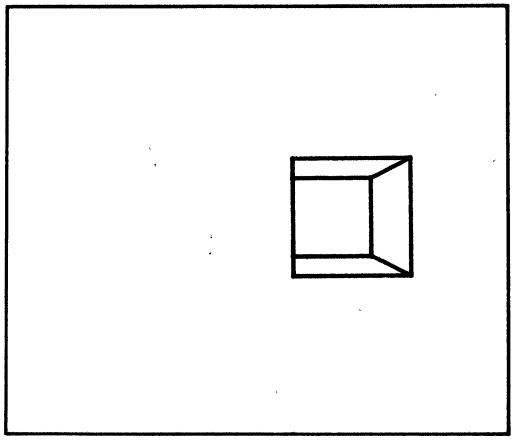


Figure 4. Same Cube, Viewpoint, +2 X Translate

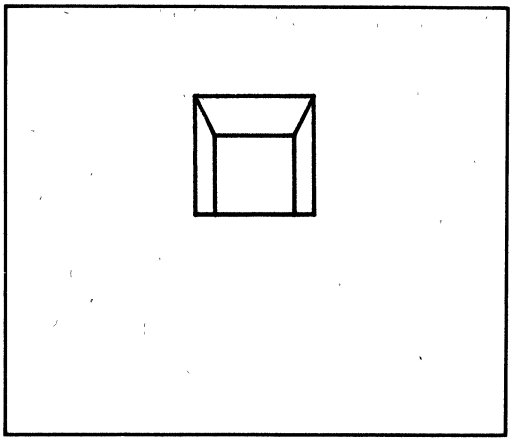


Figure 5. Same Cube, Viewpoint, +2 Y Translate

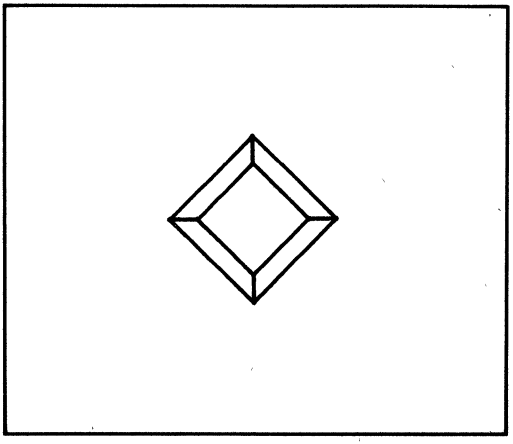


Figure 6. Same Cube, Viewpoint, 45 Degree Rotation About Z

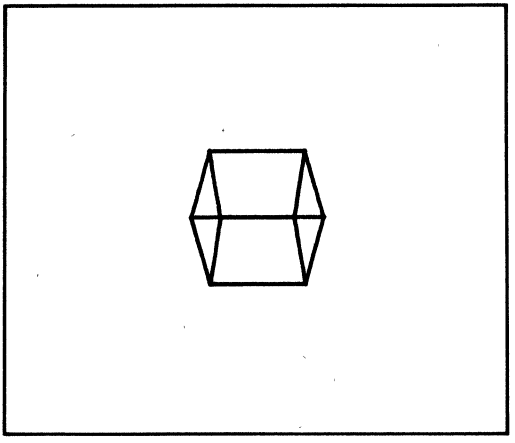


Figure 7. Same Cube, Viewpoint, 45 Degree Rotation About X

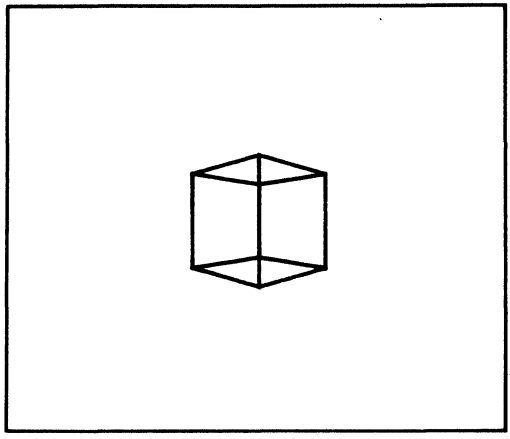


Figure 8. Same Cube, Viewpoint, 45 degree Rotation About Y

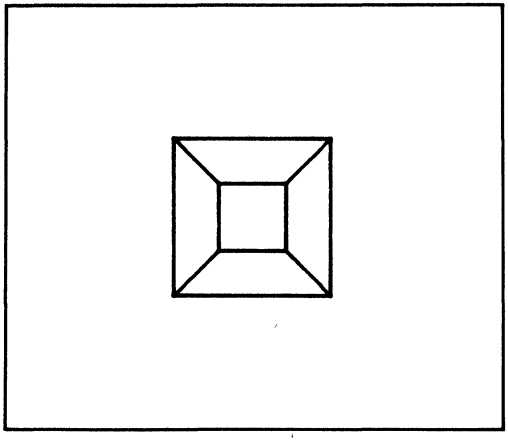


Figure 9. Same Cube, Viewpont 2 x Scale of Z

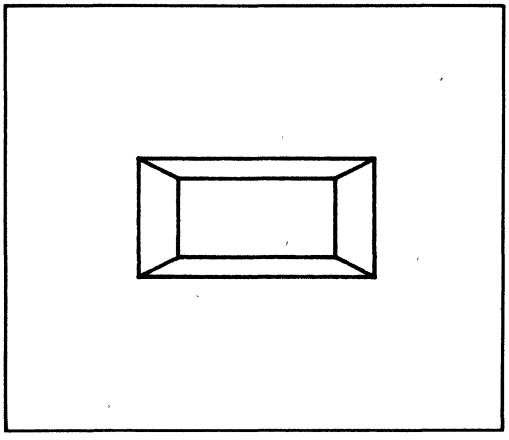


Figure 10. Same Cube, Viewpoint, 2 x Scale of X

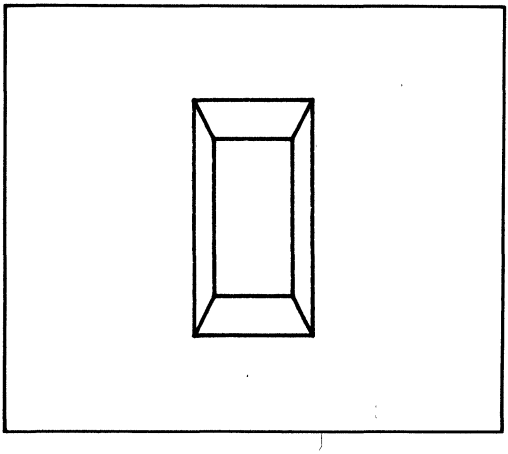


Figure 11. Same Cube, Viewpoint, 2 x Scale of Y

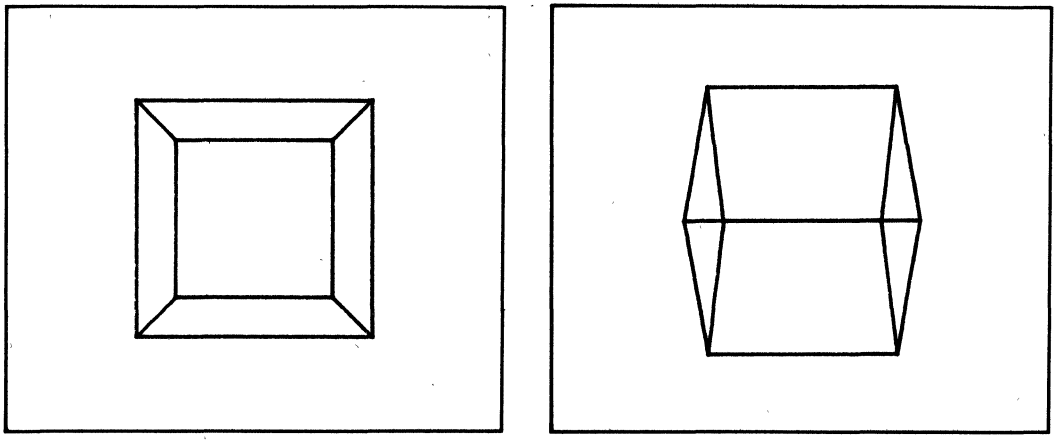


Figure 12. 2 x 2 x 2 Cube Centered at (0,0,0) Viewed from (0,0,10) Then from (10,10,0)

- **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.

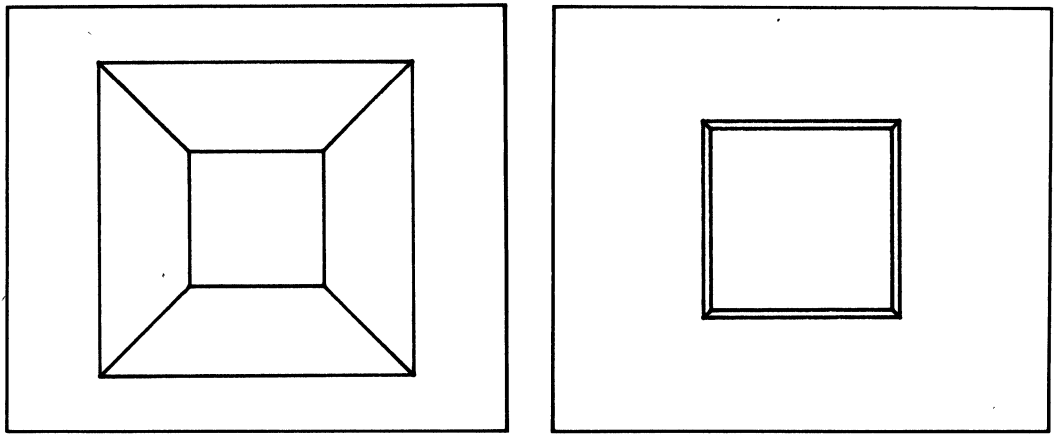


Figure 13. Example Cube Shown with Exaggerated Perspective, then with Subdued Perspective

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.

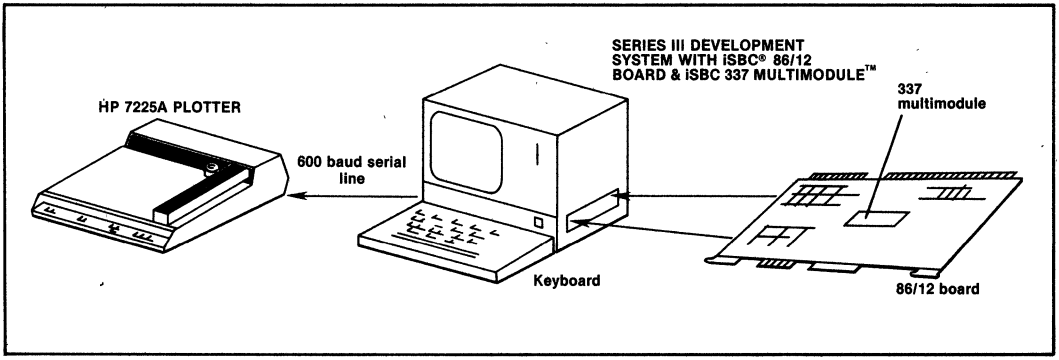


Figure 14. Computer System Used in This Graphics Implementation

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-

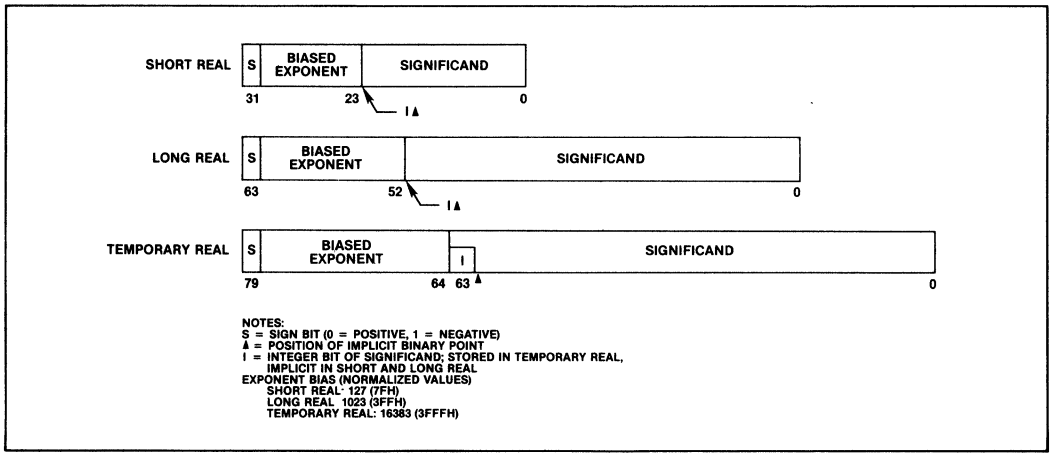


Figure 15. Floating Point Data Formats

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.

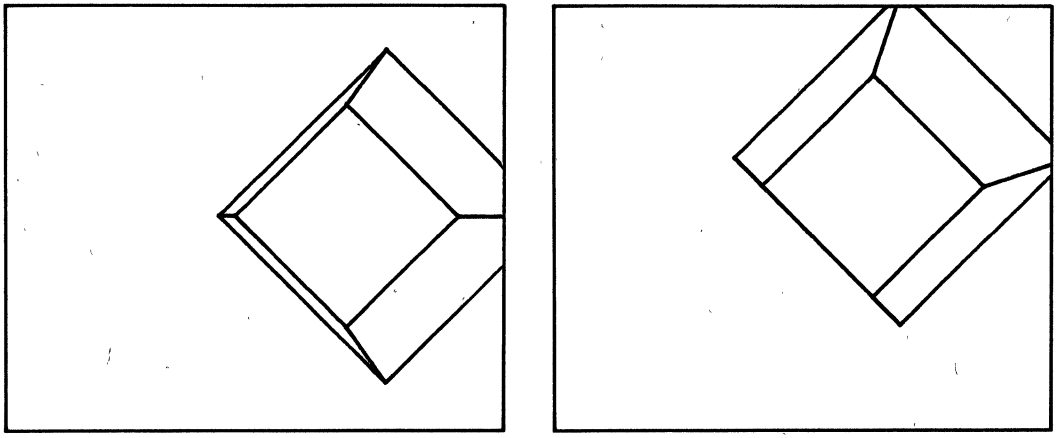


Figure 16. Example Cube Viewed from (0,0,10) First Rotated then Translated then Translated then Rotated.

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.

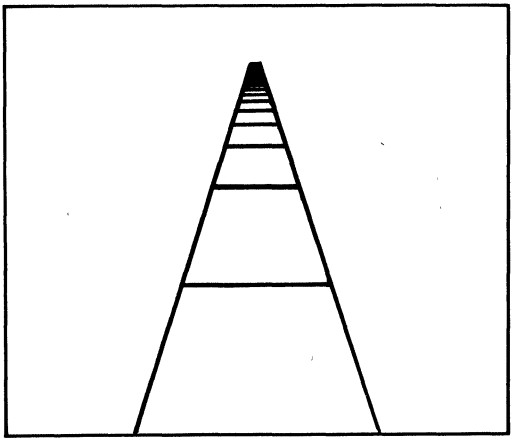


Figure 17. Two Rails, Vanishing into the Distance

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.

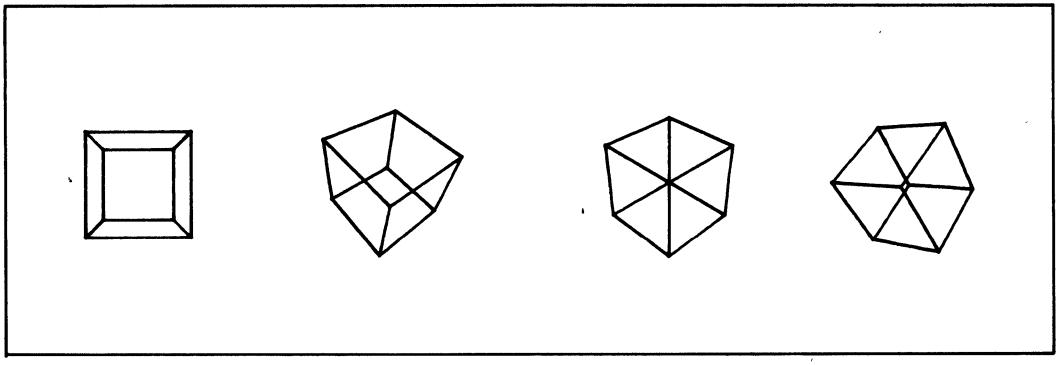


Figure 18. Demonstration Picture 1

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.

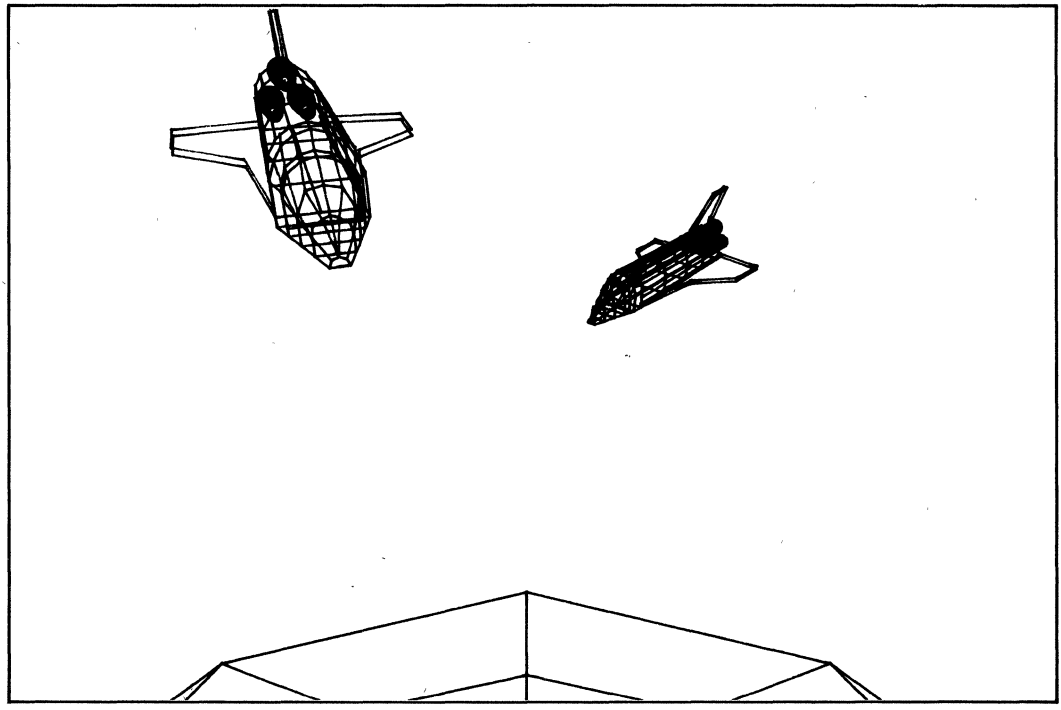


Figure 19. Demonstration Picture 2

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

```
      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 get1
      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(mpl,mp2,mpr)
      c
      c               subroutine mmult4 multiplies the mpl 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(mpl,mp2,mpr)
 2            real*8 mpl(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 + mpl(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 mmultl(mpl,mp2,mpr)
      c
      c               subroutine mmultl multiplies the mpl 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 mpl vector may be used
      c               as the destination of the result
      c
20            subroutine mmultl(mpl,mp2,mpr)
21            real*8 mpl(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 + mpl(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' /
```

intel®

APPLICATION NOTE

AP-186

March 1983

Introduction to the 80186 Microprocessor

Ken Shoemaker
Applications Engineer

ORDER NUMBER: 210973-003

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.

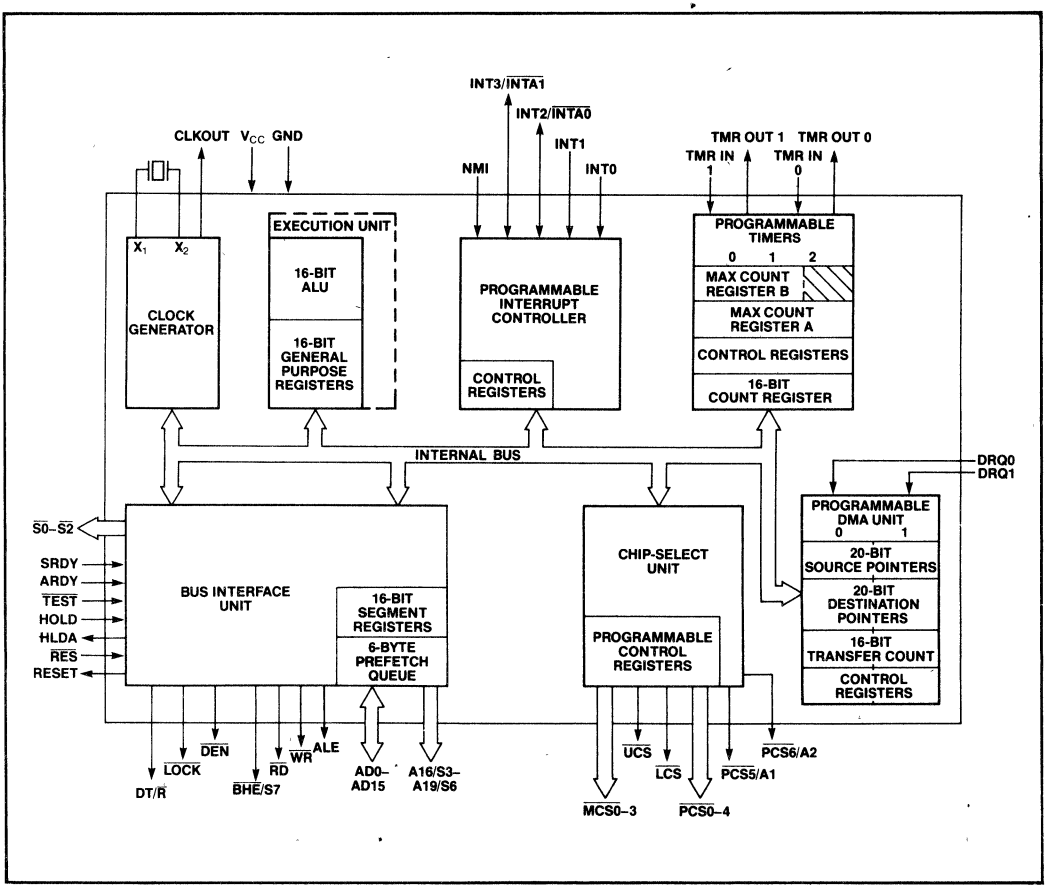


Figure 1. 80186 Block Diagram

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.

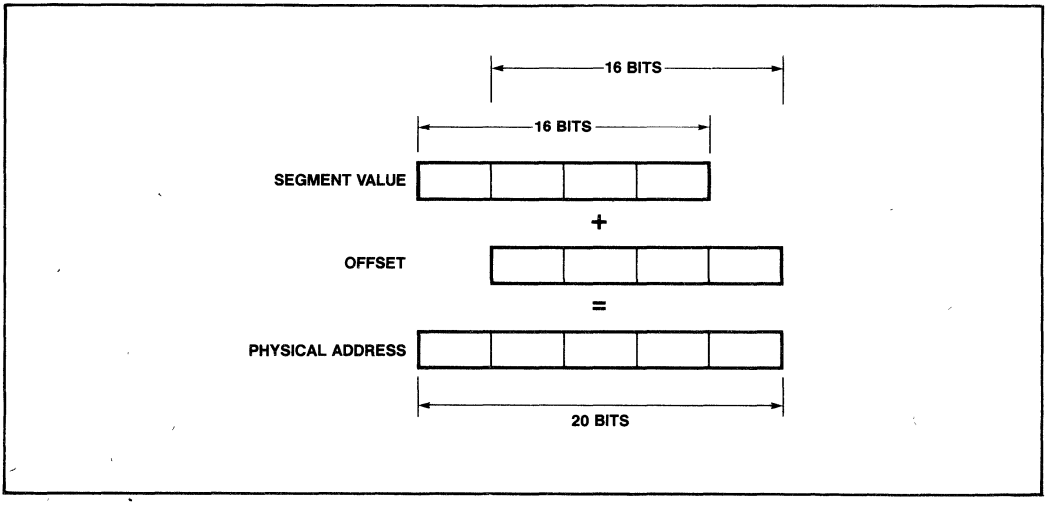


Figure 2. Physical Address Generation in the 80186

210973-003

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.

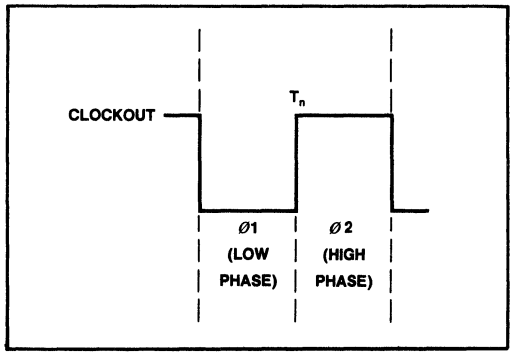


Figure 3. T-state in the 80186

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

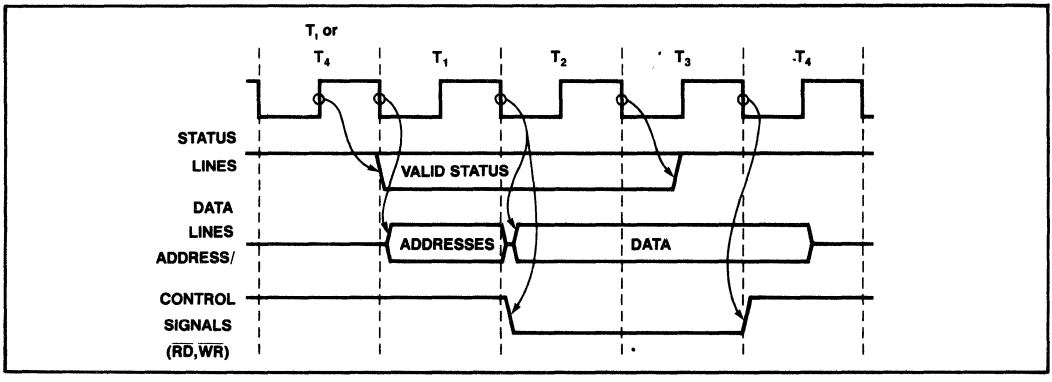


Figure 4. Example Bus Cycle of the 80186

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).

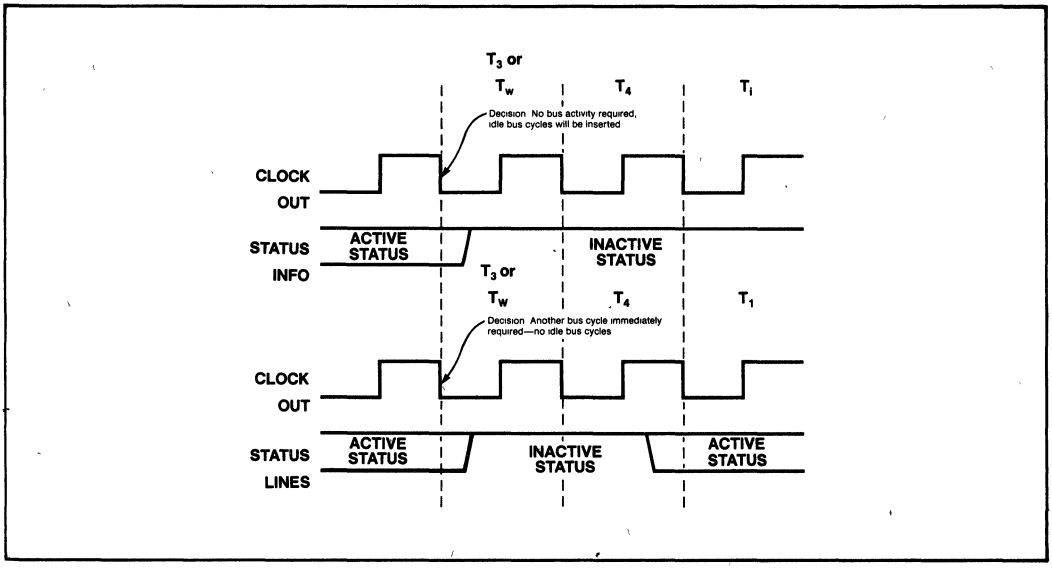
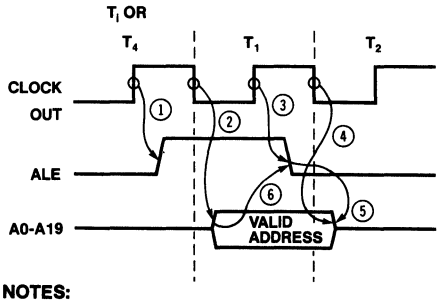


Figure 5. Active-Inactive Status Transitions in the 80186



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:

$$t_{CLAV}\ (44ns) + t_{IVOV}\ (30ns) = 74ns$$

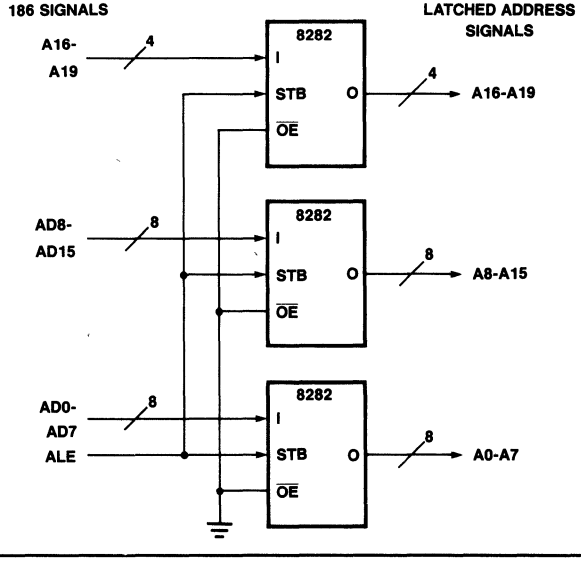


Figure 7. Demultiplexing the Address Bus of the 80186

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

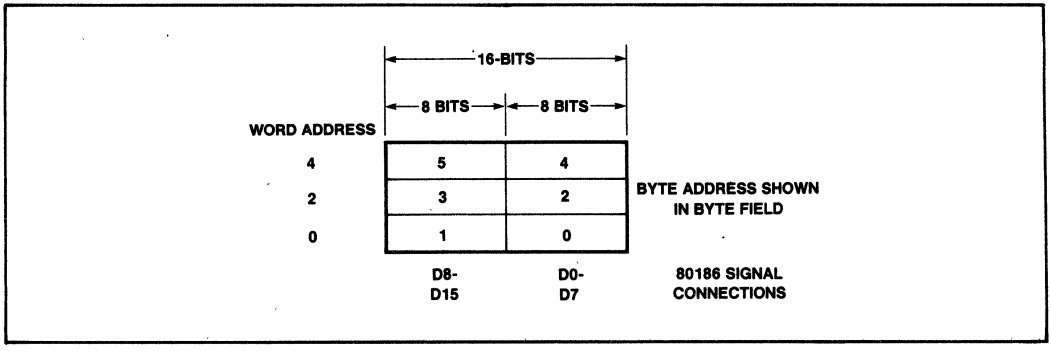


Figure 8. Physical Memory Byte/Word Addressing in the 80186

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.

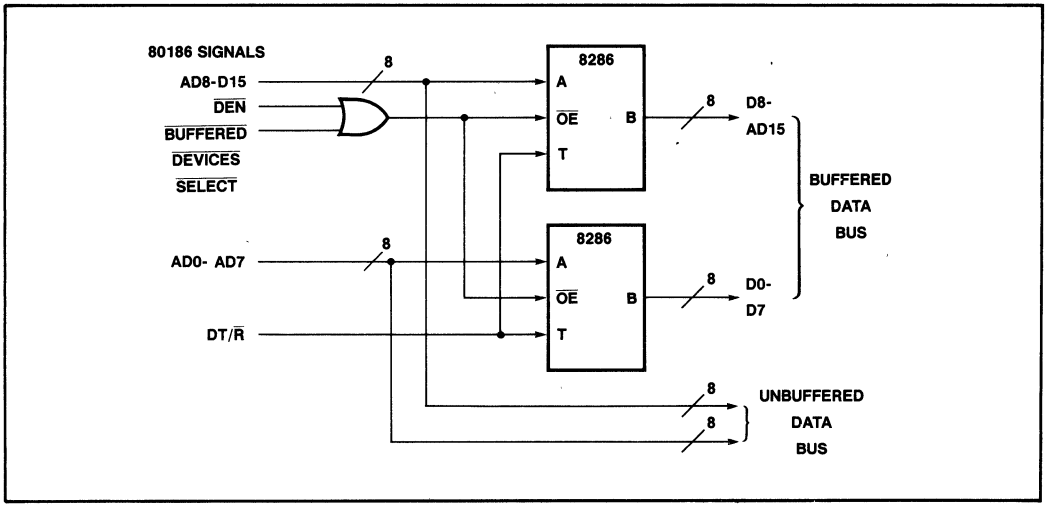
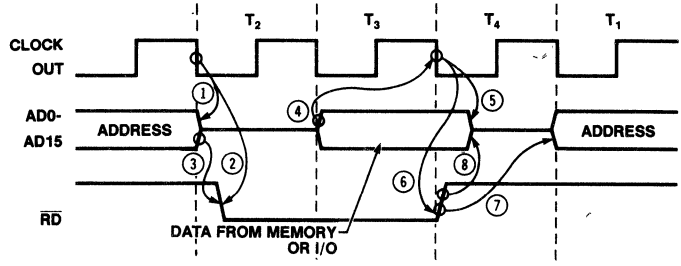


Figure 9. Example 80186 Buffered/Unbuffered Data Bus

210973-003



1. t_{CLAZ}: Clock low until address float=35 ns max
2. t_{CLRL}: Clock low until $\overline{RD}$ active = 70 ns max
3. t_{AZRL}: Address float until $\overline{RD}$ 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 $\overline{RD}$ high = 10 ns min
7. t_{RHAV}: $\overline{RD}$ high until addresses valid = 85 ns min
8. t_{RHDX}: Read high until data invalid (data input hold from $\overline{RD}$) = 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 $\overline{RD}$, $\overline{WR}$, $\overline{LOCK}$ and $\overline{TEST}$. In addition, the 80186 provides the status signals $\overline{S0}$-$\overline{S2}$ and S6 from which all other required bus control signals can be generated.

3.1.6.1 $\overline{RD}$ and $\overline{WR}$

The $\overline{RD}$ and $\overline{WR}$ signals strobe data to or from memory or I/O space. The $\overline{RD}$ 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). $\overline{RD}$ 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. $\overline{RD}$ 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 $\overline{RD}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{RD}$ signal (see Figure 11). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to a passive state well before the beginning of T_4 (where $\overline{RD}$ goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ 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 $\overline{RD}$ signal

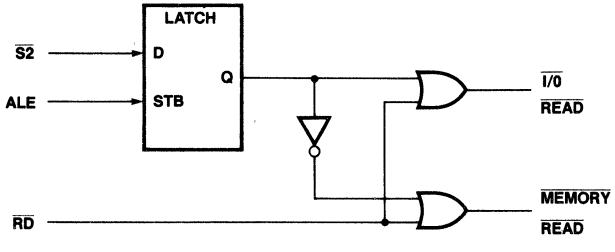
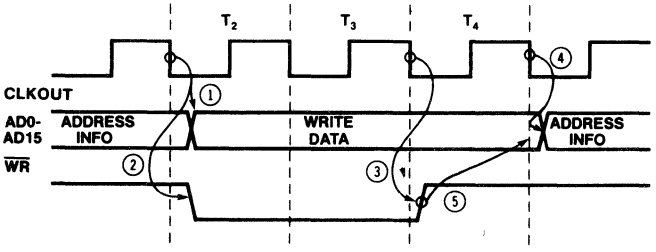


Figure 11. Generating I/O and Memory Read Signals from the 80186

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_{CLDOX}: Clock high until data invalid = 10 ns max
5. $\overline{WR}$ inactive until data invalid = t_{CLCL} min — t_{CVCTX} + t_{CLDOX}
   = 125 — 55 + 10
   = 80 ns

Figure 12. Write Cycle Timing of the 80186

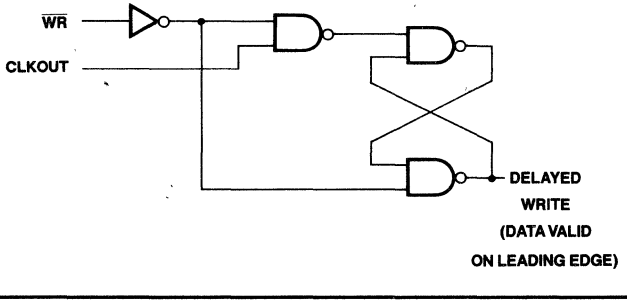


Figure 13. Synthesizing Delayed Write from the 80186

210973-003

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.

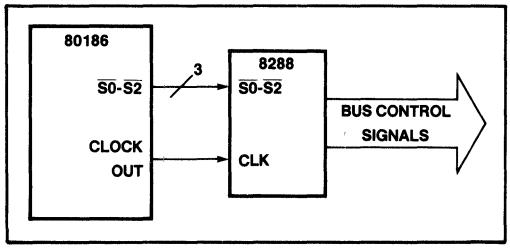


Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	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 $\overline{TEST}$ 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.

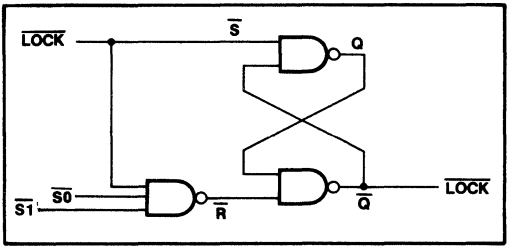


Figure 15. Circuit Holding $\overline{Lock}$ Active Until Ready is Returned

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

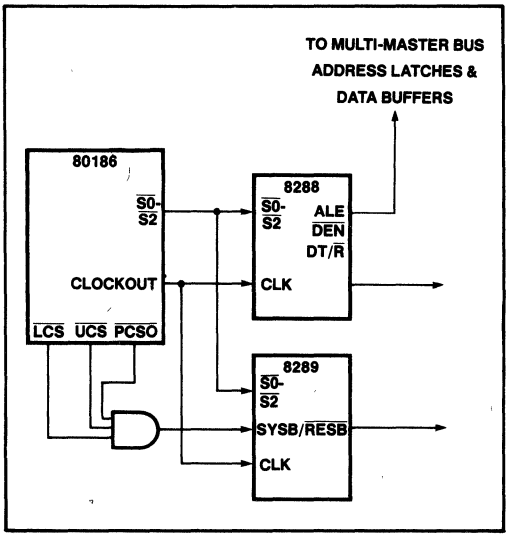


Figure 16. 80186/8288/8289 Interconnection

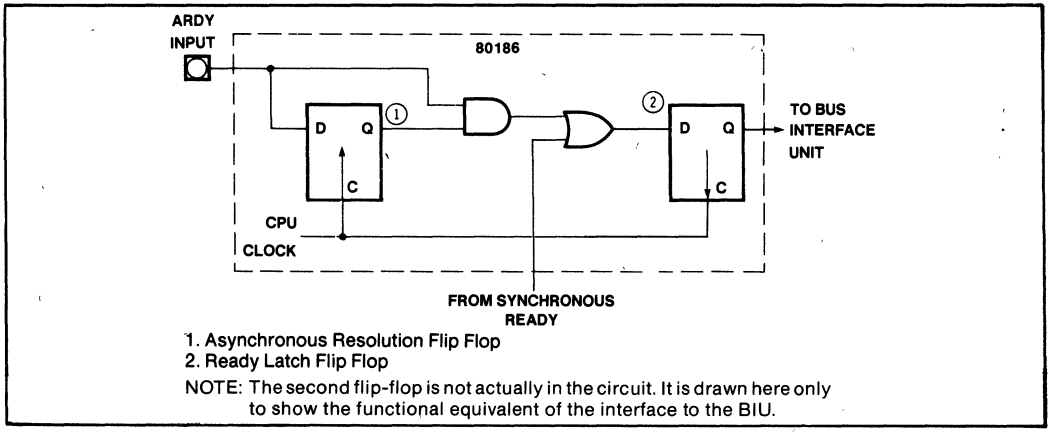
8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold 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-



1. Asynchronous Resolution Flip Flop
2. Ready Latch Flip Flop

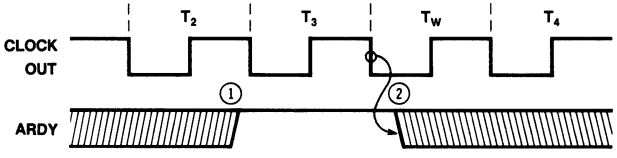
NOTE: The second flip-flop is not actually in the circuit. It is drawn here only to show the functional equivalent of the interface to the BIU.

Figure 17. Asynchronous Ready Circuitry of the 80186

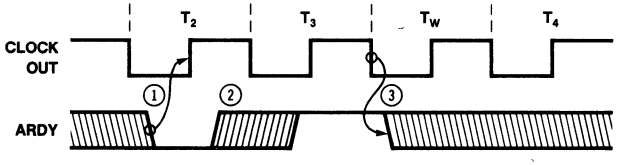
flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (see Appendix B for 80186 synchronizer information).

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.

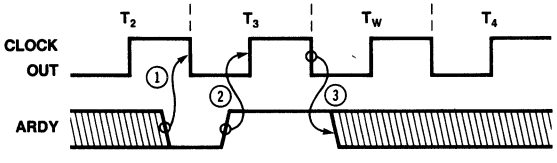
If RDY is sampled active into the RDY flip-flop at the beginning of T_3 or T_w (meaning that ARDY was sam-



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

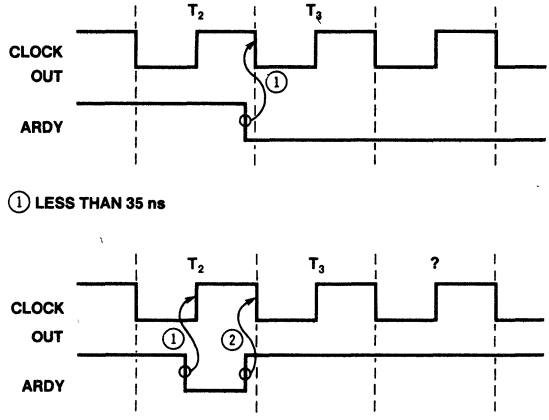
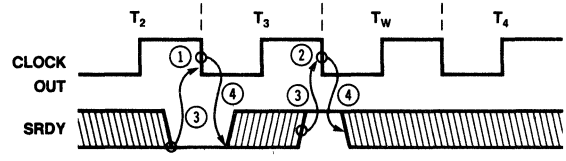


Figure 18B. Invalid ARDY Transitions

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



1. Decision: Not ready, T-state will be followed by a wait state
2. Decision: Ready, T-state will not be followed by a wait state
3. t_{SRYCL}: Synchronous ready stable until clock low (SRDY set-up time) = 35 ns min
4. t_{CLSRY}:

Clock low until synchronous ready transition (SRDY hold time) = 15 ns min

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.

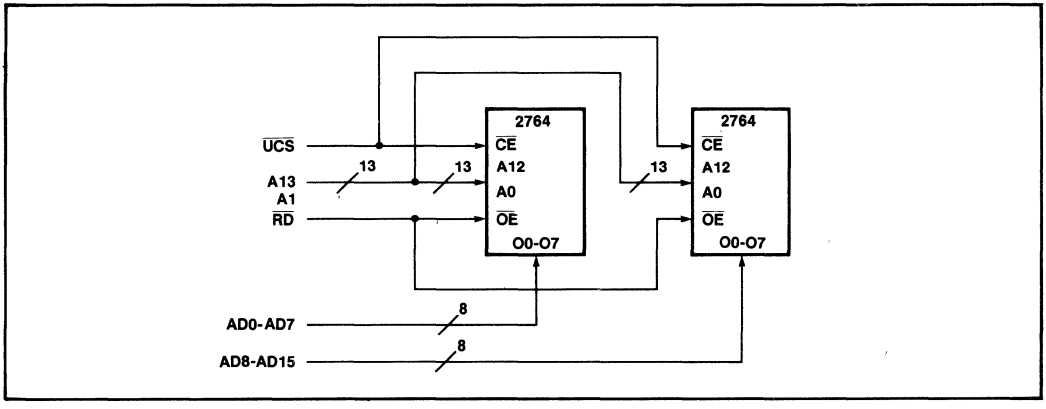


Figure 20. Example 2764/80186 Interface

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.

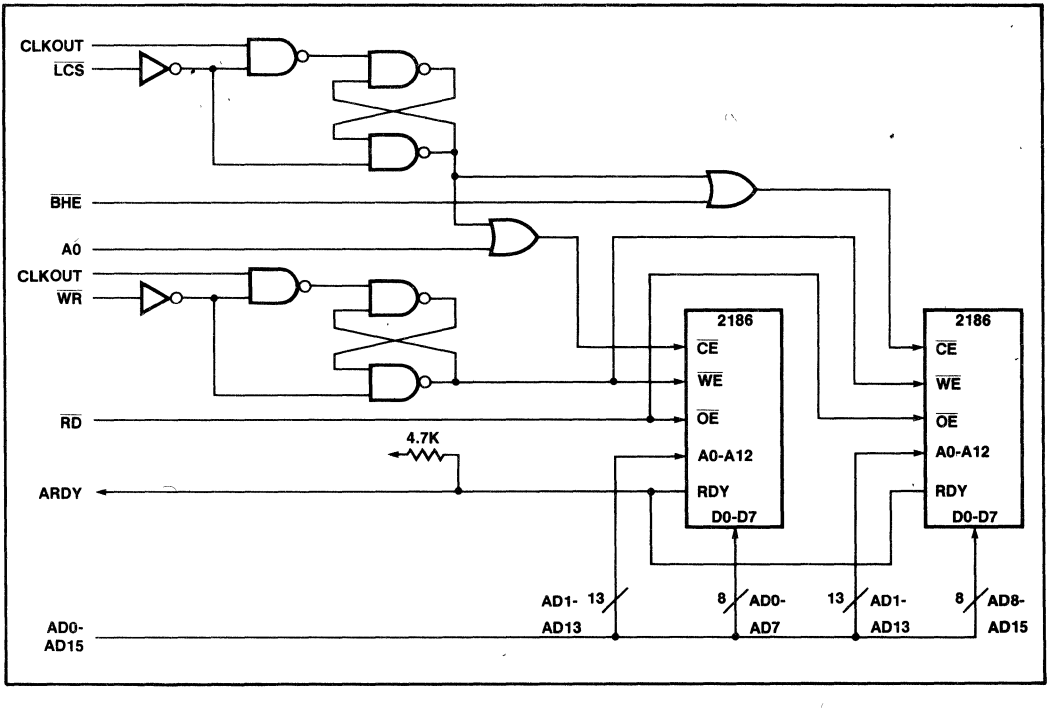


Figure 21. Example 2186/80186 Interface

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.

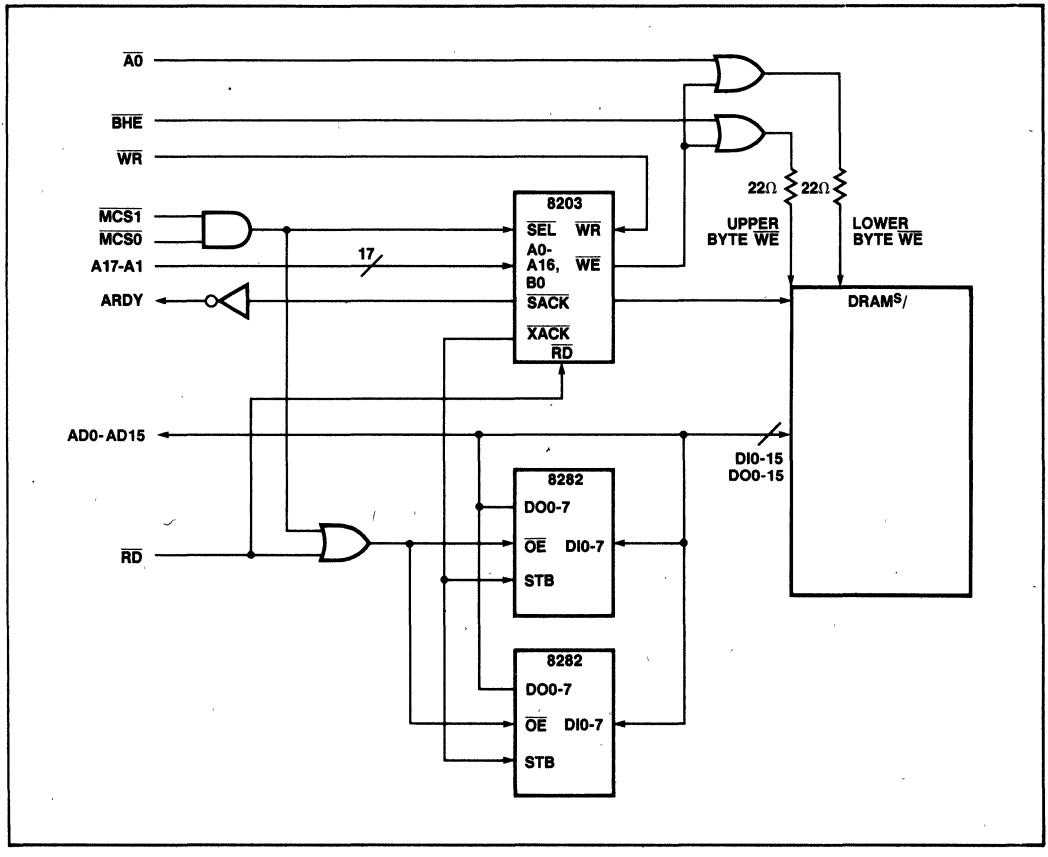
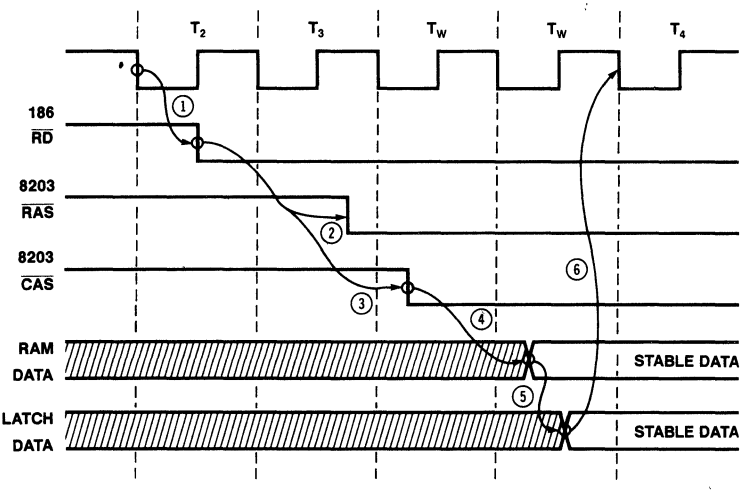


Figure 22. Example 8203/DRAM/80186 Interface



1. t_{CLEL}: Clock low until read low = 70 ns max
2. t_{CR}: Command active until RAS = 150 ns max*
3. t_{CC}: Command active until CAS = 245 ns max*
4. t_{CAC}: Access time from CAS = 85 ns max
5. t_{ISOU}: Input to output delay = 30 ns max
6. t_{DVCL}: Data valid to clock low (data in set up) = 20 ns min

Total Access Time = 70 + 245 +85 +30 +20 = 450 ns (3.6 T-states)

① & ⑥ are 186 specs
② & ③ are 8203 specs
④ is a 2164A-15 spec
⑤ is on 8282 spec

*Assumes 25MHz 8203 operation

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.

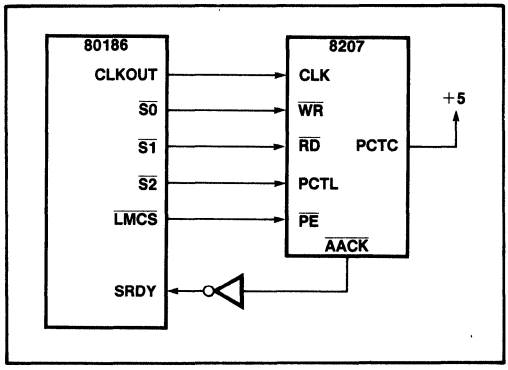


Figure 24. 80186/8207/DRAM Interface

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.

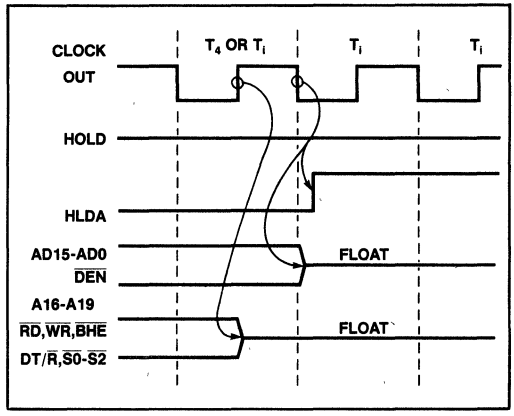


Figure 25. Signal Float/HLDA Timing of the 80186

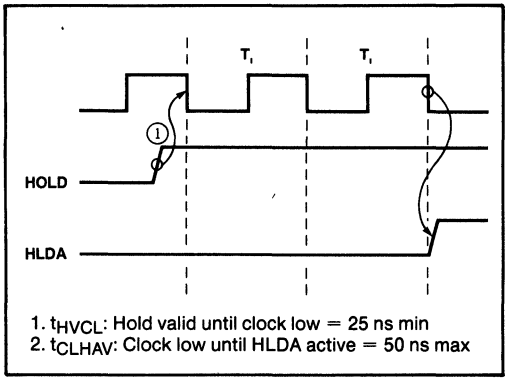
Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the 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).

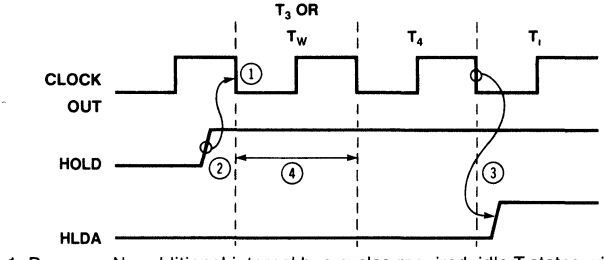


1. t_{HVCL}: Hold valid until clock low = 25 ns min
2. t_{CLHAV}: Clock low until HLDA active = 50 ns max

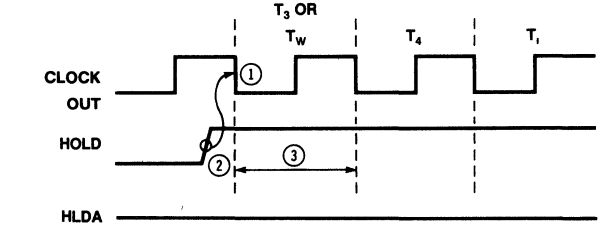
Figure 26. 80186 Idle Bus Hold/HLDA Timing

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

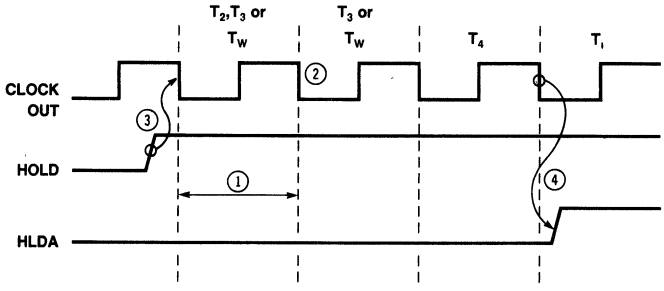


1. Decision: No additional internal bus cycles required, idle T-states will be inserted after T_4
2. Greater than 25 ns (t_{HVCL})
3. Less than 50 ns (t_{CLHAV})
4. HOLD request internally synchronized



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 4.4.1) 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.

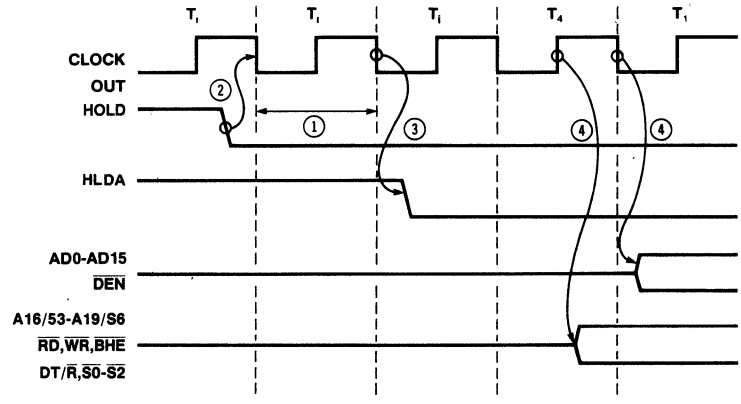


Figure 28. 80186 Coming out of Hold

- **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).

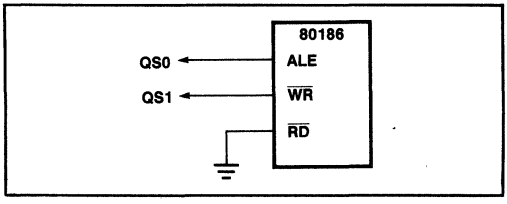


Figure 29. Generating Queue Status Information from the 80186

- **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.

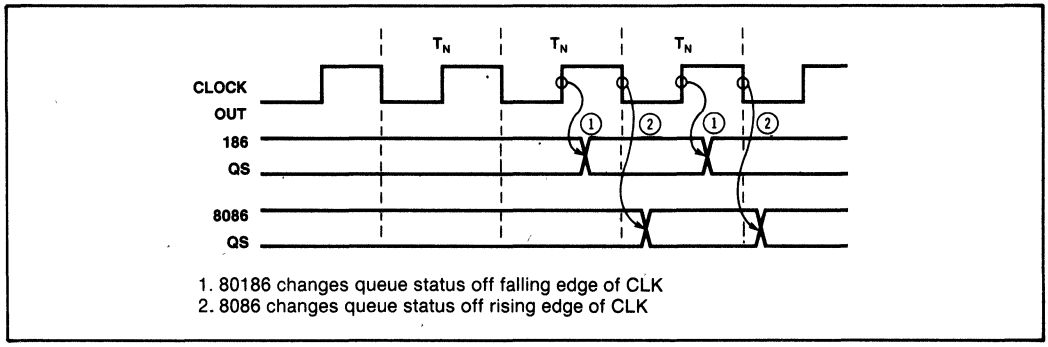


Figure 30. 80186 and 8086 Queue Status Generation

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.

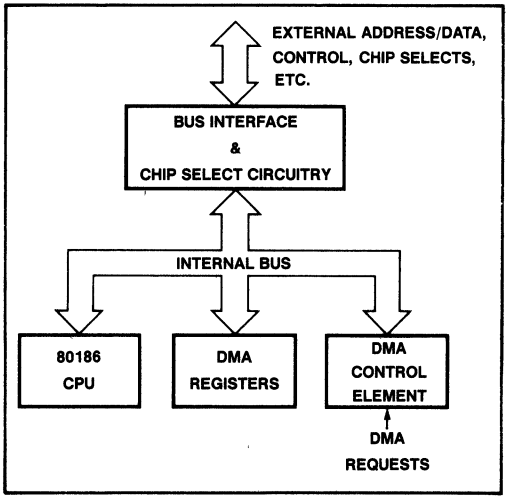


Figure 31. 80186 CPU/DMA Channel Internal Model

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

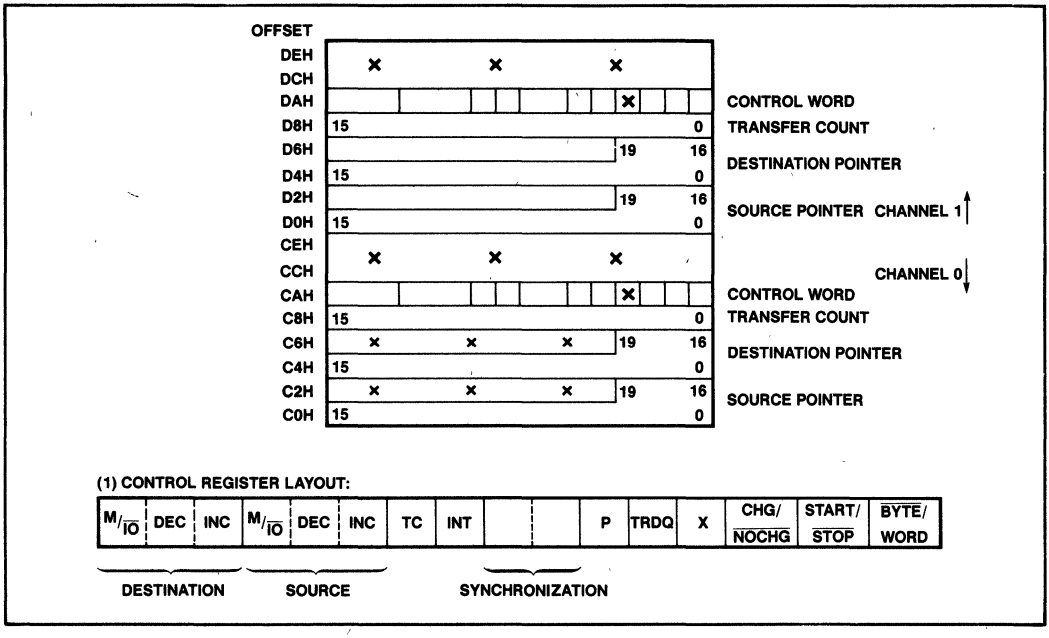


Figure 32. 80186 DMA Register Layout

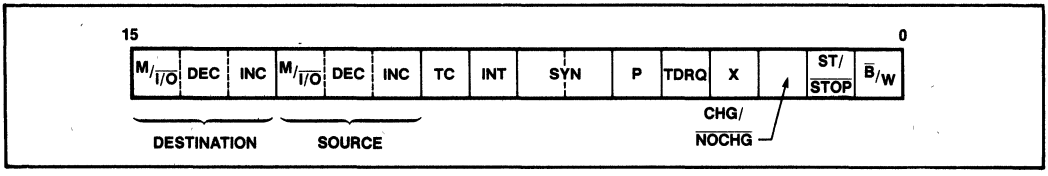
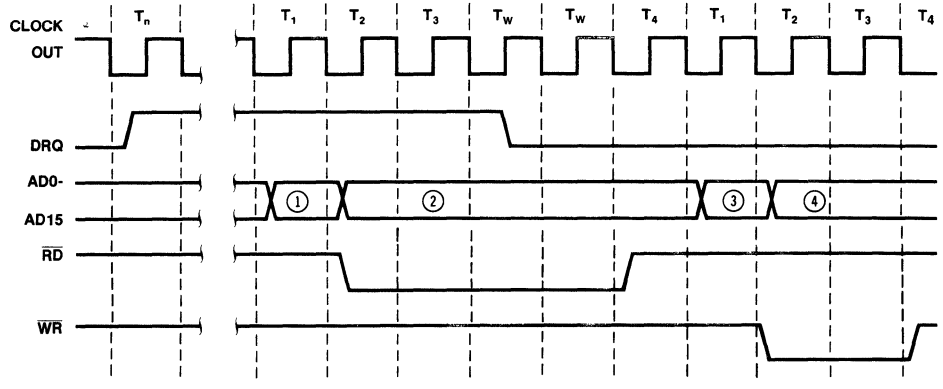


Figure 33. DMA Control Register

"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 0. 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.

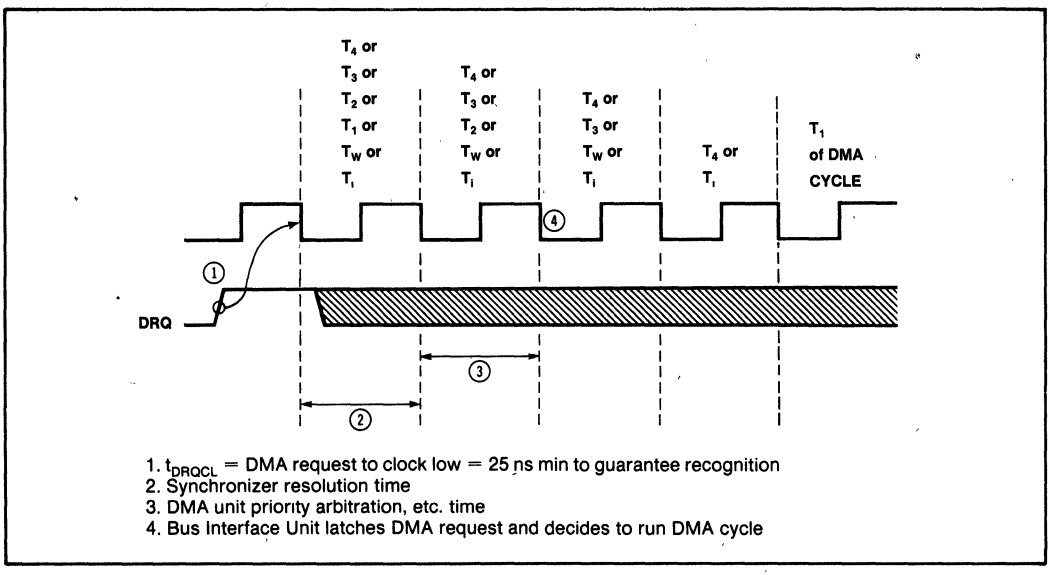
4.4 DMA Requests

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.

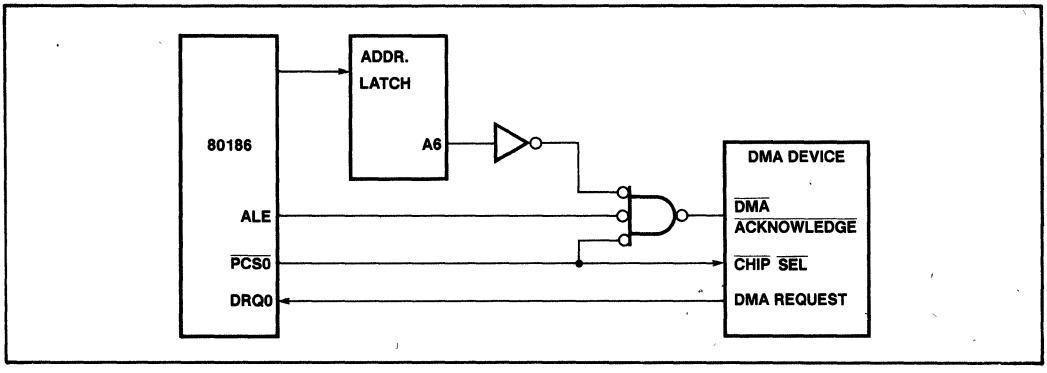


Figure 36. DMA Acknowledge Synthesis from the 80186

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

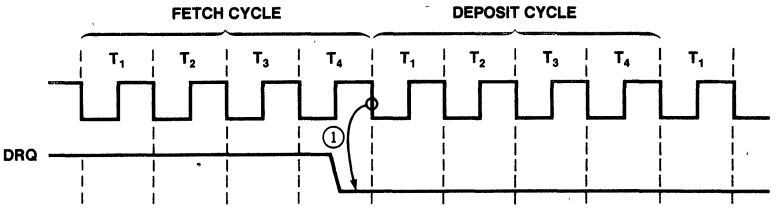
There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes 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).

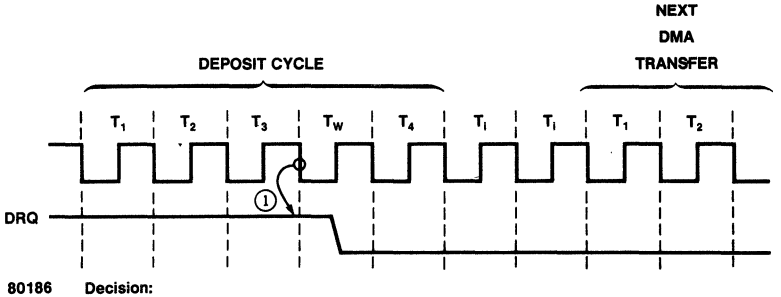
4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

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:

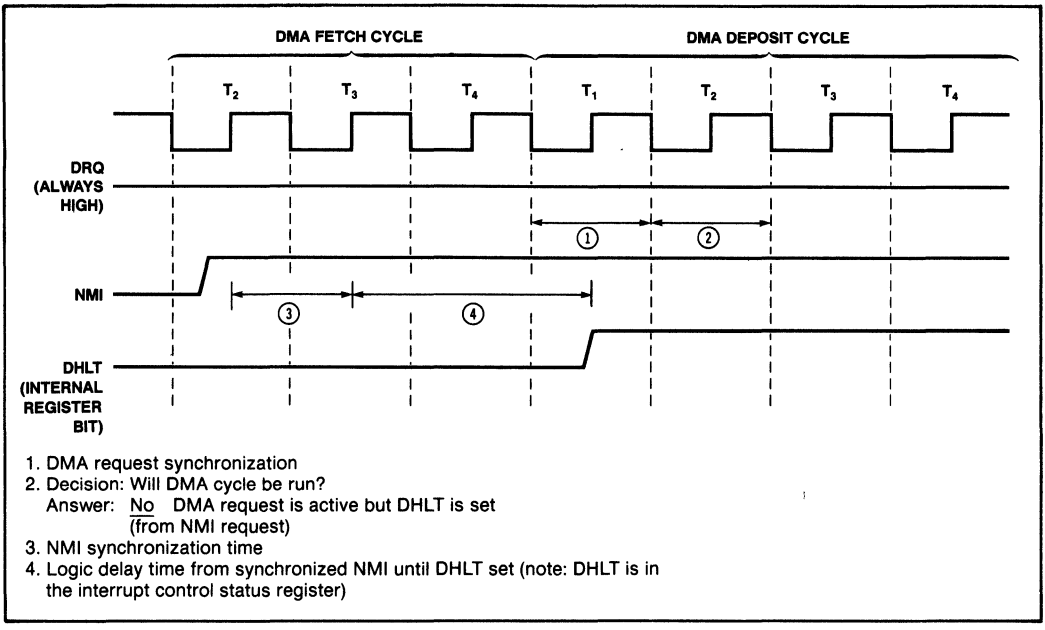
1. Current DMA destination synchronized transfer will be followed immediately by another DMA transfer

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 6.3.7). 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-

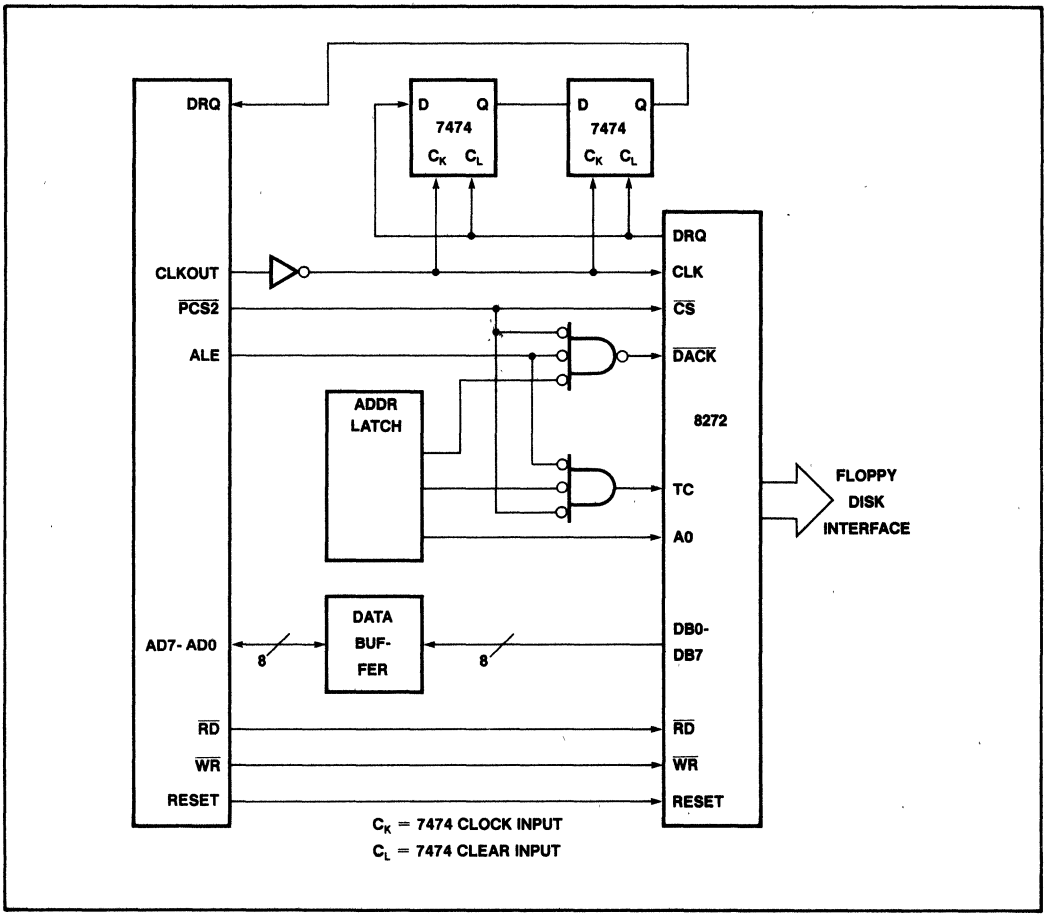


Figure 39. Example 8272/80186 DMA Interface

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 PCS line is free to generate all 8272 select signals. If more than one chip select is free, however, different 80186 generated PCS lines could be used for each function. For example, PCS2 could be used to select the 8272, 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 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 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.

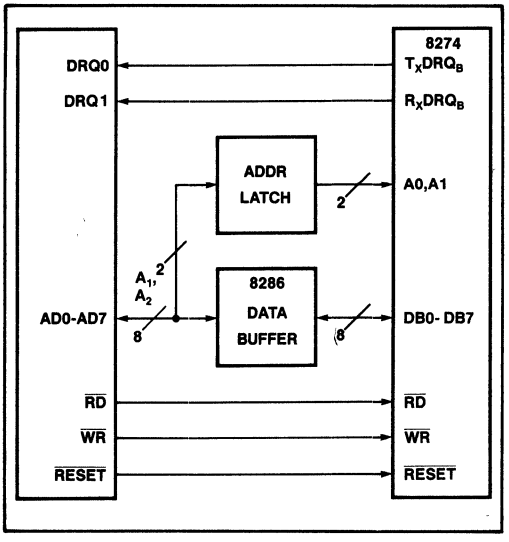


Figure 40. Example 8274/80186 DMA Interface

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

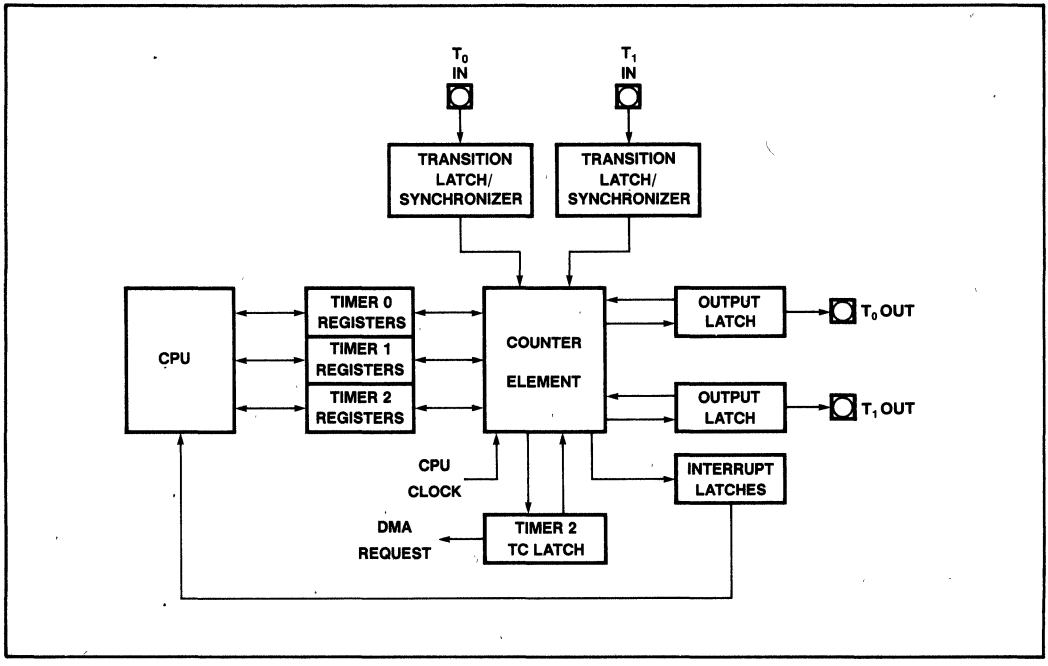


Figure 41. 80186 Timer Model

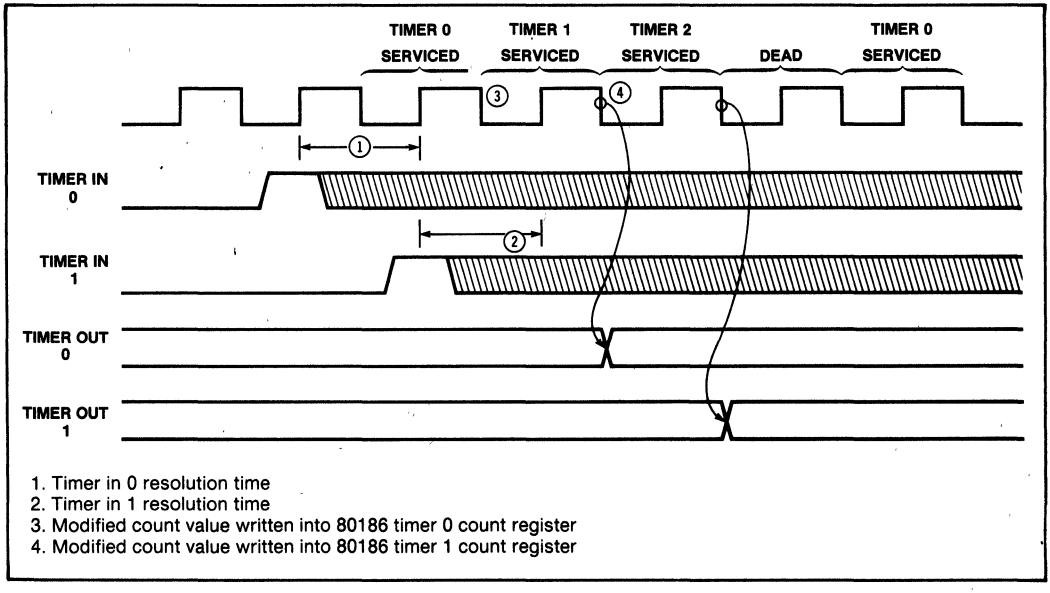


Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

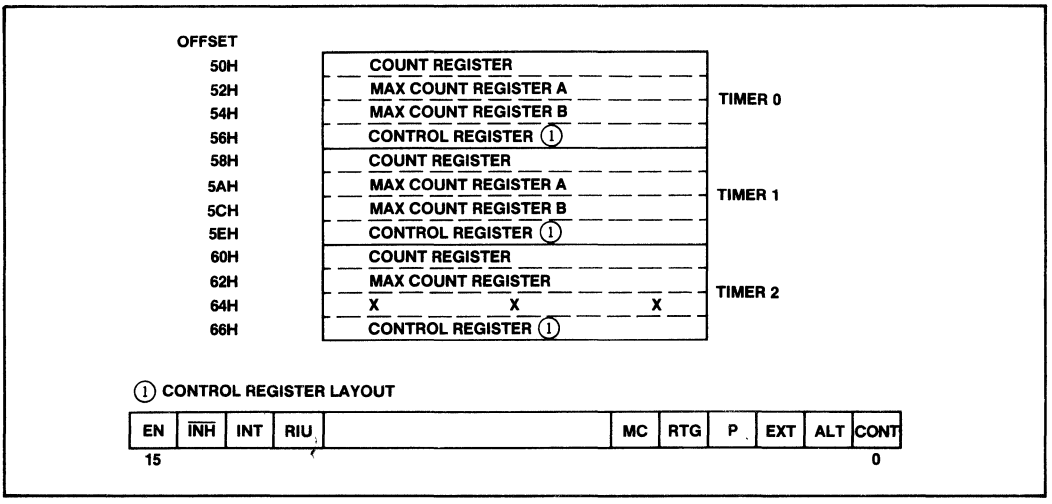


Figure 43. 80186 Timer Register Layout

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.

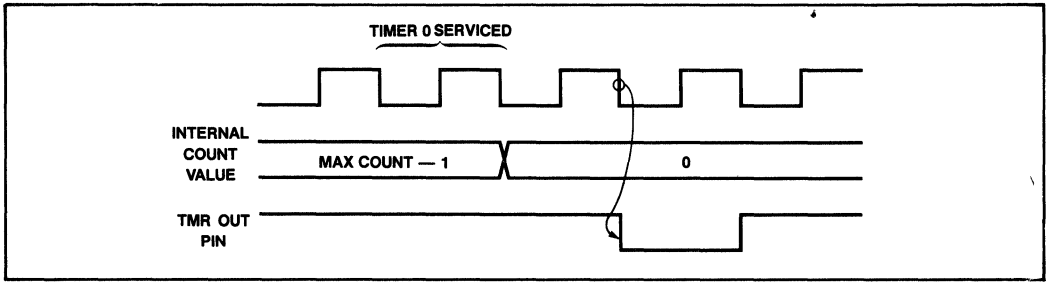


Figure 44. 80186 Timer Out Signal

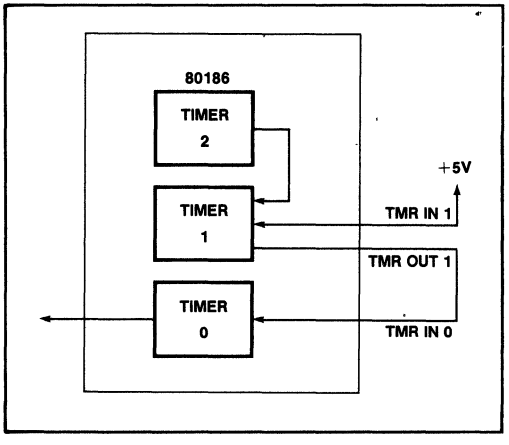


Figure 45. 80186 Real Time Clock

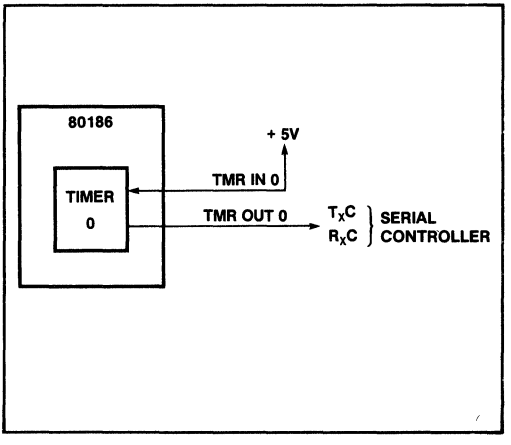


Figure 46. 80186 Baud Rate Generator

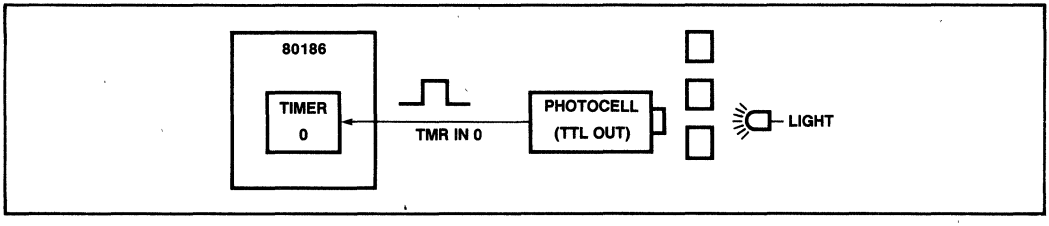


Figure 47.

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.

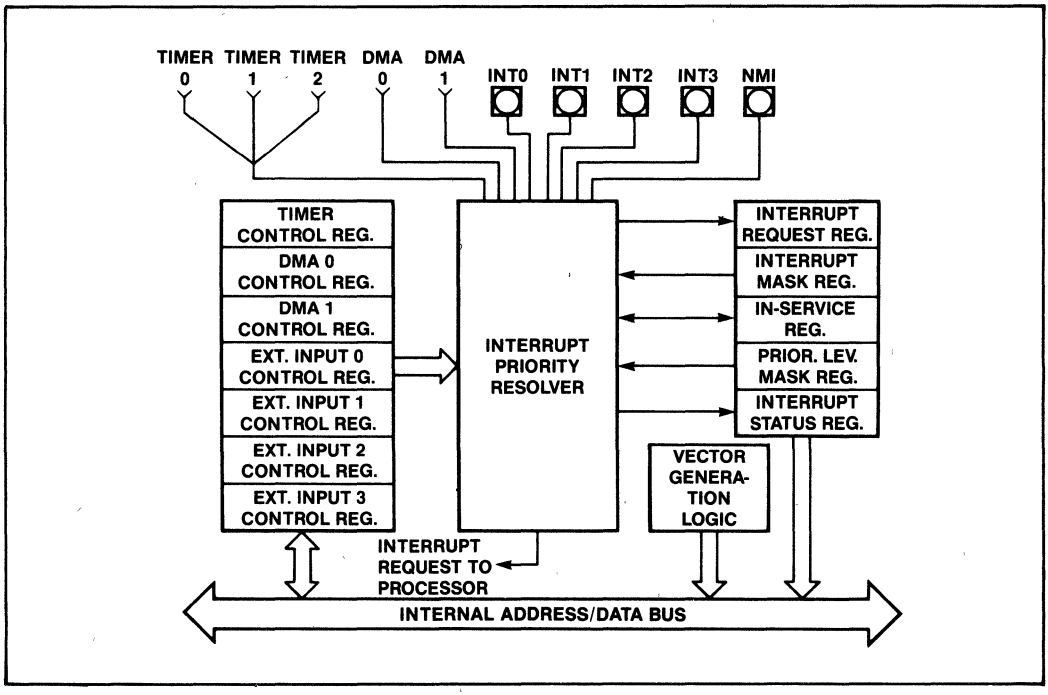


Figure 48. 80186 Interrupt Controller Block Diagram

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

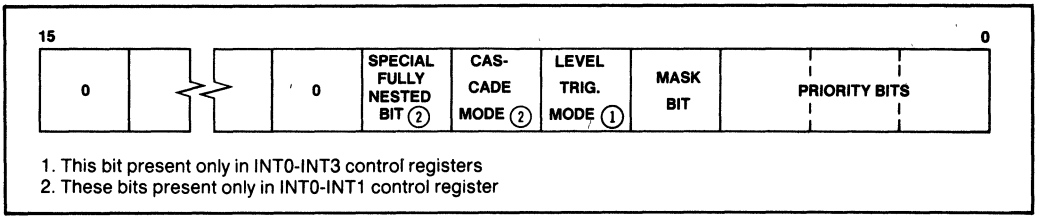


Figure 50. Interrupt Controller Control Register

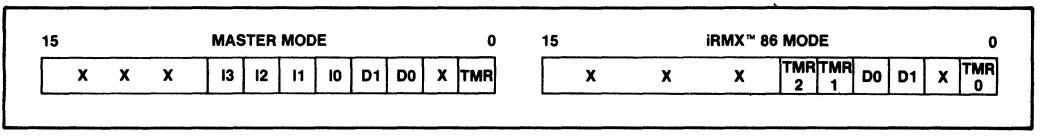


Figure 51. 80186 Interrupt Controller In-Service, Interrupt Request and Mask Register Format

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 lowest priority an interrupt may have that will cause an interrupt acknowledge. Interrupts received which have a lower priority will be effectively masked off. Upon reset this register is set to the lowest priority of 7 to enable all interrupts of any priority. This register may be read or written.

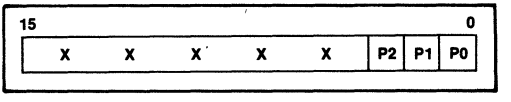


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

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.

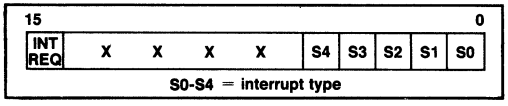


Figure 53. 80186 Poll & Poll Status Register Format

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.

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. 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 bit directly to the in-service register. If the

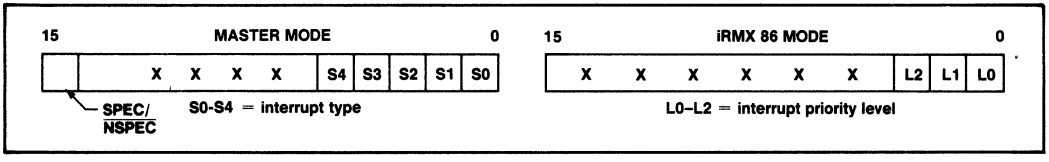


Figure 54. 80186 End of Interrupt Register Format

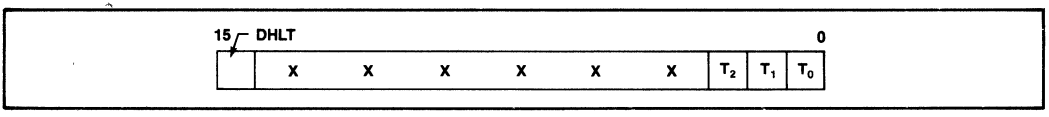


Figure 55. 80186 Interrupt Status Register Format

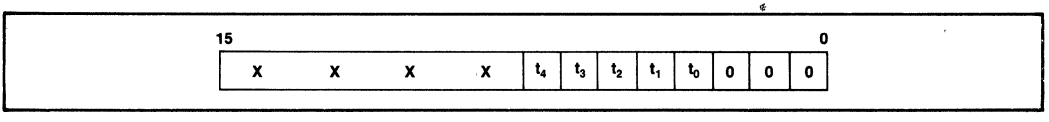


Figure 56. 80186 Interrupt Vector Register Format (iRMX 86 mode only)

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

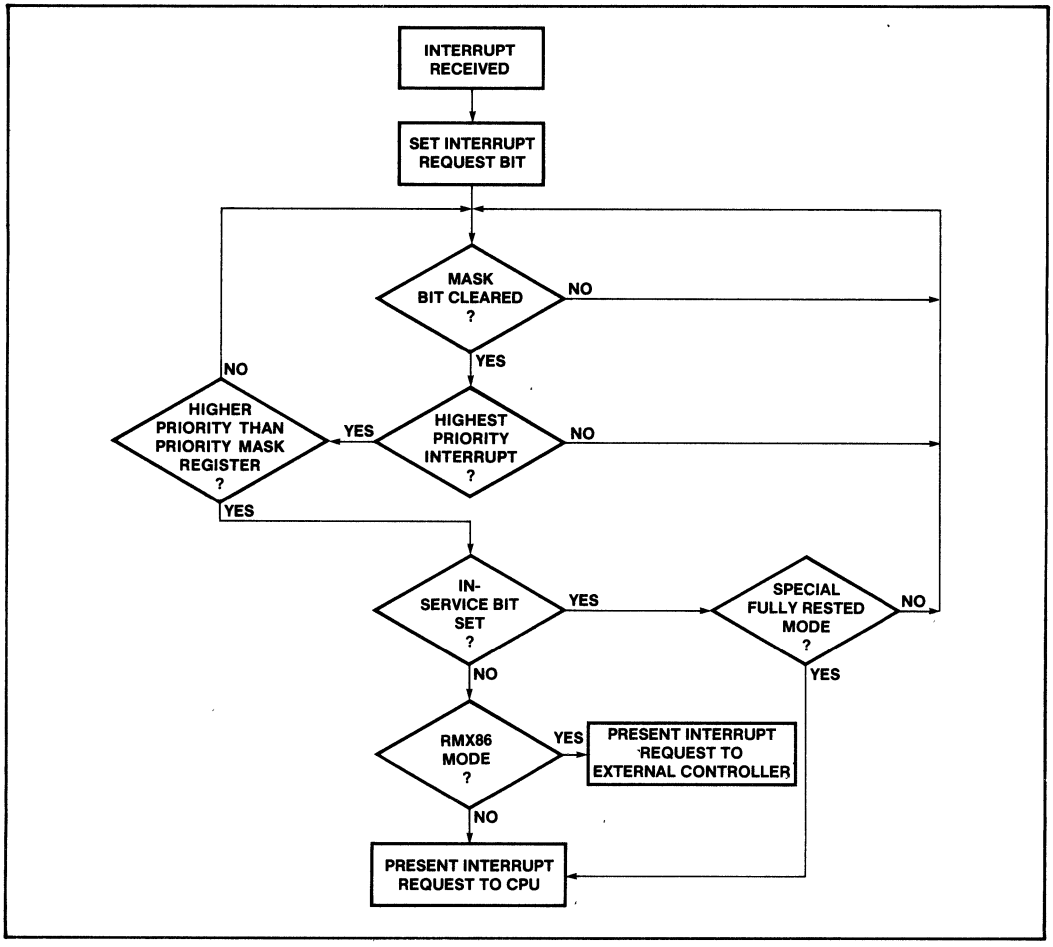


Figure 57. 80186 Interrupt Request Sequencing

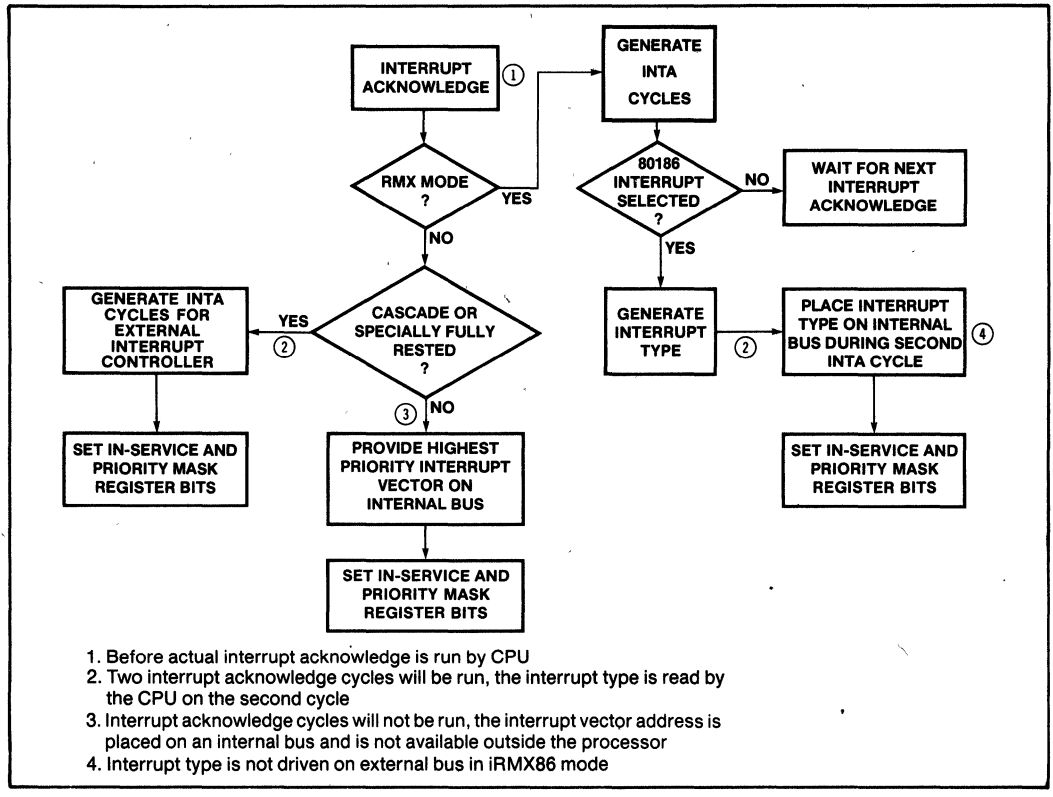
interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt 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-



1. Before actual interrupt acknowledge is run by CPU
2. Two interrupt acknowledge cycles will be run, the interrupt type is read by the CPU on the second cycle
3. Interrupt acknowledge cycles will not be run, the interrupt vector address is placed on an internal bus and is not available outside the processor
4. Interrupt type is not driven on external bus in iRMX86 mode

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 reset the interrupt request bit in the integrated controller. 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

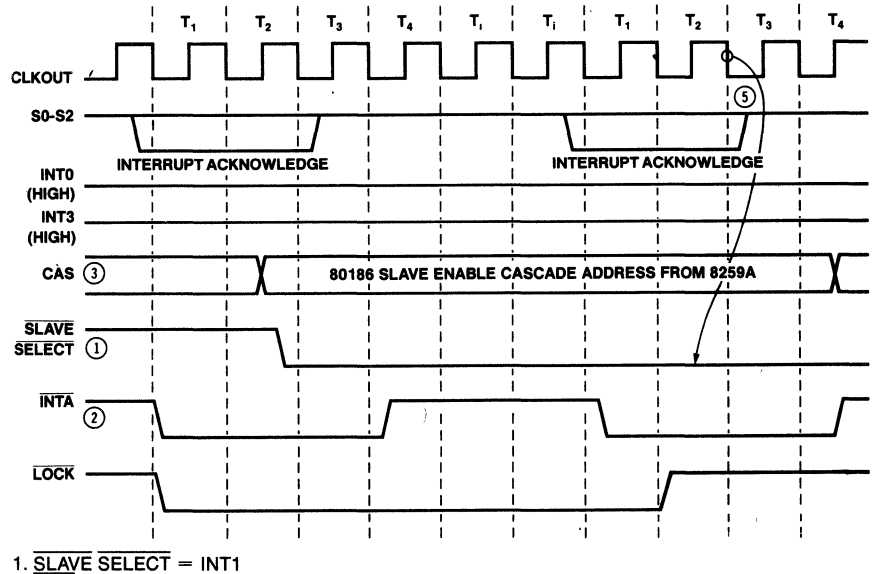
Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need be be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster then the interrupt response when external vectoring is required, or if the interrupt controller 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



1. SLAVE SELECT = INT1
2. INTA = INT2
3. Driven by external interrupt controller
4. SLAVE SELECT must be driven before Phase 2 of T_2 of the second INTA cycle
5. SLAVE SELECT read by 80186

Figure 59. 80186 iRMX-86 Mode Interrupt Acknowledge Timing

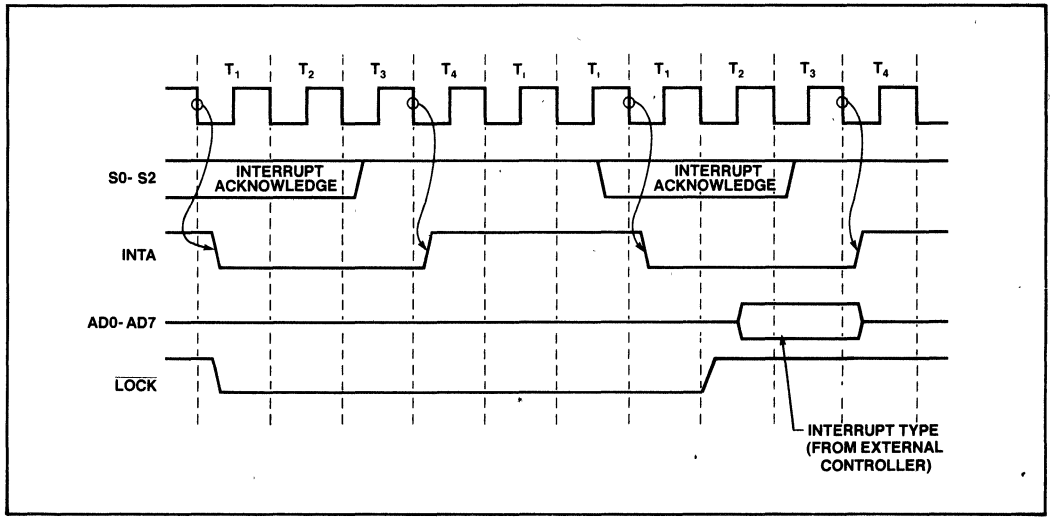


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

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.

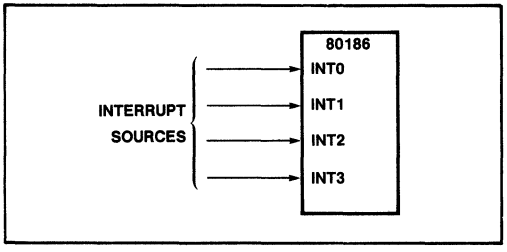


Figure 61. 80186 Non-Cascaded Interrupt Connection

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.

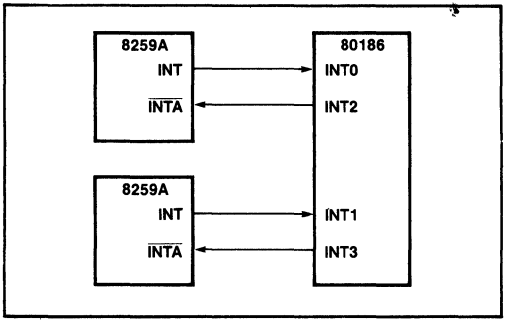


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

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.

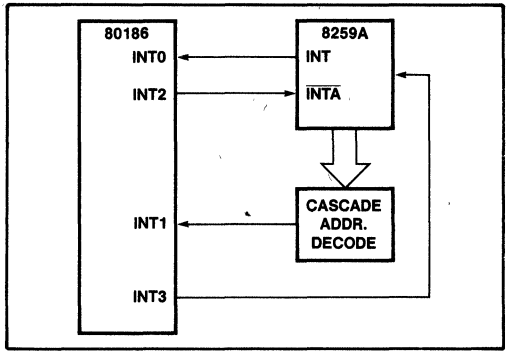


Figure 63. 80186 iRMX86 Mode Interface

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.

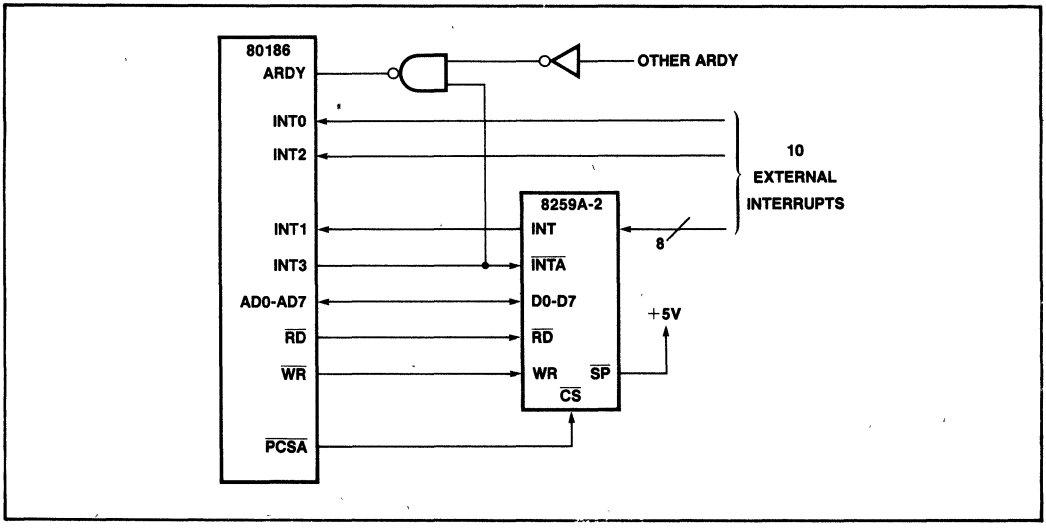


Figure 64. 80186/8259A Interrupt Cascading

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

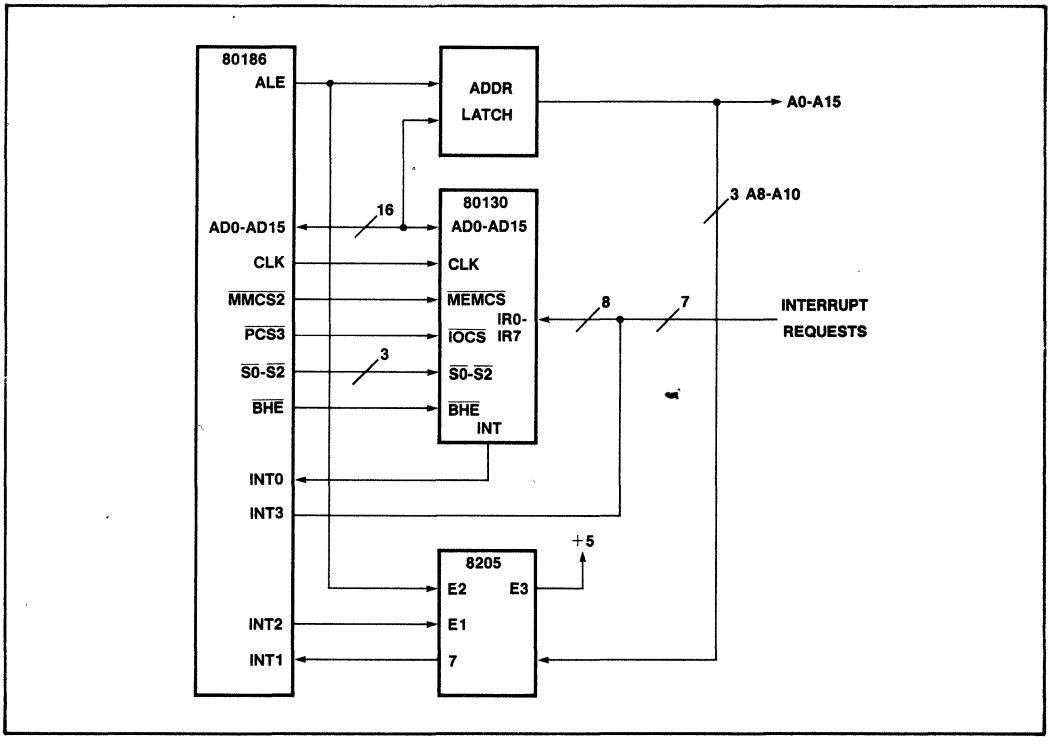


Figure 65. 80186/80130 iRMX86 Mode Interface

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.

The following parameters may be used for choosing a crystal:

Temperature Range:	0 to 70° C
ESR (Equivalent Series Resistance):	30Ω max
C_0 (Shunt Capacitance of Crystal):	7.0 pf max
C_1 (Load Capacitance):	20 pf ± 2 pf
Drive Level:	1 mw max

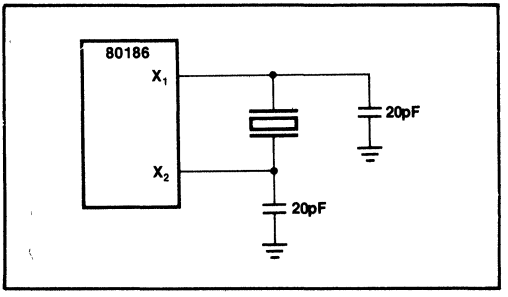


Figure 67. 80186 Crystal Connection

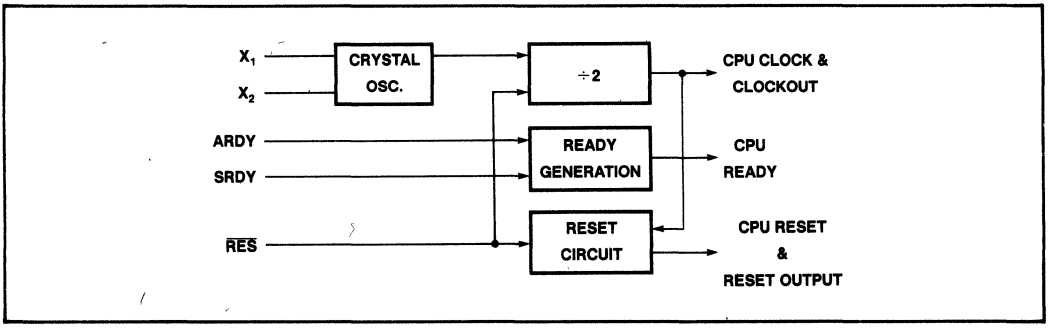


Figure 66. 80186 Clock Generator Block Diagram

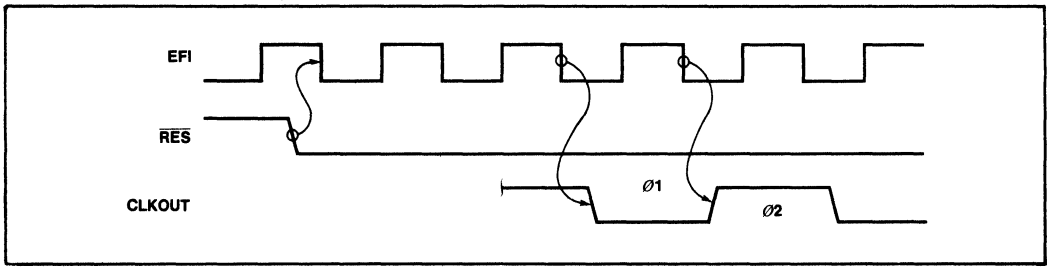


Figure 68. 80186 Clock Generator Reset

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 divide-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).

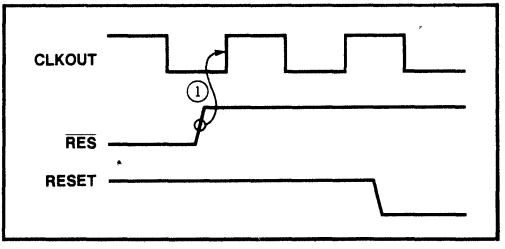


Figure 69. 80186 Coming out of Reset

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).

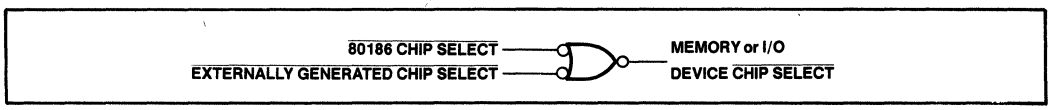


Figure 70. 80186/External Chip Select/Device Chip Select Generation

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).

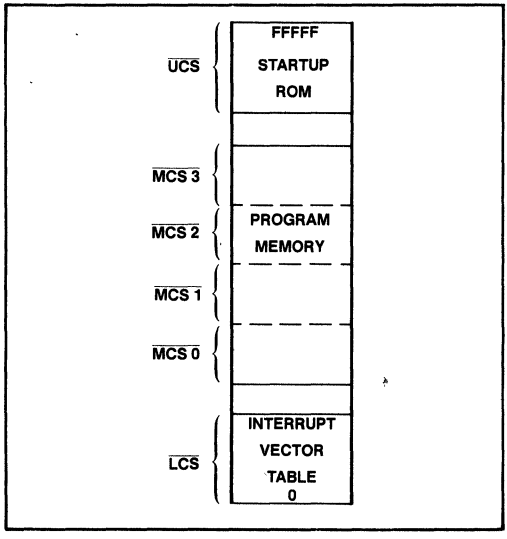


Figure 71. 80186 Memory Areas & Chip Selects

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.

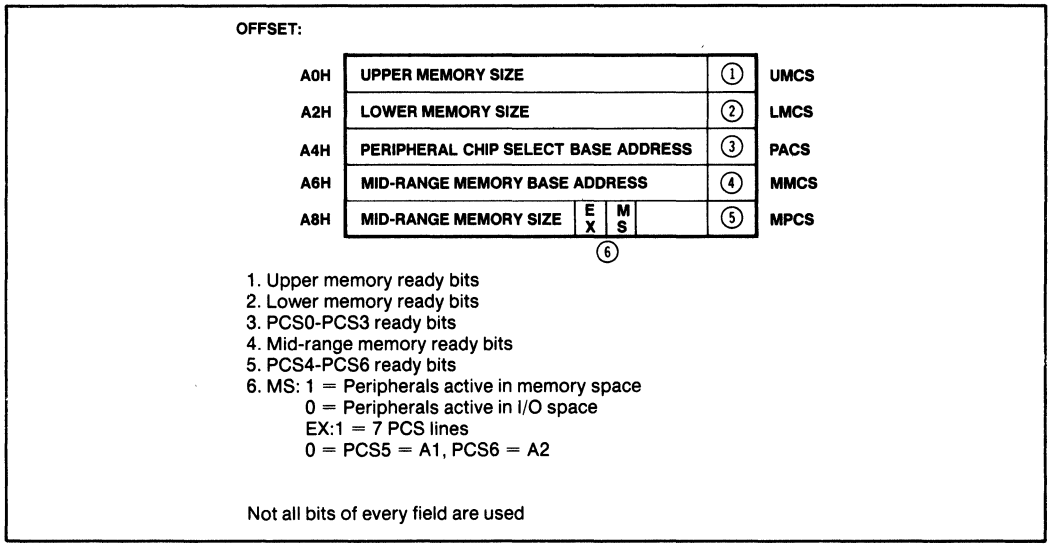


Figure 72. 80186 Chip Select Control Registers

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.

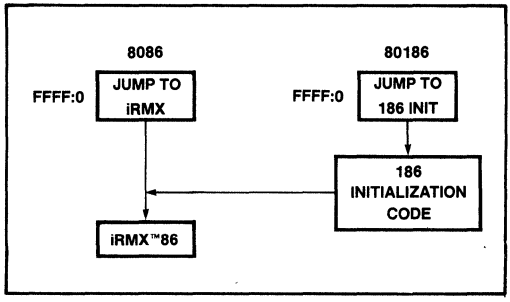


Figure 73. iRMX-86 Initialization with 8086 & 80186

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.

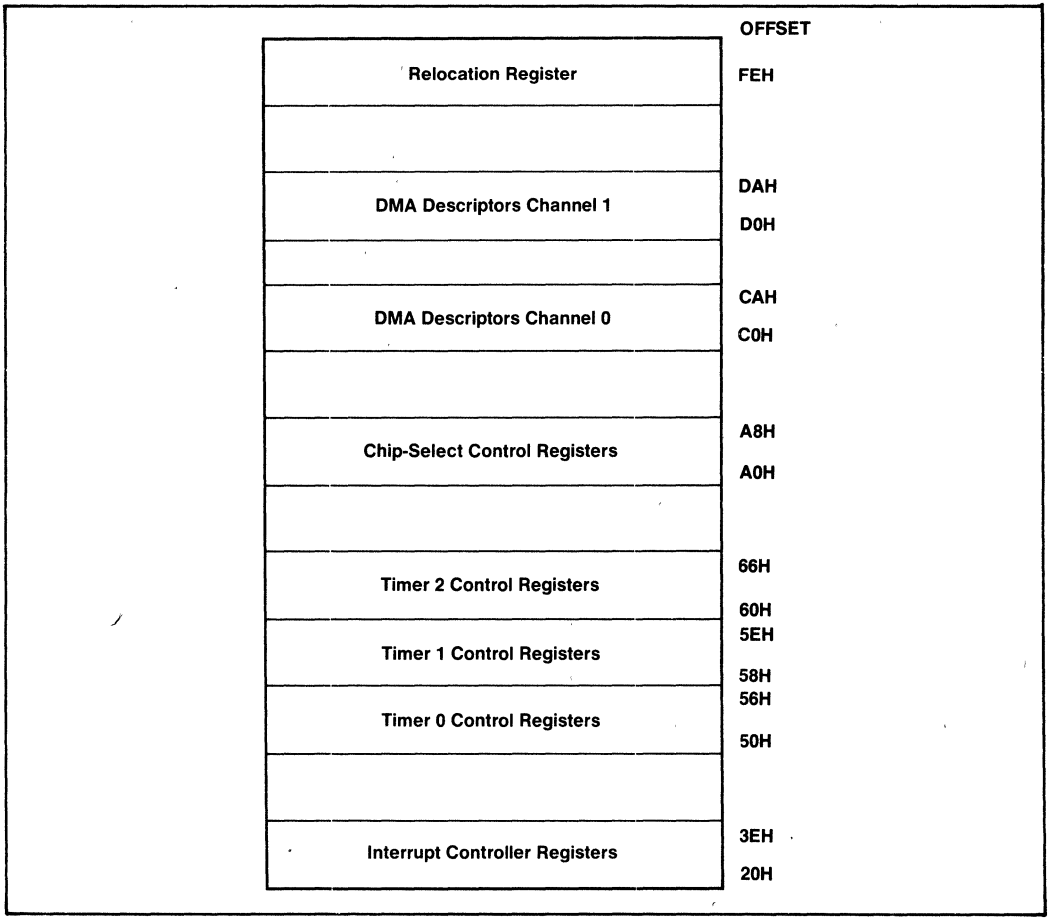


Figure A-1. 80186 Integrated Peripheral Control Block

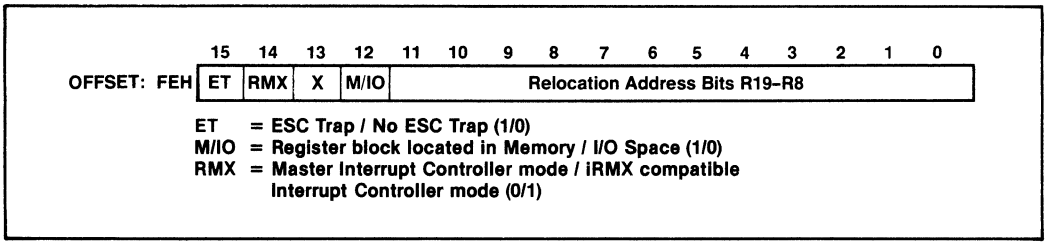


Figure A-2. 80186 Relocation Register Layout

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.

Whenever mapping the 188 peripheral control block to another location, the programming of the relocation register should be done with a byte write (i.e. OUT DX,AL). Any access to the control block is done 16 bits at a time. Thus, internally, the relocation register will get written with 16 bits of the AX register while externally, the BIU will run only one 8 bit bus cycle. If a word instruction is used (i.e. OUT DX,AX), the relocation register will be written on the first bus cycle. The BIU will then run a second bus cycle which is unnecessary. The address of the second bus cycle will no longer be within the control block (i.e. the control block was moved on the first cycle), and therefore, will require the generation of an external ready signal to complete the cycle. For this reason we recommend byte operations to the relocation register. Byte instructions may also be used for the other registers in the control block and will eliminate half of the bus cycles required if a word operation had been specified. Byte operations are only valid on even addresses though, and are undefined on odd addresses.

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 internally, while externally, the BIU runs two bus cycles.

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 will 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 internally while externally the BIU runs two bus cycles.

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.

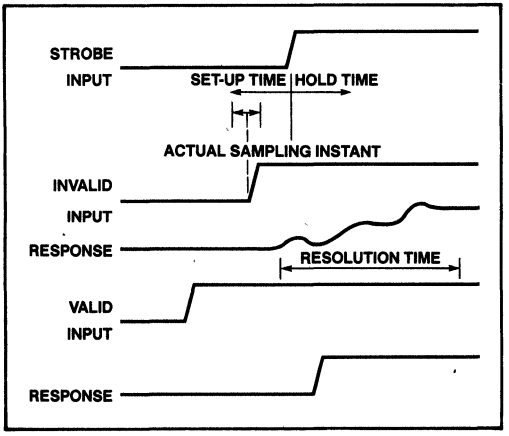


Figure B-1. Valid & Invalid Latch Input Transitions & Responses

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 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,0100000000000000B    ;  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      0F038H        ;  64K, no wait states
LMCS_value              equ      07F8H         ;  32K, no wait states
PACS_value              equ      007EH         ;  peripheral base at 400H, 2 ws
MPCS_value              equ      81B8H         ;  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.

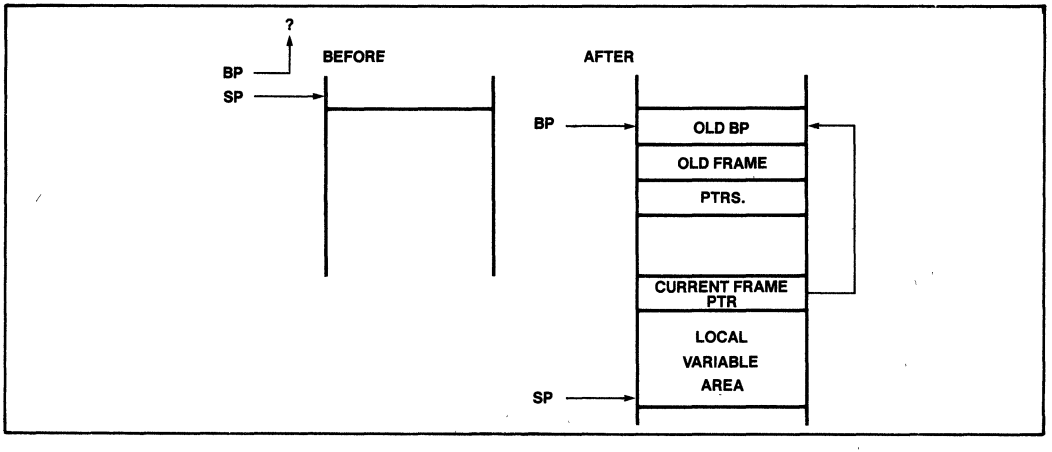


Figure H-1. ENTER Instruction Stack Frame

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 286 Microprocessors

4

ADVANCE INFORMATION

iAPX 286/10
HIGH PERFORMANCE MICROPROCESSOR WITH MEMORY MANAGEMENT AND PROTECTION

(80286-8, 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 80286-8**
  - **—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.

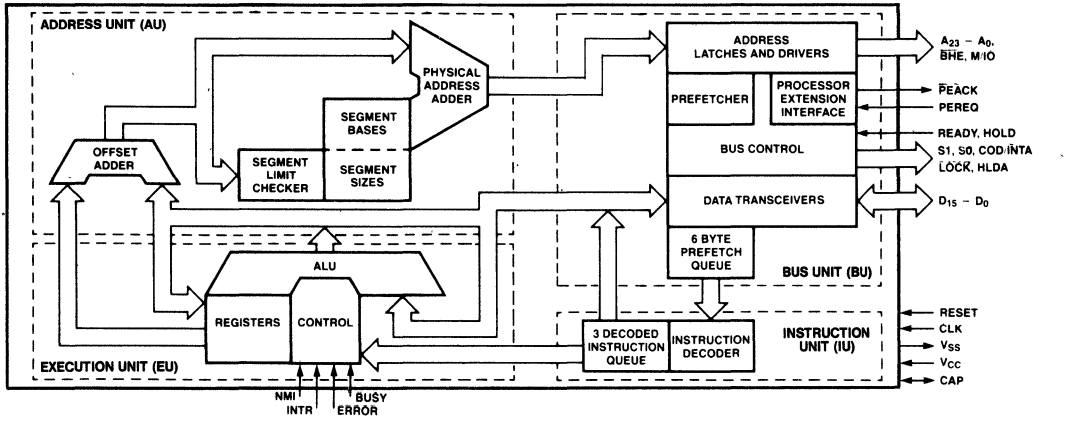
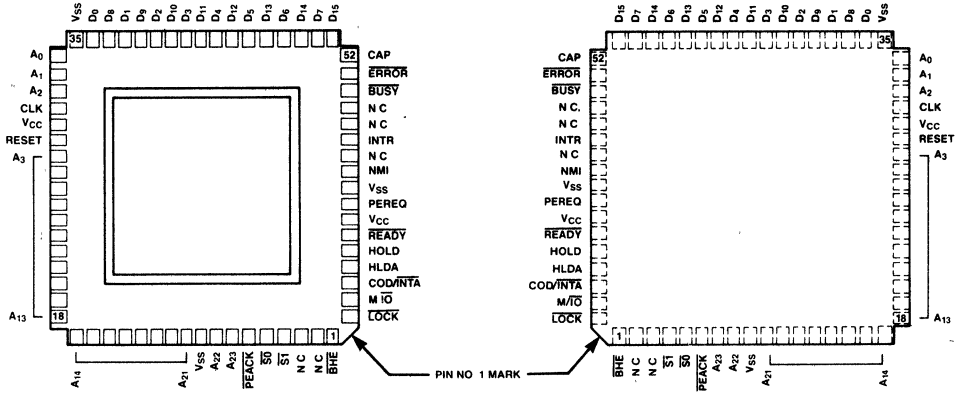


Figure 1. 80286 Internal Block Diagram

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.
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.

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

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) / S0, S1, PEACK, A23-A0, BHE, LOCK 0 (LOW) / M/IO, COD/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.

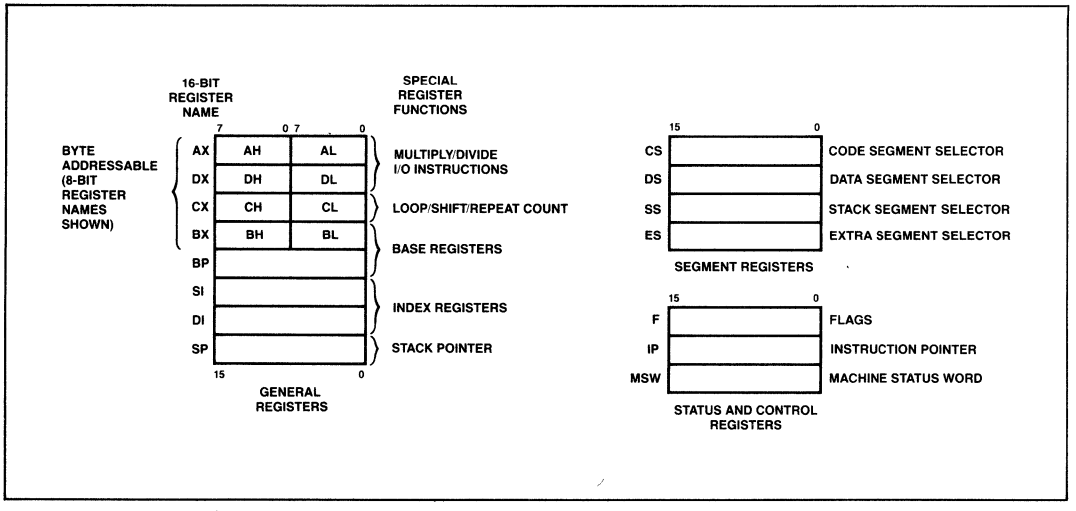


Figure 3. Register Set

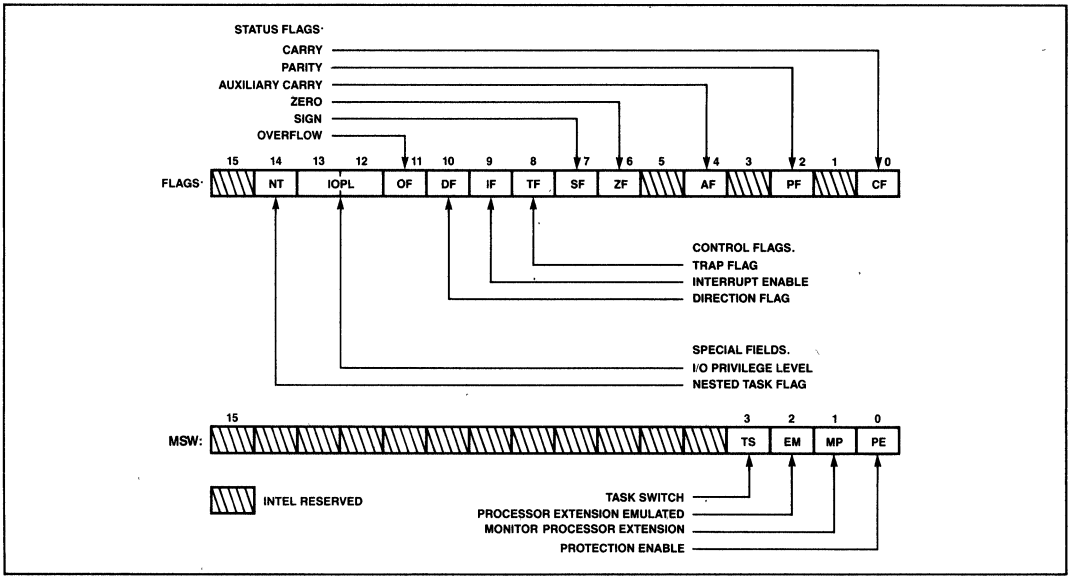


Figure 3a. Status and Control Register Bit Functions

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.

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.

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

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 $\overline{BUSY}$ 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.

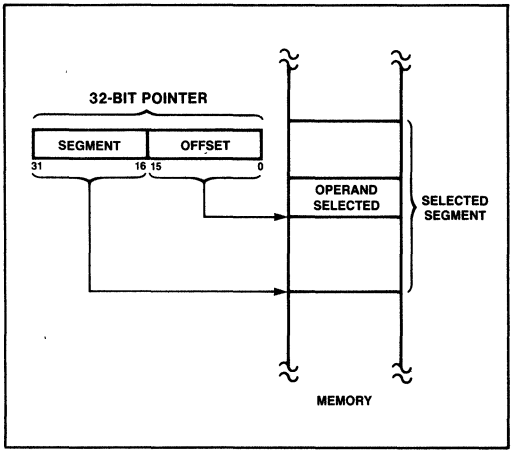


Figure 5. Two Component Address

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)

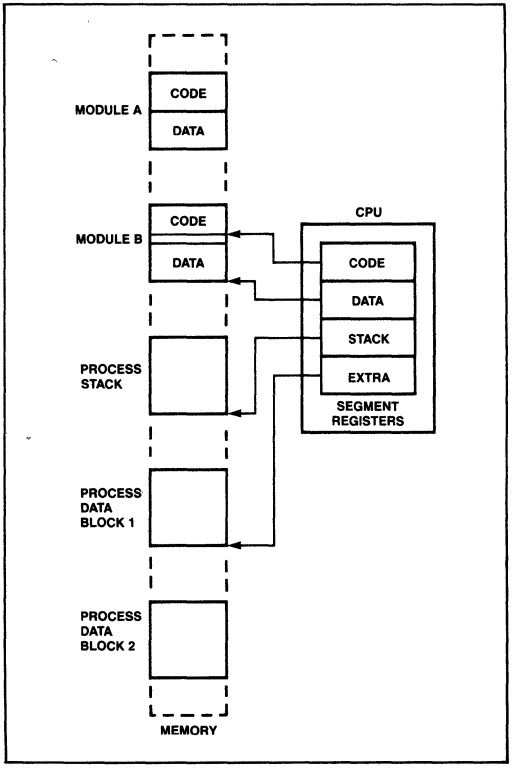


Figure 6. Segmented Memory Helps Structure Software

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.

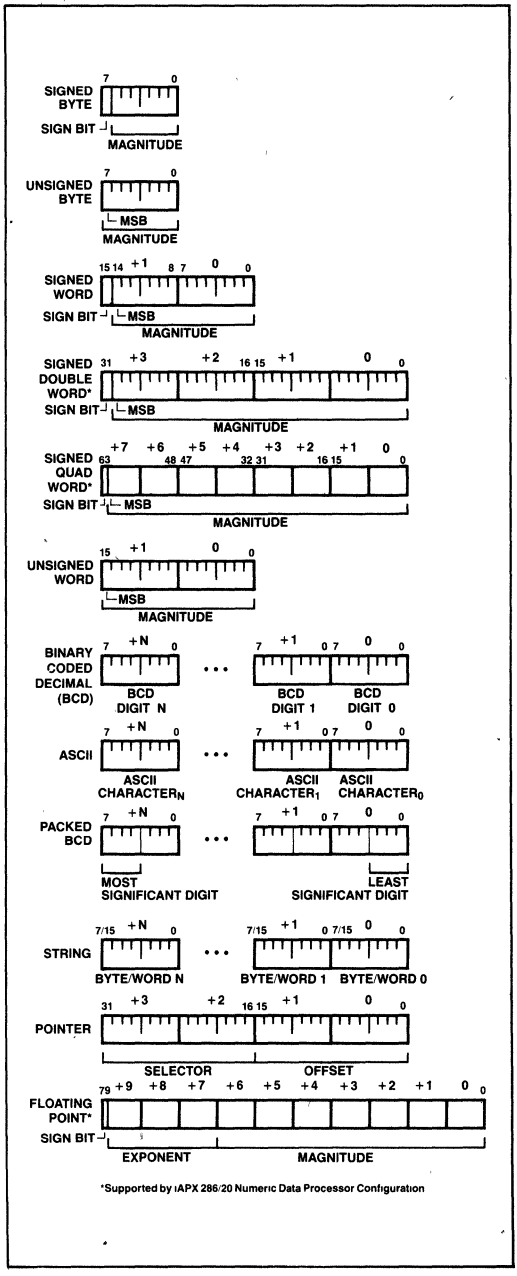


Figure 7. iAPX 286 Supported Data Types

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.

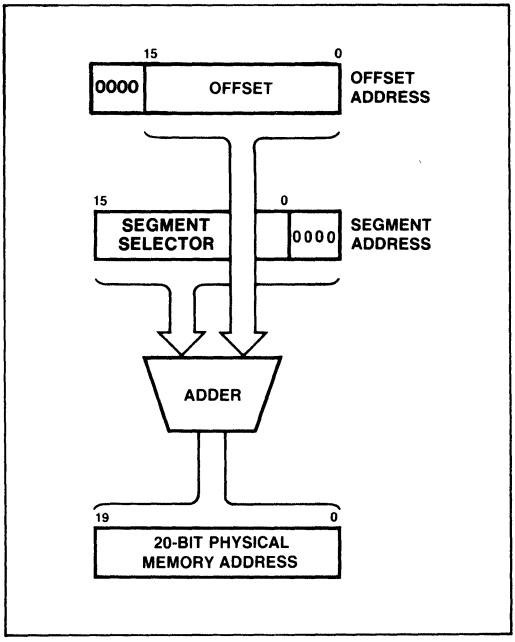


Figure 8. iAPX 86 Real Address Mode Address Calculation

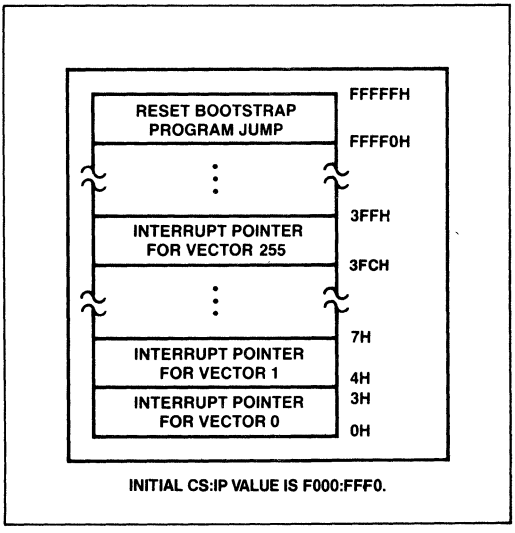


Figure 9. iAPX 86 Real Address Mode Initially Reserved Memory Locations

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.

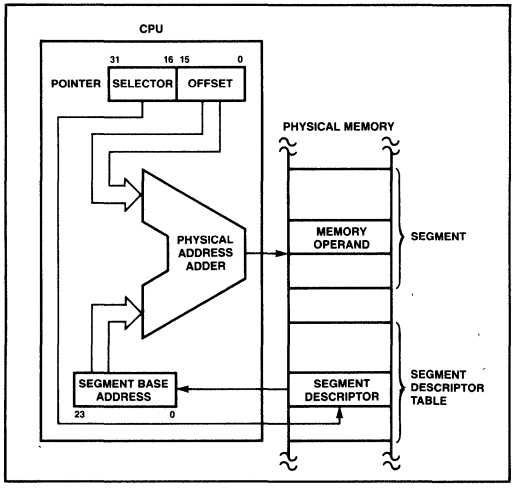


Figure 10. Protected Mode Memory Addressing

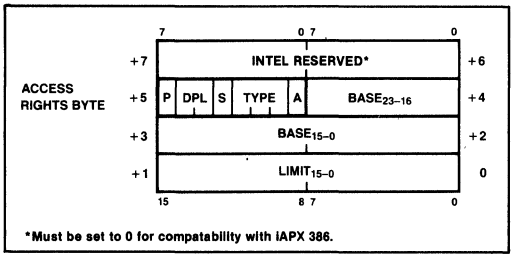
DESCRIPTORS

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



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	
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.	

Type Field Definition

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).

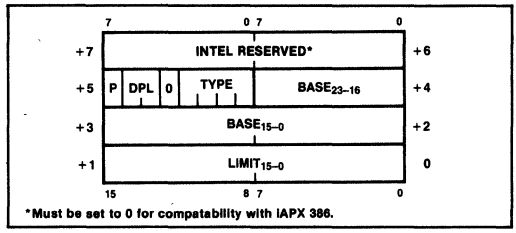
A code segment (S = 1, E = 1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R = 0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared 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 is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

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

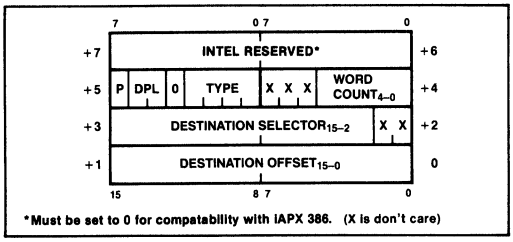
GATE DESCRIPTORS (S = 0, TYPE = 4-7)

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.

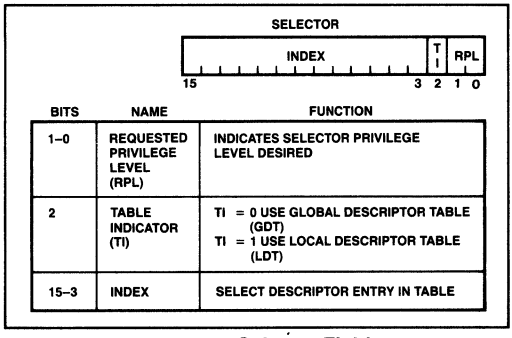
The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege 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

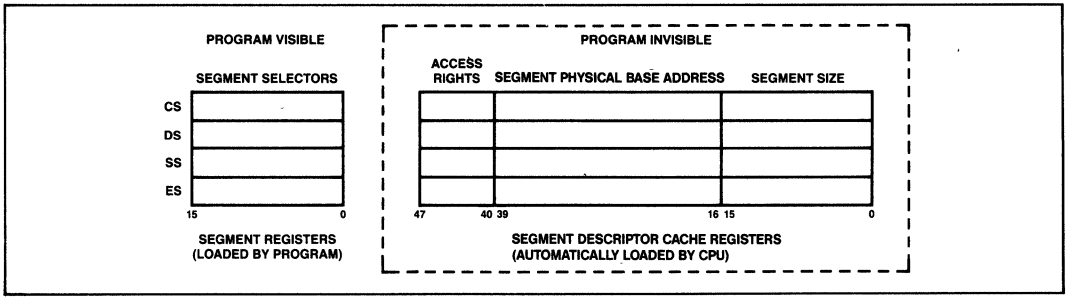


Figure 14. Descriptor Cache Registers

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.

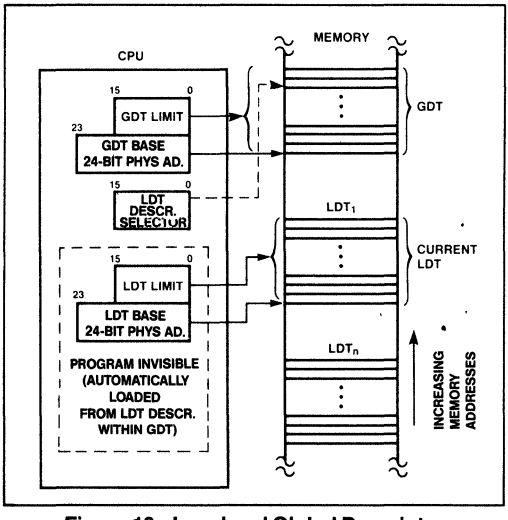


Figure 16. Local and Global Descriptor Table Definition

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.

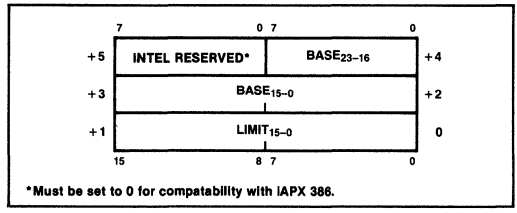


Figure 17. Global Descriptor Table and Interrupt Descriptor Table Data Type

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).

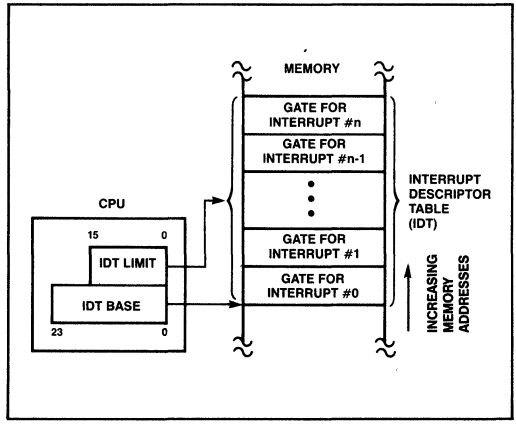


Figure 18. Interrupt Descriptor Table Definition

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

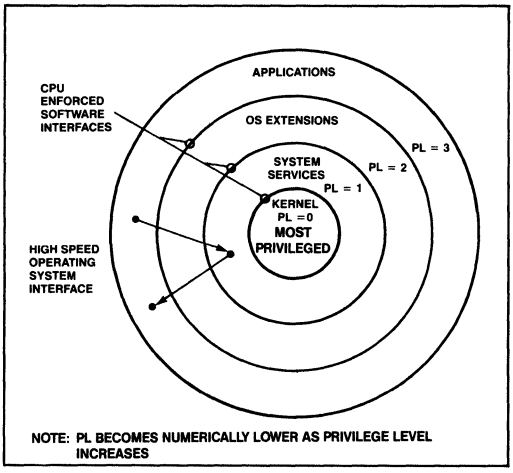


Figure 19. Hierarchical Privilege Levels

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.

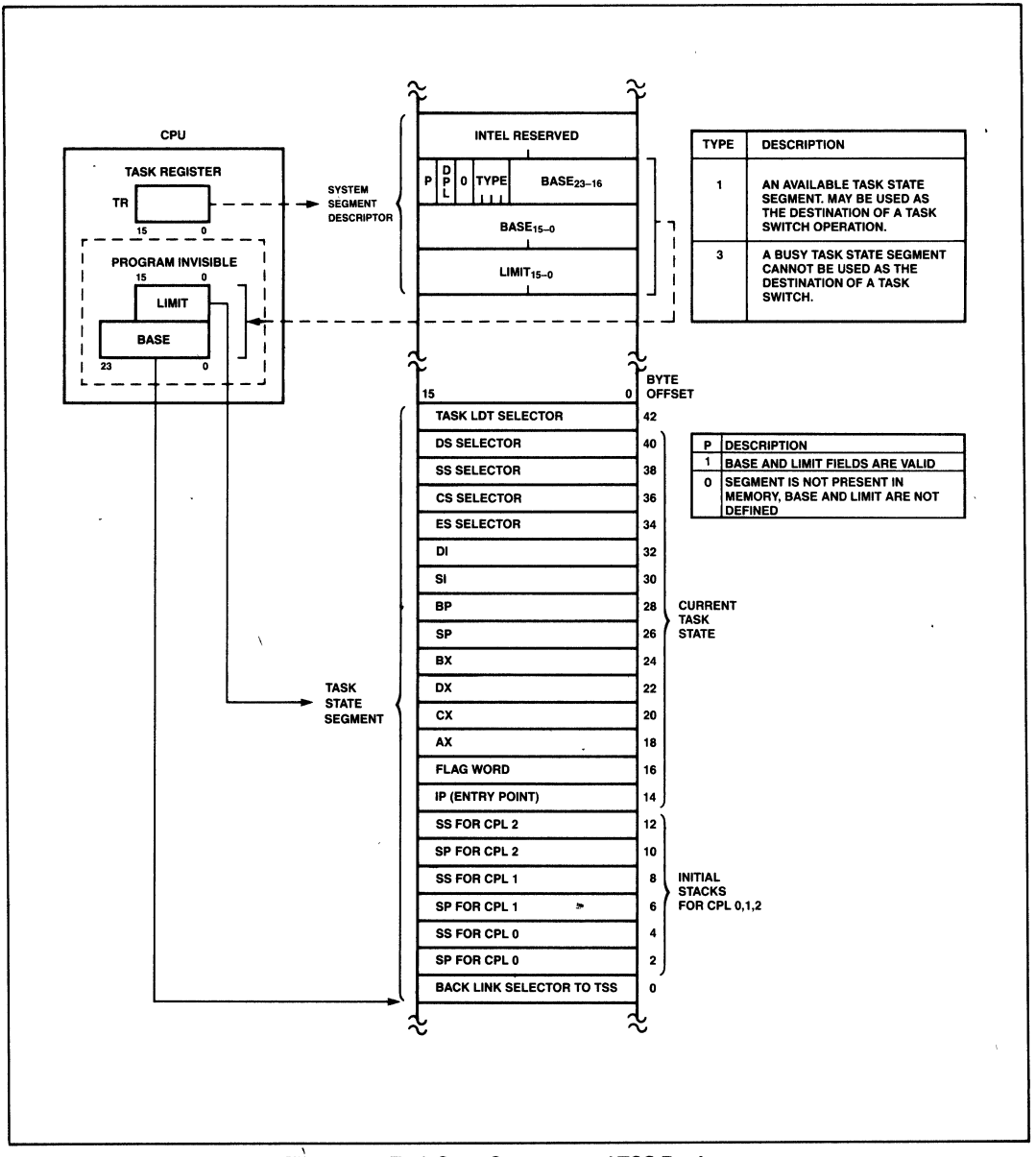


Figure 20. Task State Segment and TSS Registers

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 $\overline{\text{READY}}$ 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 $\overline{\text{BHE}}$, 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.

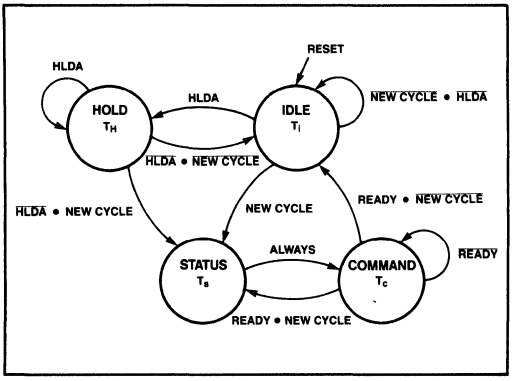


Figure 22. 80286 Bus States

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.)

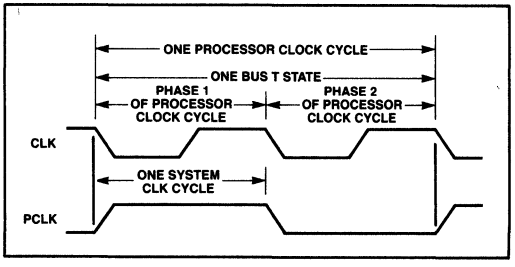


Figure 21. System and Processor Clock Relationships

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-

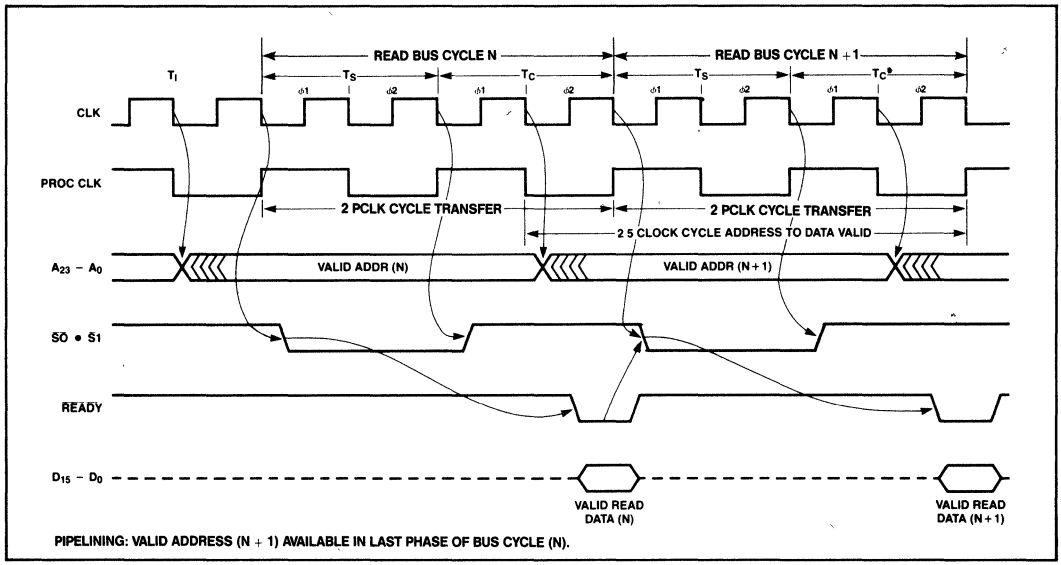


Figure 23. Basic Bus Cycle

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/$\overline{R}$.

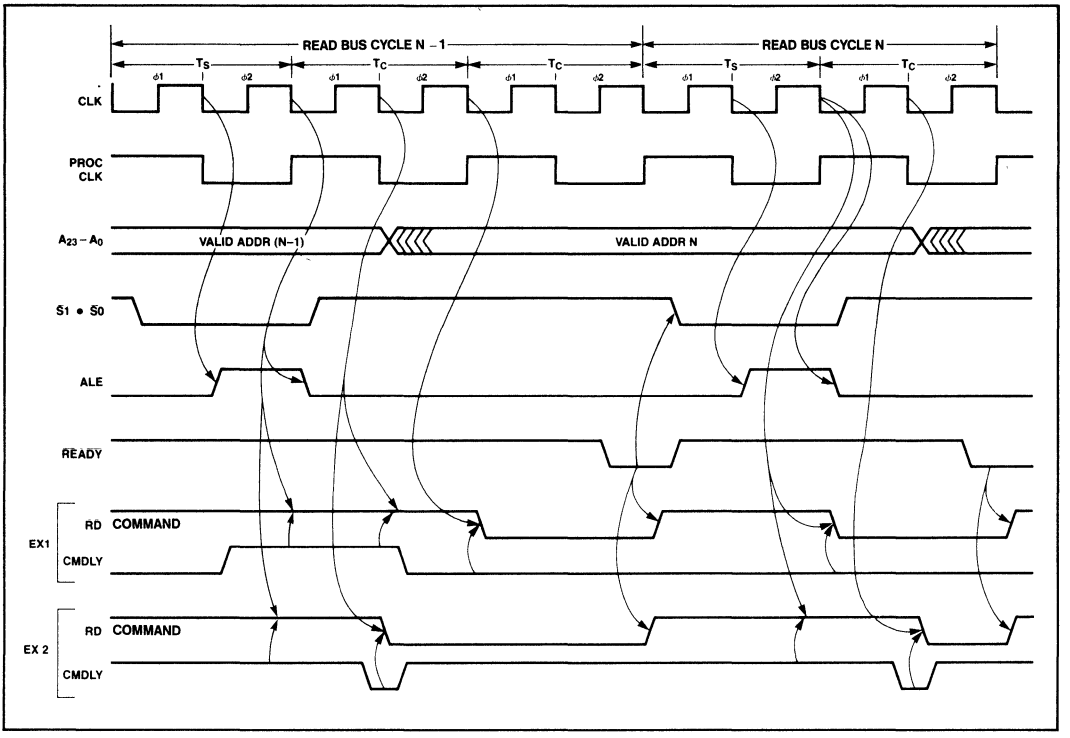


Figure 24. CMDLY Controls the Leading Edge of Command Signal.

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.

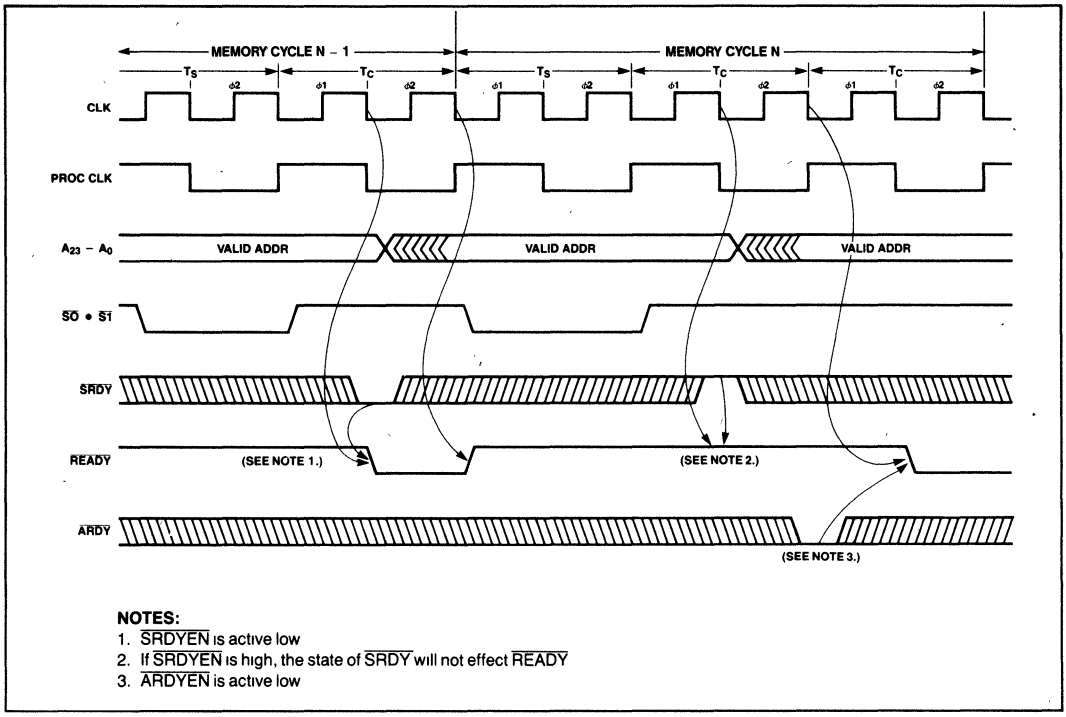


Figure 25. Synchronous and Asynchronous Ready

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.

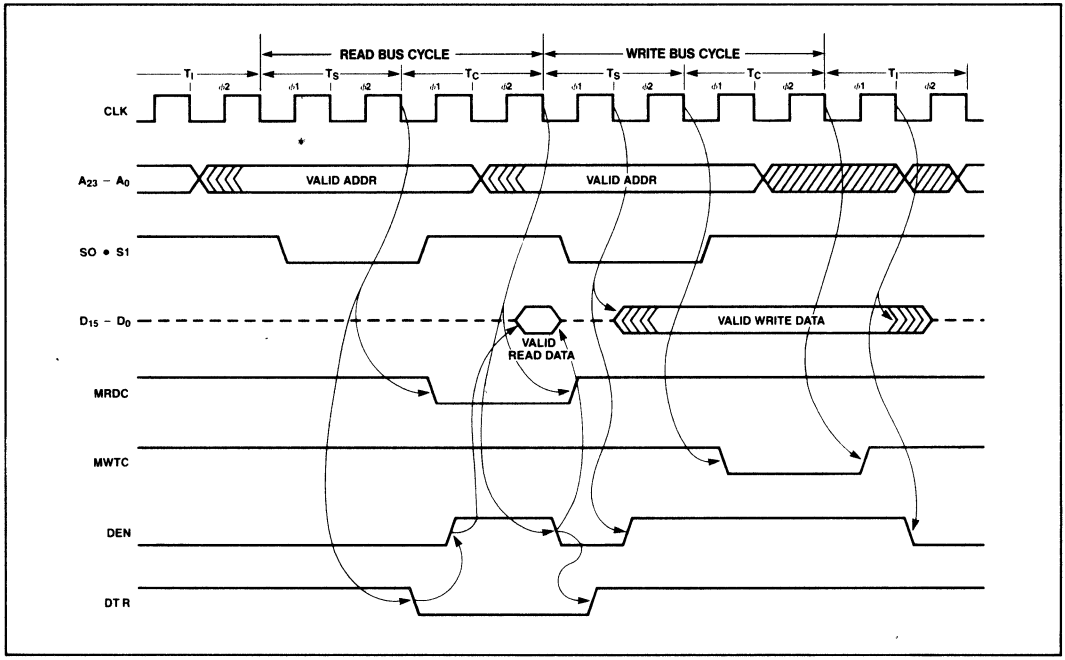


Figure 26. Back to Back Read-Write Cycles

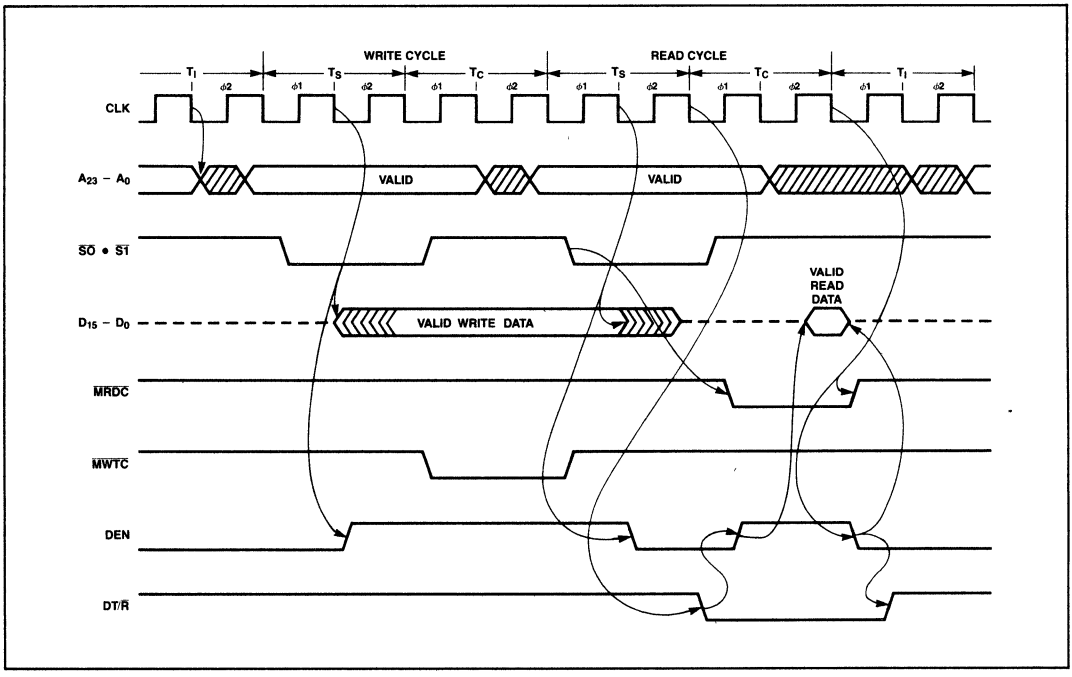


Figure 27. Back to Back Write-Read Cycles

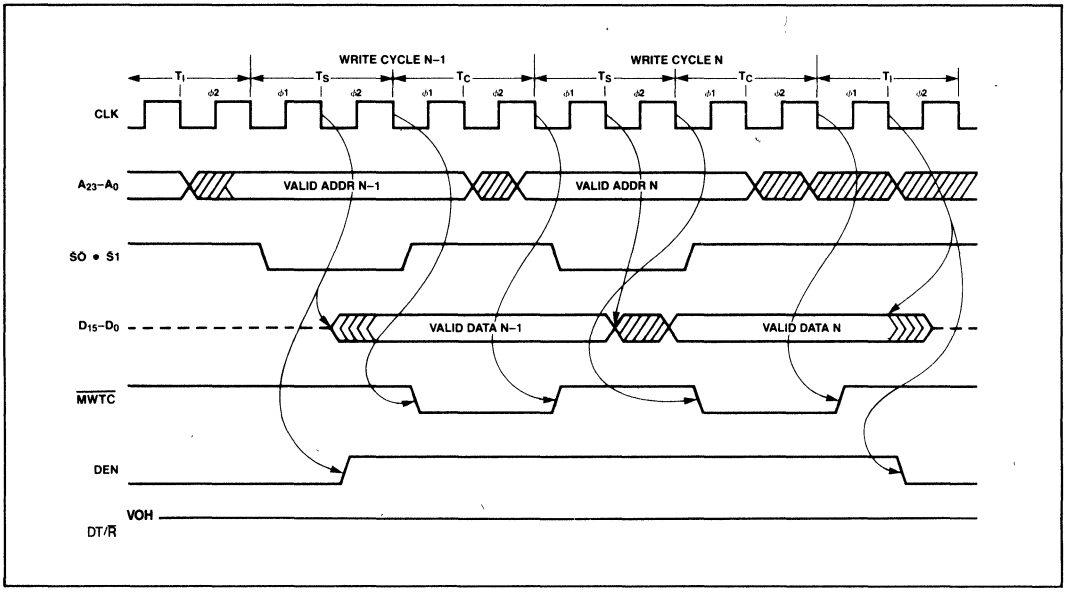


Figure 28. Back to Back Write-Write Cycles

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.

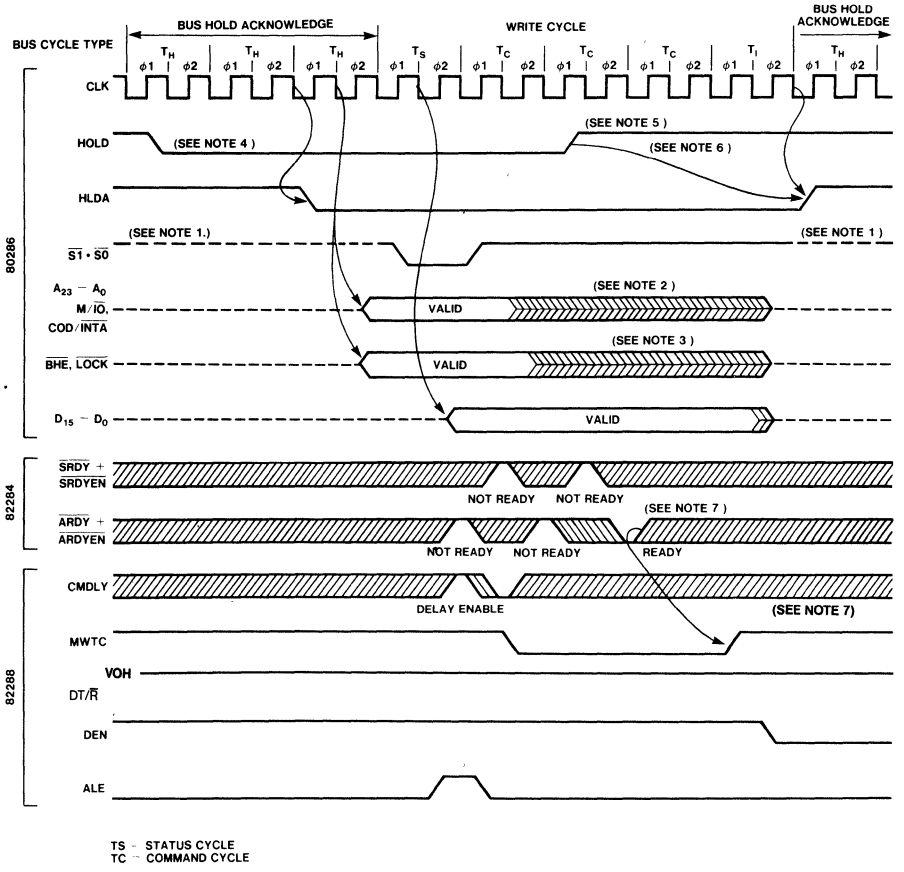
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

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 ϕ2 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 ϕ1 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.

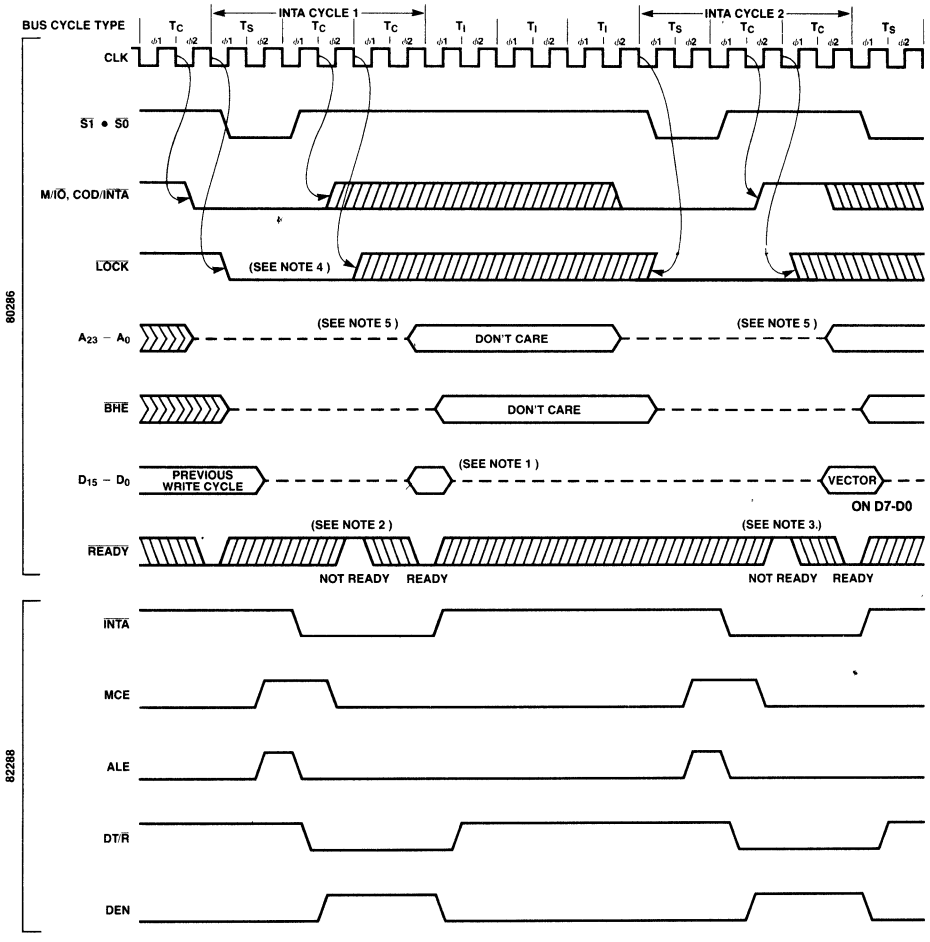
Data transfer performed by EU as part of an instruction.

(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

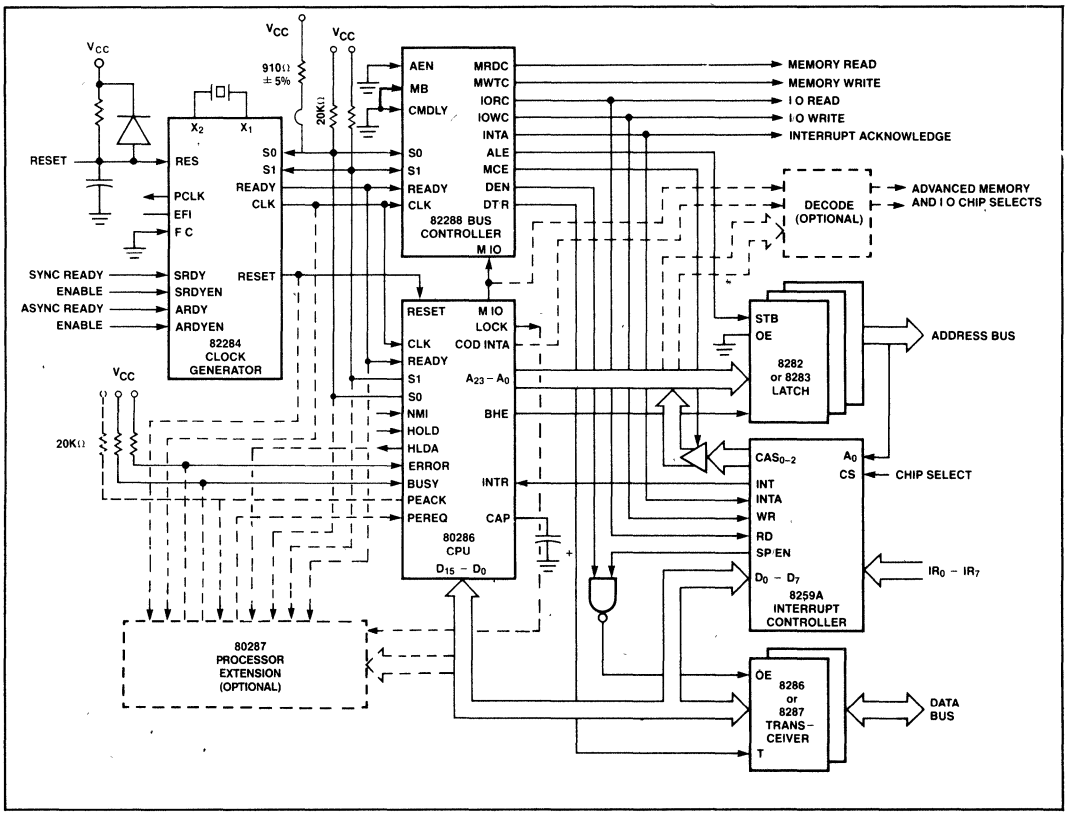


Figure 31. Basic iAPX 286 System Configuration

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

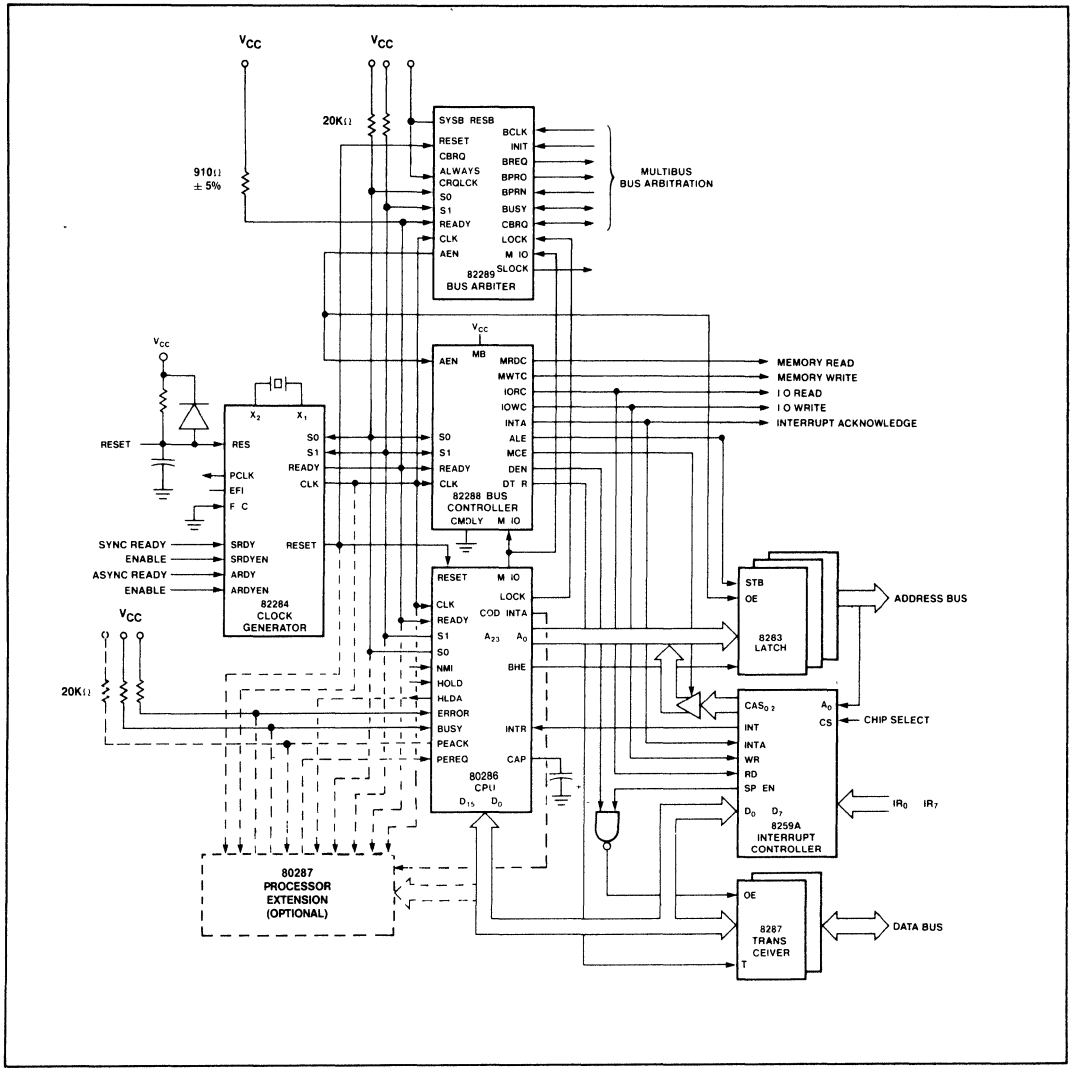


Figure 32. Multibus System Bus Interface

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/$\overline{\text{INTA}}$ and M/$\overline{\text{IO}}$ 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.

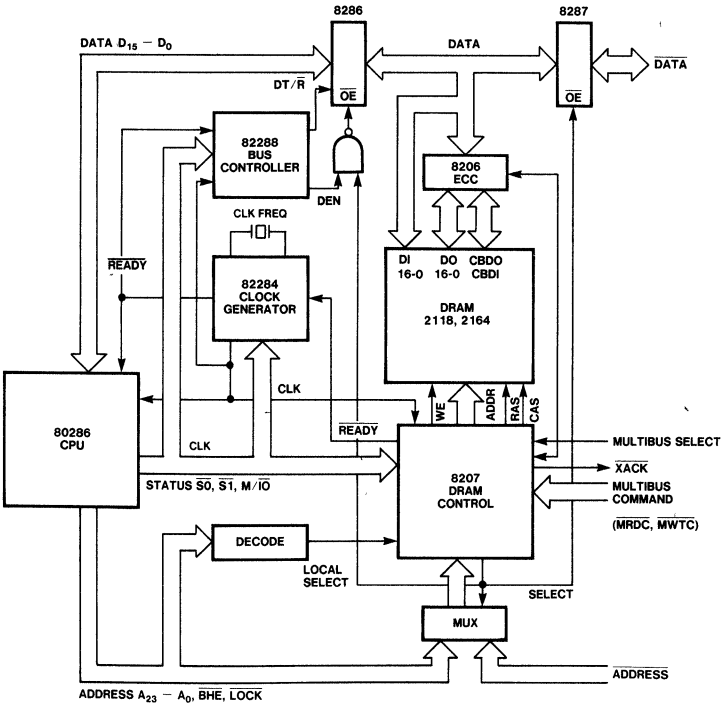


Figure 33. iAPX 286 System Configuration with Dual-Ported Memory

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.

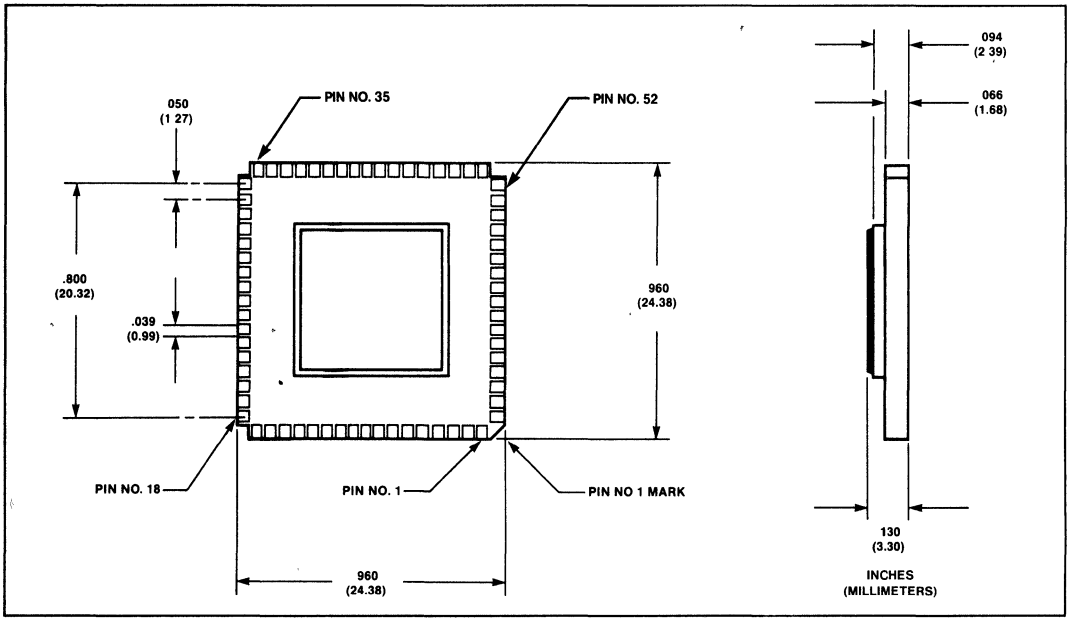


Figure 34. JEDEC Type A Package

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 . . . . . 3.3 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°C to 70°C, V_{CC} = 5V, ±5%)

		4 MHz		6MHz		8MHz			
Sym	Parameter	-4 Min	-4 Max	-6 Min	-6 Max	-8 Min	-8 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_{IL}	Input sustaining Current on $\overline{BUSY}$ and $\overline{ERROR}$ pins	30	500	30	500	30	500	μA	V_{IN} = OV
I_{LO}	Output Leakage Current	.	+-10	.	+-10	.	+-10	μA	$45V \le V_{OUT} \le V_{CC}$
I_{LO}	Output Leakage Current	.	+-1	.	+-1	.	+-1	mA	$0V \le V_{OUT} < .45V$
I_{CC}	Supply Current (turn on, 0°C)	.	600	.	600	.	600	mA	Note 1

NOTE 1: Low temperature is worst case

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

		4 MHz		6MHz		8MHz			
Sym	Parameter	-4 Min	-4 Max	-6 Min	-6 Max	-8 Min	-8 Max	Unit	Test Condition
C_{CLK}	CLK Input Capacitance		20		20		20	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.

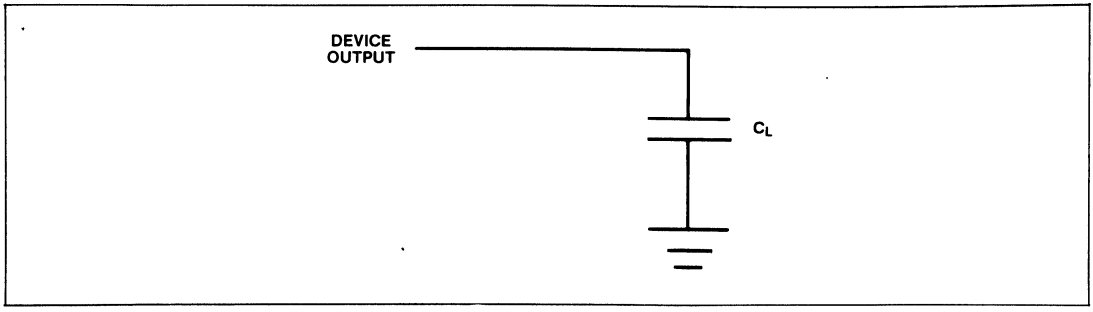
		4 MHz		6MHz		8MHz			
Sym	Parameter	-4 Min	-4 Max	-6 Min	-6 Max	-8 Min	-8 Max	Unit	Test Condition
1	System Clock (CLK) Period	124	250	83	250	62	250	ns	
2	System Clock (CLK) LOW Time	30	210	20	225	15	225	ns	at 1.0V
3	System Clock (CLK) HIGH Time	40	220	25	230	25	235	ns	at 3.6V
17	System Clock (CLK) Rise Time		10		10		10	ns	1.0V to 3.6V
18	System Clock (CLK) Fall Time		10		10		10	ns	3.6V 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		33		28		ns	
7	RESET Hold Time	5		5		5		ns	
8	Read Data Setup Time	30		20		10		ns	
9	Read Data Hold Time	8		8		8		ns	
10	$\overline{\text{READY}}$ Setup Time	75		50		38		ns	
11	$\overline{\text{READY}}$ Hold Time	50		35		25		ns	
12	Status/$\overline{\text{PEACK}}$ Valid Delay	1	80	1	55	1	40	ns	Note 2 Note 3
13	Address Valid Delay	1	120	1	80	1	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	50	ns	Note 2 Note 4
16	HLDA Valid Delay	0	120	0	80	0	50	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

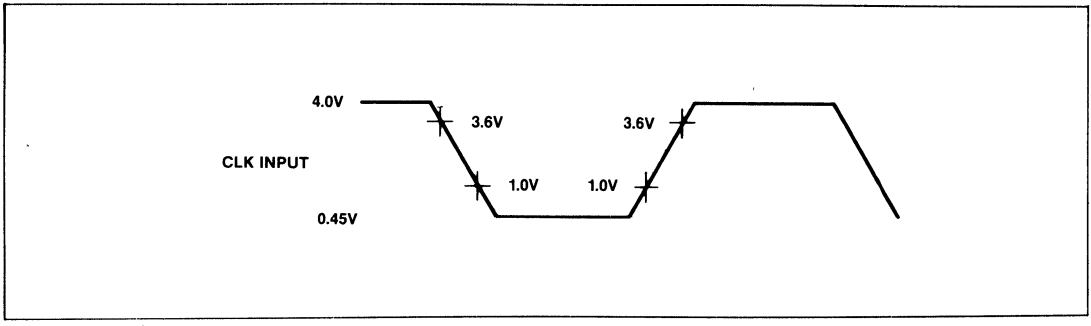
NOTE 2: Delay from 0.8V on the CLK, to 0.8V or 2.0V or float on the output as appropriate for valid or floating condition

NOTE 3: Ouput load C_L = 100pF

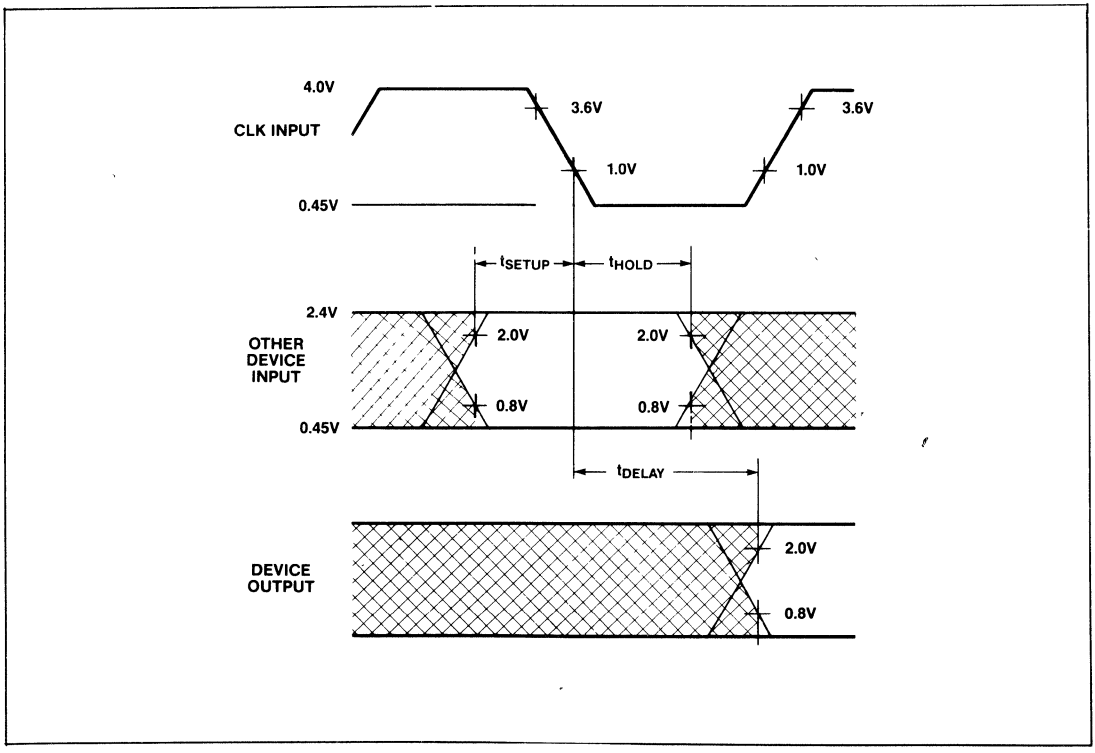
NOTE 4: Float condition occurs when output current is less than I_{LO} in magnitude



NOTE 5: AC Test Loading on Outputs



NOTE 6: AC Drive and Measurement Points — CLK Input



NOTE 7. AC Setup, Hold and Delay Time Measurement — General

A.C. CHARACTERISTICS (Cont.)

82284 Timing Requirements

		82284-6		82284-8			
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**	**Test Conditions**
11	$\overline{SRDY}/\overline{SRDYEN}$ setup time	25		15		ns	
12	$\overline{SRDY}/\overline{SRDYEN}$ hold time	0		0		ns	
13	$\overline{ARDY}/\overline{ARDYEN}$ setup time	5		0		ns	See note 1
14	$\overline{ARDY}/\overline{ARDYEN}$ 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

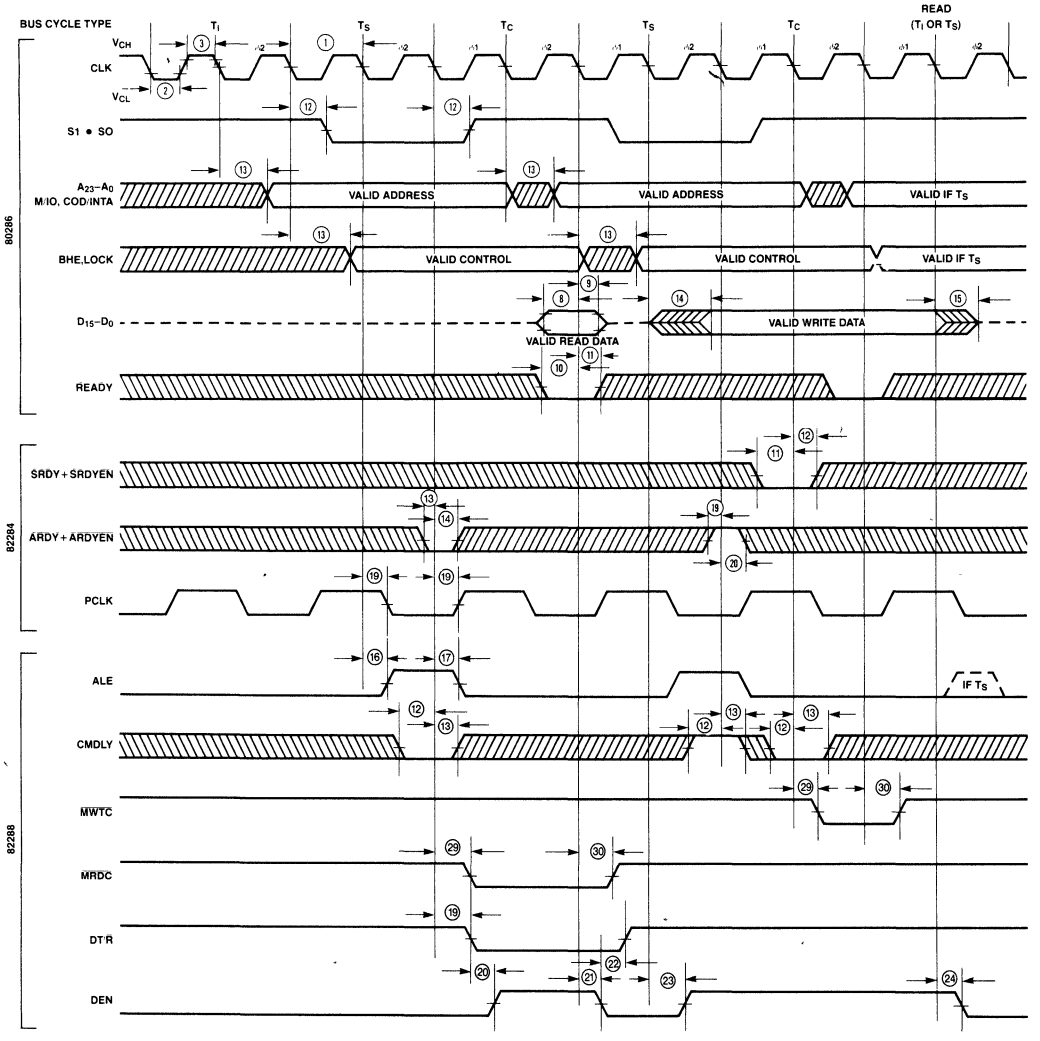
82288 Timing Requirements

			82288-6		82288-8			
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/$\overline{R}$ read active delay			40	0	20	ns	
22	DT/$\overline{R}$ read inactive delay		5	45	10	40	ns	C_L = 150 pF I_{OL} = 16 ma max I_{OH} = -1 ma max
20	DEN read active delay		10	50	10	40	ns	
21	DEN read inactive delay		3	40	3	15	ns	
23	DEN write active delay			35		30	ns	
24	DEN write inactive delay		3	35	3	30	ns	

NOTE 1: Asychronous inputs are INTR, NMI, HOLD, PEREQ, $\overline{ERROR}$, AND $\overline{BUSY}$ This specification is given only for testing purposes, to assure recognition at a specific CLK edge

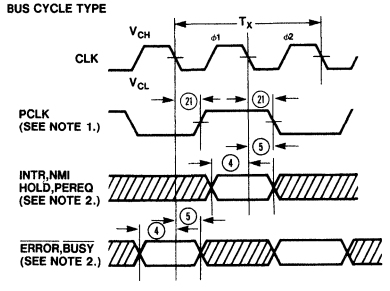
WAVEFORMS

MAJOR CYCLE TIMING



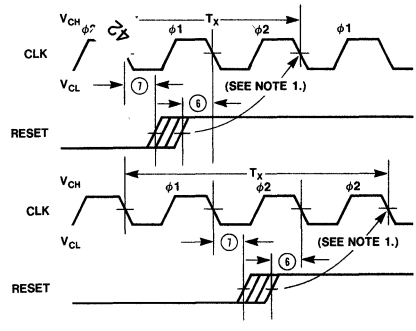
WAVEFORMS (Continued)

80286 ASYNCHRONOUS INPUT SIGNAL TIMING



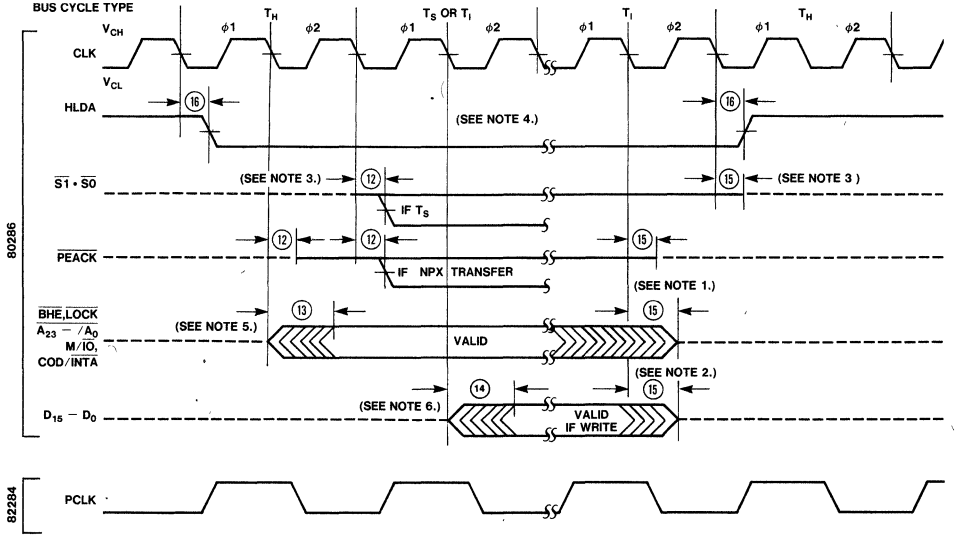
NOTES:
1. PCLK indicates which processor cycle phase will occur on the next CLK. PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

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.

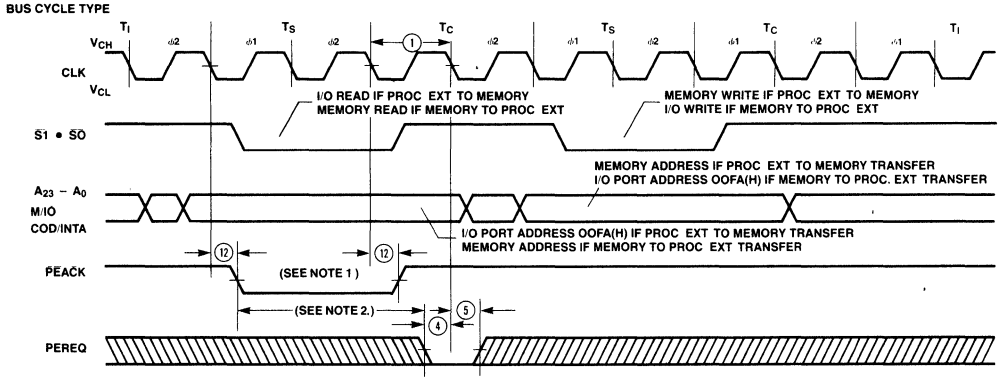
EXITING AND ENTERING HOLD



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)

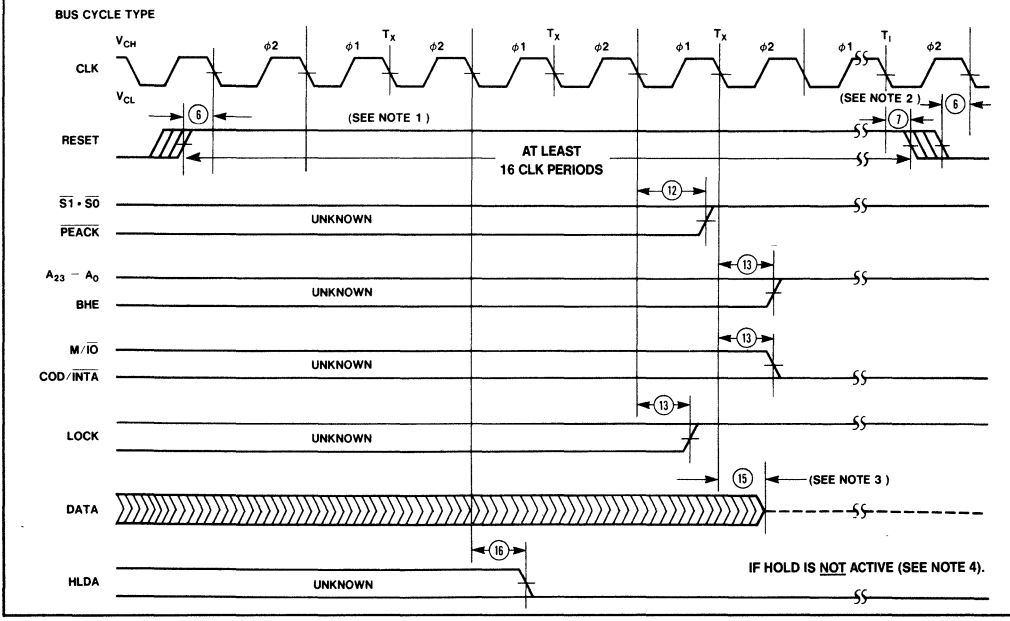
80286 PEREQ/PEACK TIMING FOR ONE TRANSFER ONLY



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· $3X(1) - (11)_{max} - (4)_{min} + A X 2 X(1)$.
   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 ⌀1 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 ⌀1 or ⌀2
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

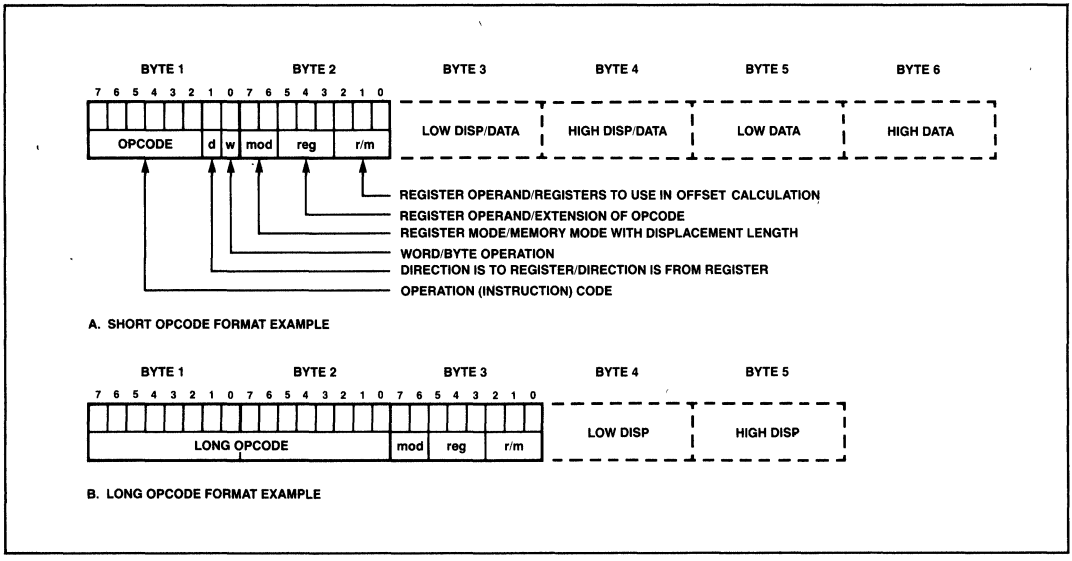


Figure 35. 80286 Instruction Format Examples

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		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	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 = 01	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 = 01	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 = 01	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 = 01	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 = 01	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			2,8	8,9
L=0		11	11	2,8	8,9
L=1		15	15	2,8	8,9
L>1		16+4(L−1)	16+4(L−1)	2,8	8,9
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1	5	5		
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	6,8,9,11,12,18

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
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.

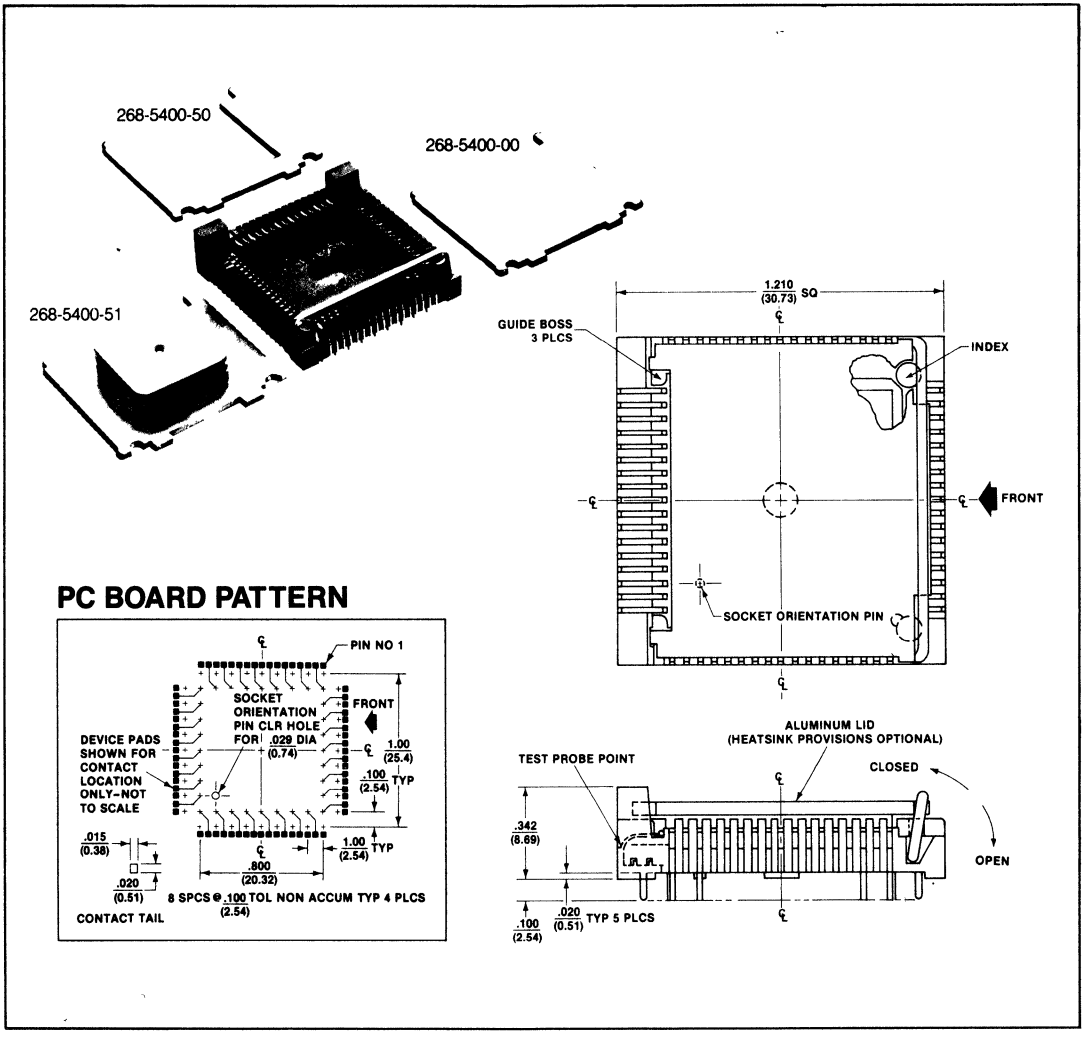


Figure 36. Textool 68 Lead Chip Carrier Socket

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.

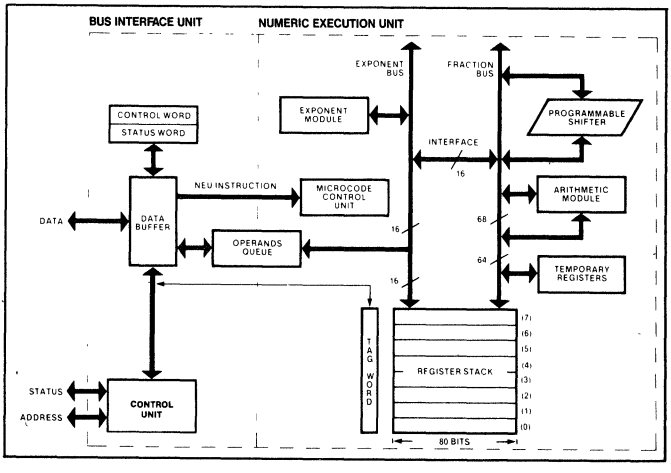
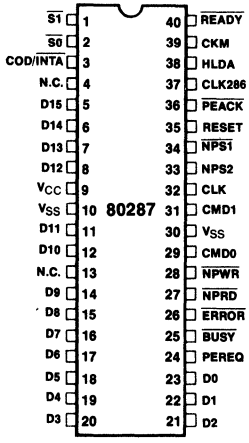


Figure 1. 80287 Block Diagram



NOTE:
N.C. PINS MUST NOT BE CONNECTED.

Figure 2. 80287 Pin Configuration

January 1985
Order Number: 210920-003

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.

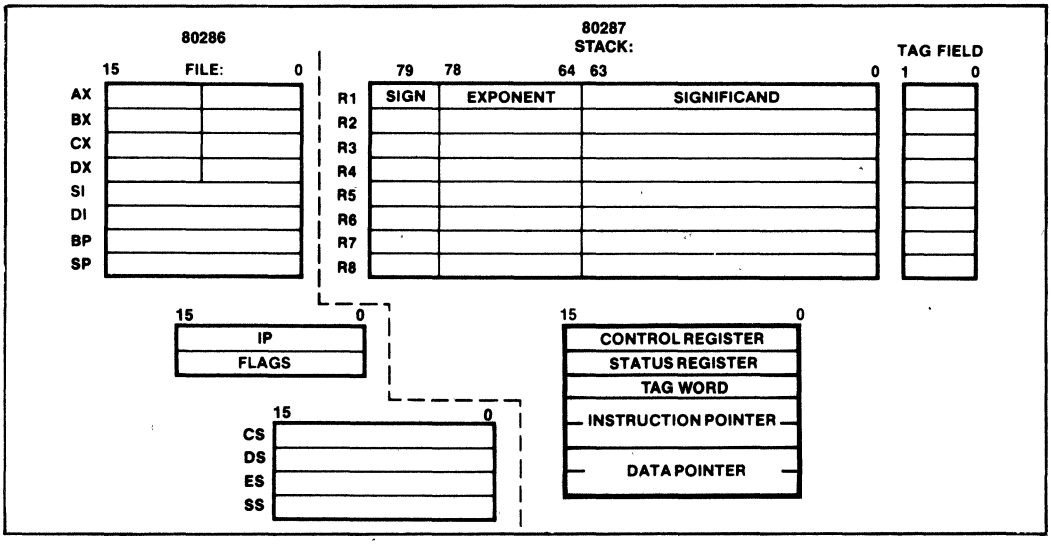


Figure 3. iAPX 286/20 Architecture

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

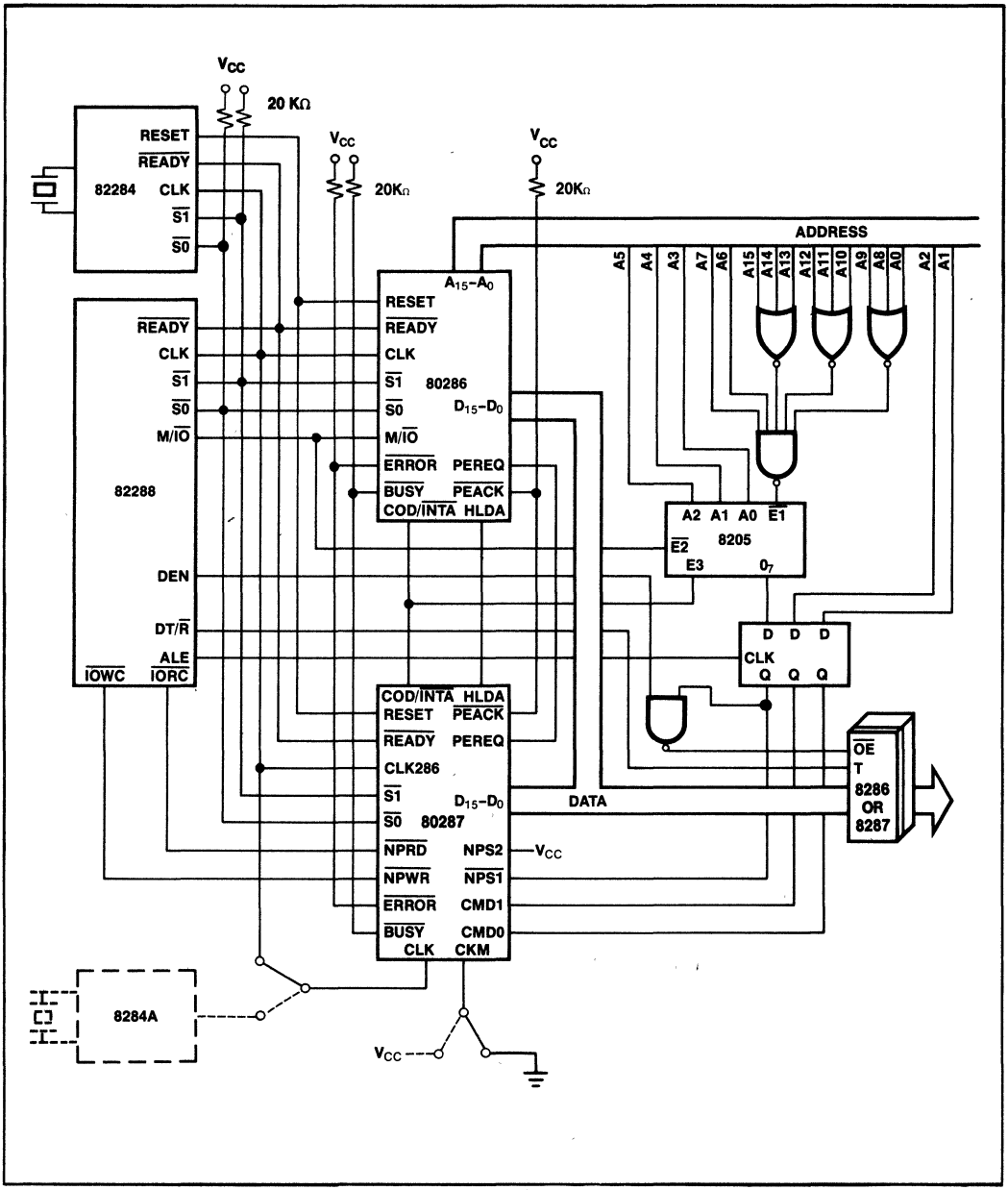
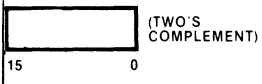
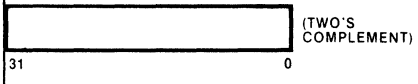
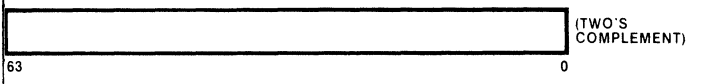
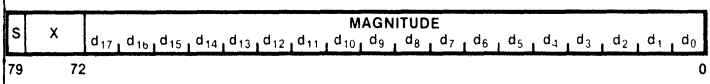
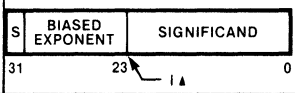
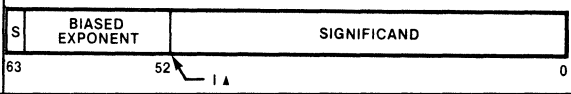
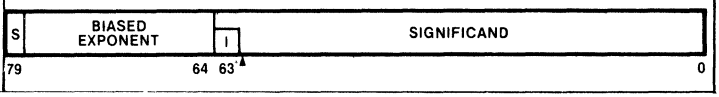


Figure 4. iAPX 286/20 System Configuration

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

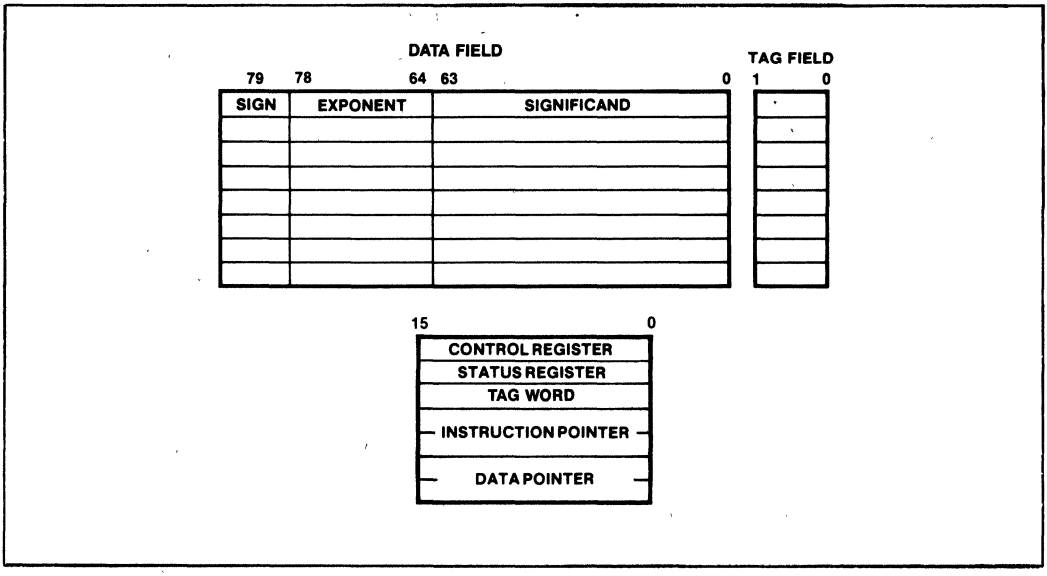


Figure 5. 80287 Register Set

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.

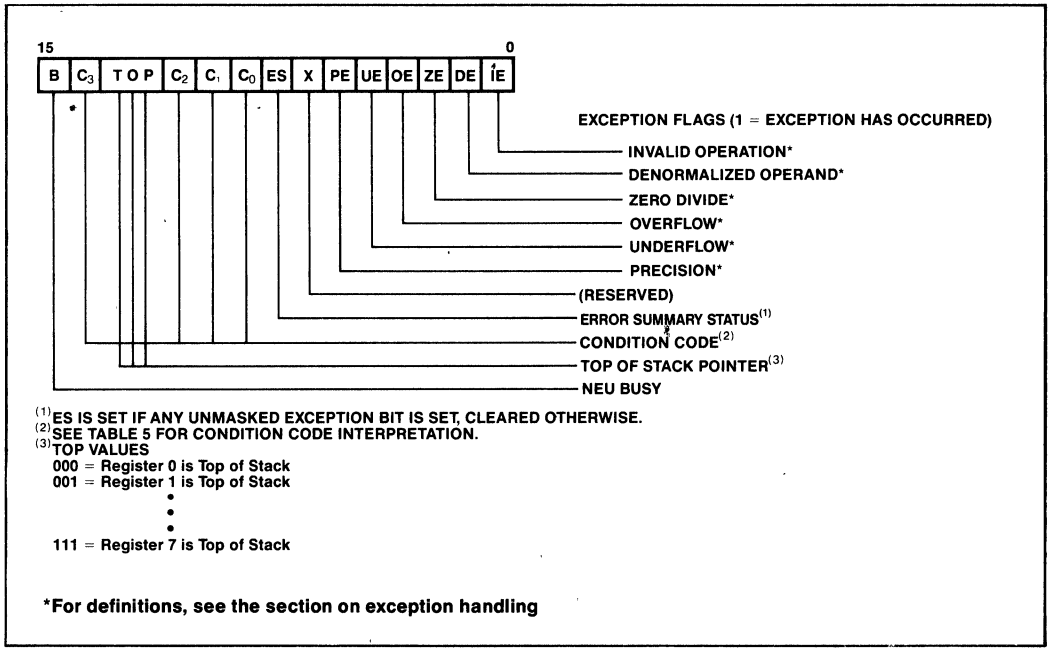


Figure 6. 80287 Status Word

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).

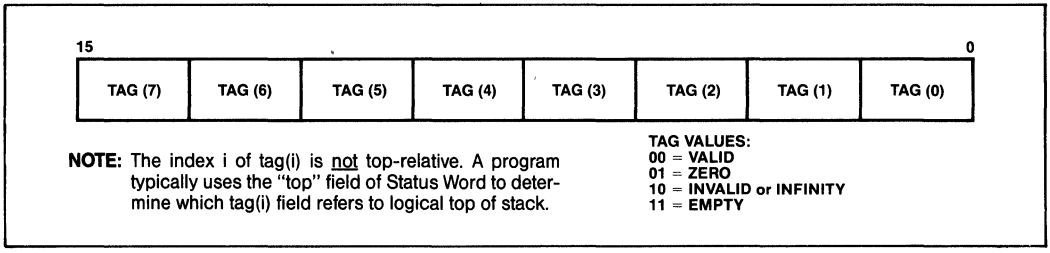


Figure 7. 80287 Tag Word

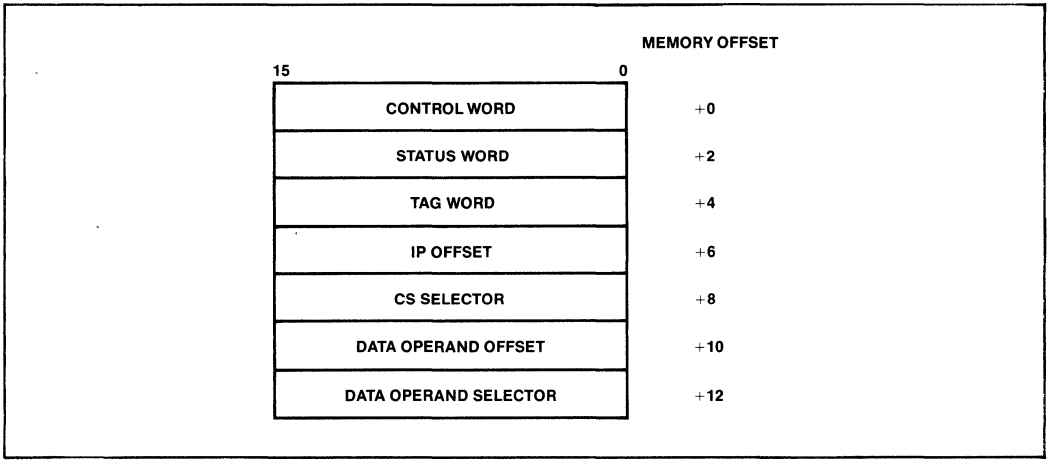


Figure 8a. Protected Mode 80287 Instruction and Data Pointer Image in Memory

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.

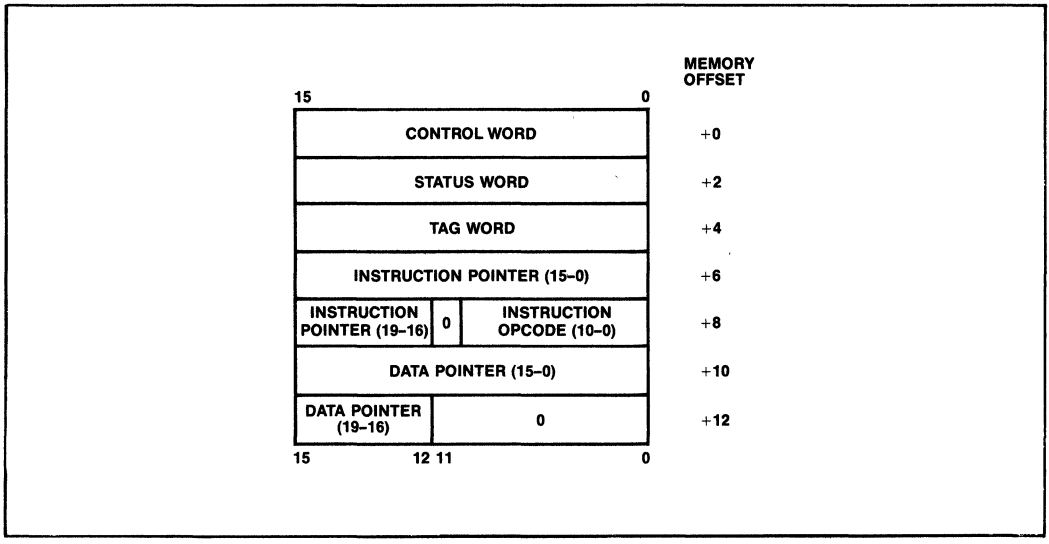


Figure 8b. Real Mode 80287 Instruction and Data Pointer Image in Memory

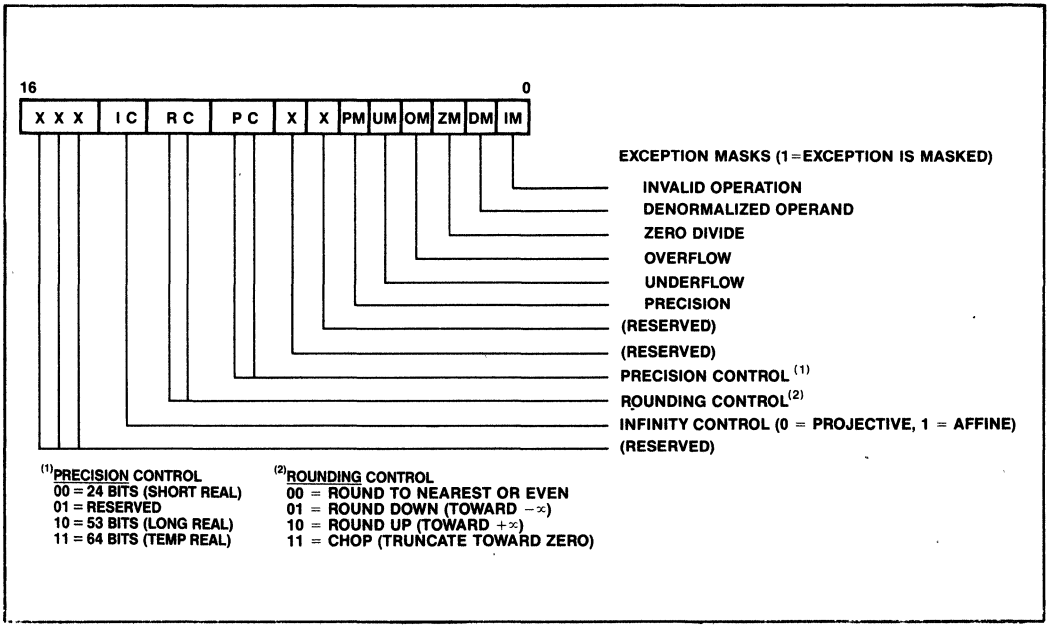


Figure 9. 80287 Control Word

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	$OV \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current	.	±10	μA	$.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	Power Supply Current	.	600	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		8 MHz			
Symbol	**Parameter**	**-3 Min**	**-3 max**	**Min**	**max**	**Unit**	**Test Conditions**
T_{CLCL}	CLK Period CKM = 1: CKM = 0:	 200 62.5	 500 166	 125 62.5	 500 166	 ns ns	
T_{CLCH}	CLK LOW Time CKM = 1: CKM = 0:	 118 15	 343 146	 68 15	 343 146	 ns ns	 At 0.8V At 0.6V
T_{CHCL}	CLK HIGH Time CKM = 1: CKM = 0:	 69 20	 230 151	 43 20	 230 151	 ns ns	 At 2.0V At 3.8V
T_{CH1CH2}	CLK Rise Time	.	10	.	10	ns	1.0V to 3.6V if CKM = 1
T_{CL2CL1}	CLK Fall Time	.	10	.	10	ns	3.6V to 1.0V if CKM = 1
T_{DVWH}	Data Setup to $\overline{NPWR}$ Inactive	75	.	75	.	ns	
T_{WHDX}	Data Hold from $\overline{NPWR}$ Inactive	30	.	15	.	ns	
T_{WLWH}, T_{RLRH}	$\overline{NPWR}$, $\overline{NPRD}$ Active Time	95	.	90	.	ns	At 0.8V
T_{AVRL}, T_{AVWL}	Command Valid to $\overline{NPWR}$ or $\overline{NPRD}$ Active	0	.	0	.	ns	
T_{MHRL}	Minimum Delay from PEREQ Active to $\overline{NPRD}$ Active	130	.	130	.	ns	
T_{KLKH}	$\overline{PEACK}$ Active Time	85	.	85	.	ns	At 0.8V
T_{KHKL}	$\overline{PEACK}$ Inactive Time	250	.	250	.	ns	At 2.0V
T_{KHCH}	$\overline{PEACK}$ Inactive to $\overline{NPWR}$, $\overline{NPRD}$ Inactive	50	.	40	.	ns	
T_{CHKL}	$\overline{NPWR}$, $\overline{NPRD}$ Inactive to $\overline{PEACK}$ Active	-30	.	-30	.	ns	
T_{WHAX}, T_{RHAX}	Command Hold from $\overline{NPWR}$, $\overline{NPRD}$ Inactive	30	.	30	.	ns	
T_{KLCL}	$\overline{PEACK}$ Active Setup to $\overline{NPWR}$, $\overline{NPRD}$ Active	50	.	40	.	ns	
T_{2CLCL}	CLK286 Period	62.5	.	62.5	.	ns	
T_{2CLCH}	CLK286 LOW Time	15	.	15	.	ns	At 0.8V
T_{2CHCL}	CLK286 HIGH Time	20	.	20	.	ns	At 2.0V
T_{2SVCL}	$\overline{SO}$, $\overline{ST}$ Setup Time to CLK286	22	.	22	.	ns	
T_{2CLSH}	$\overline{SO}$, $\overline{ST}$ Hold Time from CLK286	0	.	0	.	ns	

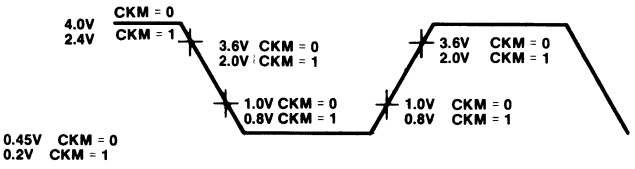
A.C. CHARACTERISTICS, continued
TIMING REQUIREMENTS

		5 MHz		8 MHz			
Sym	**Parameter**	**-3 Min**	**-3 Max**	**Min**	**Max**	**Unit**	**Test Conditions**
T_{CIVCL}	COD/$\overline{INTA}$ Setup Time to CLK286	0		0		ns	
T_{CLCIH}	COD/$\overline{INTA}$ Hold Time from CLK286	0		0		ns	
T_{RVCL}	$\overline{READY}$ Setup Time to CLK286	38.5		38.5		ns	
T_{CLRH}	$\overline{READY}$ Hold Time from CLK286	25		25		ns	
T_{HVCL}	HLDA Setup Time to CLK286	0		0		ns	
T_{CLHH}	HLDA Hold Time from CLK286	0		0		ns	
T_{IVCL}	$\overline{NPWR}$, $\overline{NPRD}$ to CLK Setup Time	70		70		ns	NOTE 1
T_{CLIH}	$\overline{NPWR}$, $\overline{NPRD}$ from CLK Hold Time	45		45		ns	NOTE 1
T_{RSCL}	RESET to CLK Setup Time	20		20		ns	NOTE 1
T_{CLRS}	RESET from CLK Hold Time	20		20		ns	NOTE 1

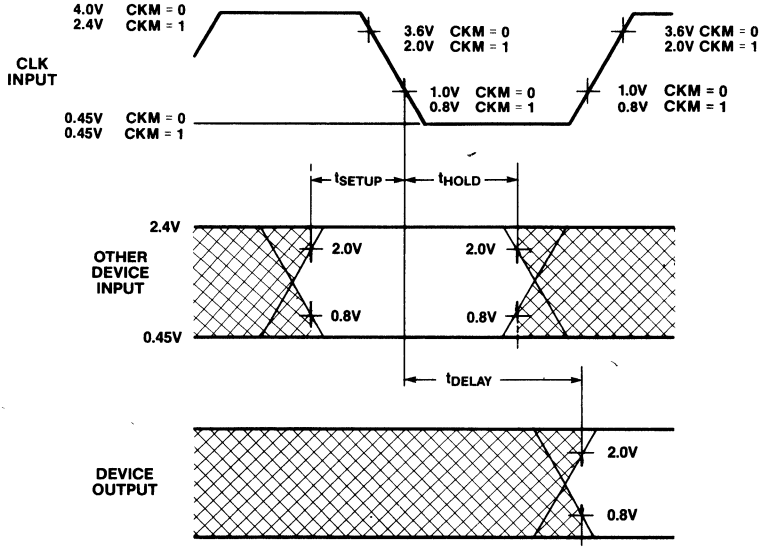
A.C. CHARACTERISTICS,
TIMING RESPONSES

		5 MHz		8 MHz			
Sym	**Parameter**	**-3 Min**	**-3 Max**	**Min**	**Max**	**Unit**	**Test Condition**
T_{RHQZ}	$\overline{NPRD}$ Inactive to Data Float		37.5		35	ns	NOTE 2
T_{RLQV}	$\overline{NPRD}$ Active to Data Valid		60		60	ns	NOTE 3
T_{ILBH}	$\overline{ERROR}$ Active to $\overline{BUSY}$ Inactive	100		100		ns	NOTE 4
T_{WLBV}	$\overline{NPWR}$ Active to $\overline{BUSY}$ Active		100		100	ns	NOTE 5
T_{KLML}	$\overline{PEACK}$ Active to $\overline{PEREQ}$ Inactive		127		127	ns	NOTE 6
T_{CMDI}	Command Inactive Time						
	Write-to-Write	95		95		ns	At 2.0V
	Read-to-Read	250		325		ns	At 2.0V
	Write-to-Read	105		95		ns	At 2.0V
	Read-to-Write	95		95		ns	At 2.0V
T_{RHQH}	Data Hold from $\overline{NPRD}$ Inactive	5		5		ns	NOTE 7

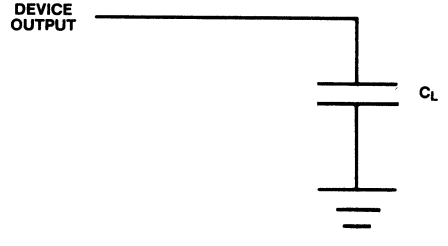
NOTES:
1. This is an asynchronous input. This specification is given for testing purposes only, to assure recognition at a specific CLK edge.
2. Float condition occurs when output current is less than I_{LO} on D0-D15.
3. D0-D15 loading: CL = 100pF
4. $\overline{BUSY}$ loading: CL = 100pF
5. $\overline{BUSY}$ loading: CL = 100pF
6. On last data transfer of numeric instruction.
4. D0-D15 loading: CL = 100pF



NOTE 8: AC Drive and Measurement Points — CLK Input



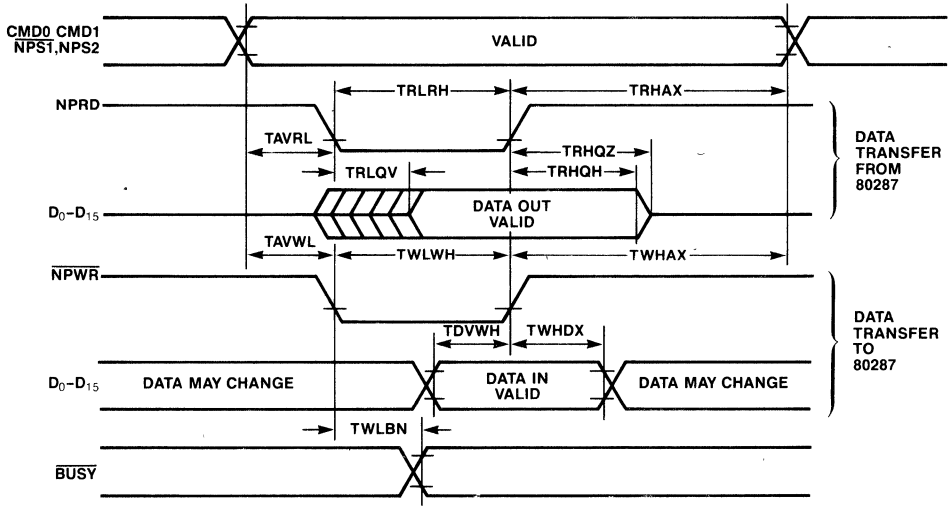
NOTE 9: AC Setup, Hold and Delay Time Measurement — General



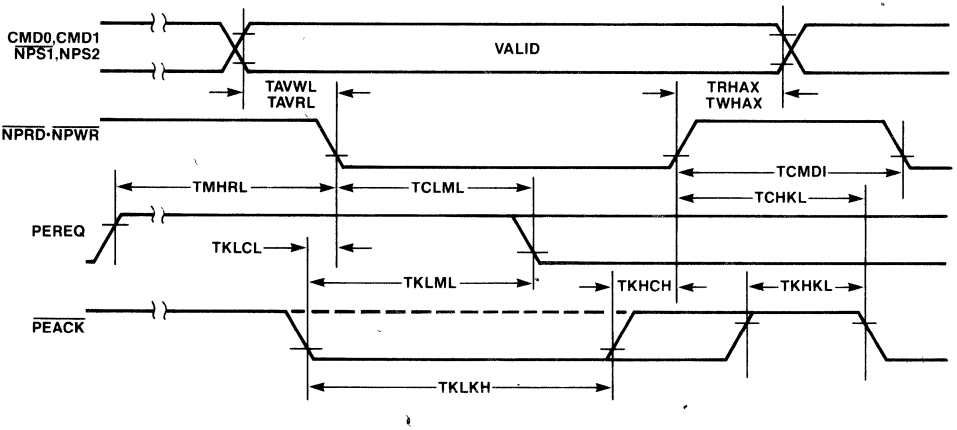
NOTE 10: AC Test Loading on Outputs

WAVEFORMS (cont.)

DATA TRANSFER TIMING (INITIATED BY 80286)

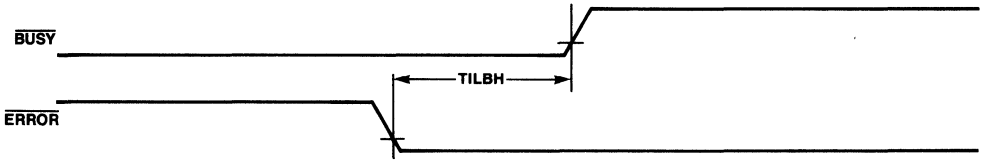


DATA CHANNEL TIMING (INITIATED BY 80287)

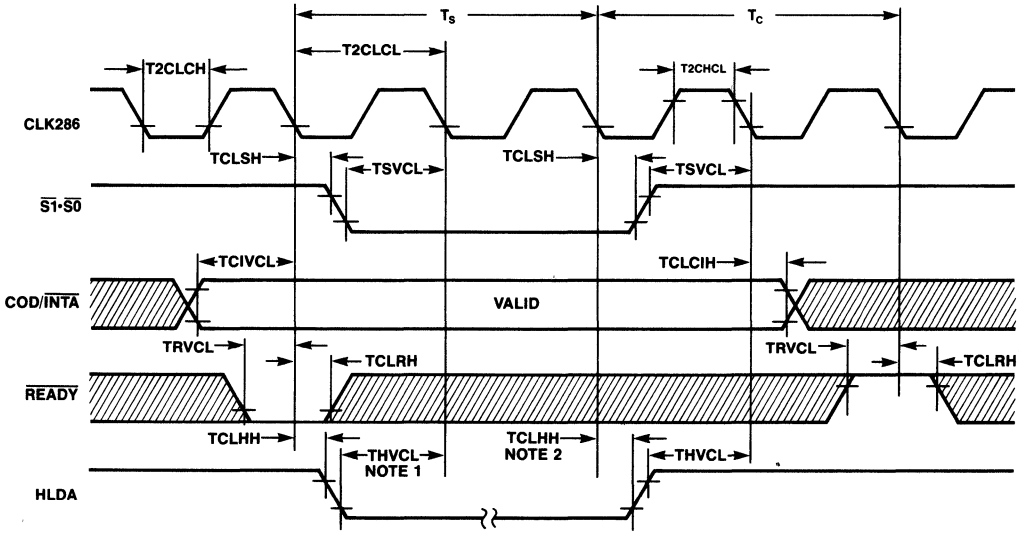


WAVEFORMS (cont.)

ERROR OUTPUT TIMING



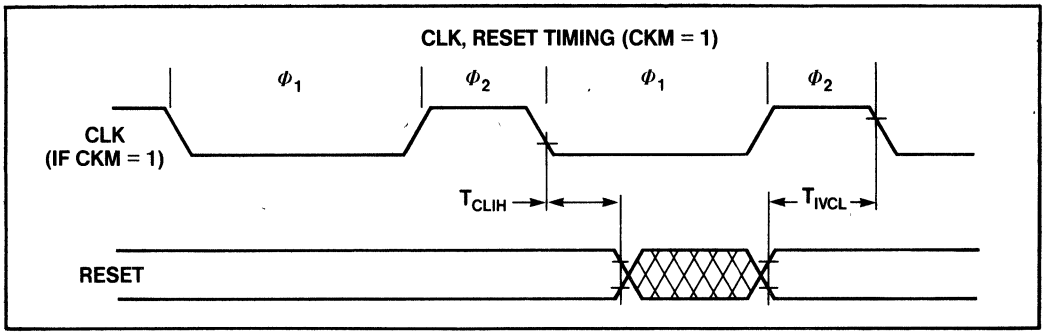
80286 STATUS TIMING

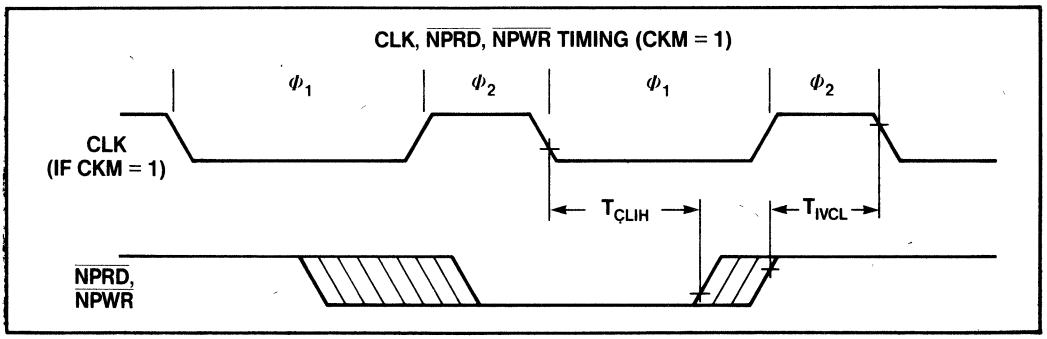


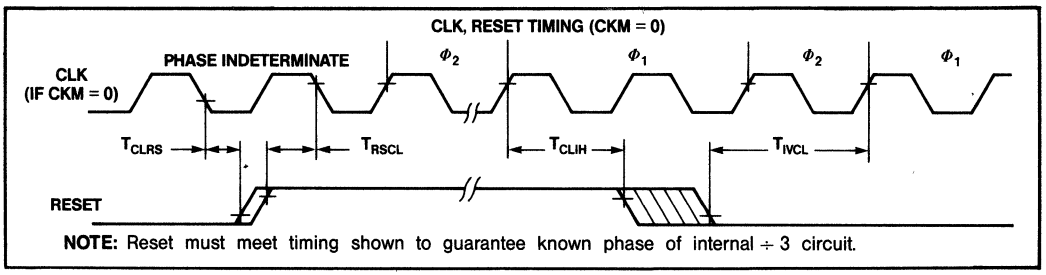
NOTES:
1. This input transition occurs before T_S.
2. This input transition occurs after T_C.

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.

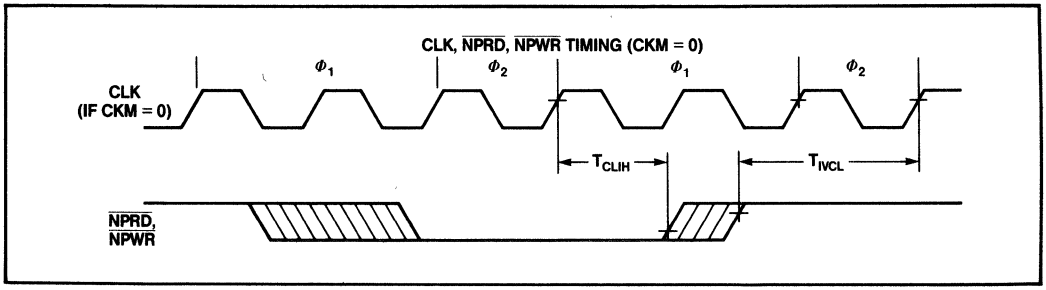


Table 6. 80287 Extensions to the 80286 Instruction Set

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.

intel

June 1984

82258
Advanced DMA Controller
Architectural Overview

Order Number: 230606-002

INTRODUCTION

As the processing of microprocessor based systems grows, it is increasingly necessary to have support components which relieve the processor from jobs like data movement, peripheral control, etc., and leave it to do what it can do best—data processing. Among the support components necessary is a DMA (Direct Memory Access) controller. A DMA controller must match the performance and architectural needs of the processor it supports to optimize the performance.

The 82258 is a 16-bit DMA controller designed primarily to meet the needs of systems based on the 80286 and 80186 processors. The 80286 is the fastest 16-bit processor available commercially with more than five times the throughput of the standard 8086. The 80186, on the other hand, integrates the functions of a CPU board (CPU, Clock, DMA, Interrupt Control, Timer Counter, Chip Select) on one chip, replacing 15–20 IC's. The 82258 is designed to support this level of performance and integration by offering features generally found only in mainframe computer systems.

FEATURES

- 8 MBytes/second transfer rate in 8 MHz iAPX 286 Systems
- 4 independent channels—each channel having its own set of registers and control lines
- Multiplexor channel operation for up to 32 I/O subchannels
- 16 MByte physical addressing range
- Block size (byte count) up to 16 MBytes
- Command chaining
- Data chaining
- Adaptive bus interface for
  - — 80286
  - — 80186
  - — 80188
  - — 8086
  - — 8088 processors
- Automatic assembly-disassembly for dissimilar bus widths
- "On the fly" compare, verify and transiate operations during DMA
- Single and double bus cycle transfers
- Local (with CPU) and remote (stand-alone) modes of operation

FLEXIBLE BUS INTERFACE

Although the 82258 is primarily designed for the 80286, it fits equally well in 8086, 8088, 80188, and 80186 based systems. This is possible because of its adaptive bus interface. The logic level on a specific pin on RESET configures the bus interface of the 82258 for the 80286 (demultiplexed) bus, or for the 80186 or minimum mode 8086 (multiplexed) bus, with all the necessary signals and timing. In the 8086 mode, the 82258 can have RQ/GT signals for the maximum mode 8086 or 8088. This adaptive bus interface makes it very easy to use the 82258 in different processor systems. It is possible to select 8- or 16-bit wide bus operations by software. Figure 1 shows the pin configuration of the 82258 in the 286 and 186 modes.

The high performance (8 MB/sec) pipelined bus of the 286 allows the 82258 to provide maximum performance. This bus enables the 80286 and the 82258 to read or write a word (16 bits) in 250 ns, when operating at 8 MHz. Therefore, in the 286 mode, the 82258 can achieve data transfer rates of 8 MBytes/second. In the 186 and 8086 mode, the data transfer rate is 4 MBytes/second, since each bus cycle is 4 clocks or 500 ns at 8 MHz.

The 82258 has four (4) independent channels. Each channel has three (3) dedicated pins: DREQ (DMA request), DACK (DMA acknowledge) and EOD (End of DMA). See Figure 2. A peripheral generates request for DMA over the DREQ line. DACK is sent by the 82258 to the peripheral, indicating that a data transfer operation can begin. EOD is used by the 82258 to generate an interrupt on the completion of a DMA operation. The EOD line can also be used by a peripheral to terminate the DMA.

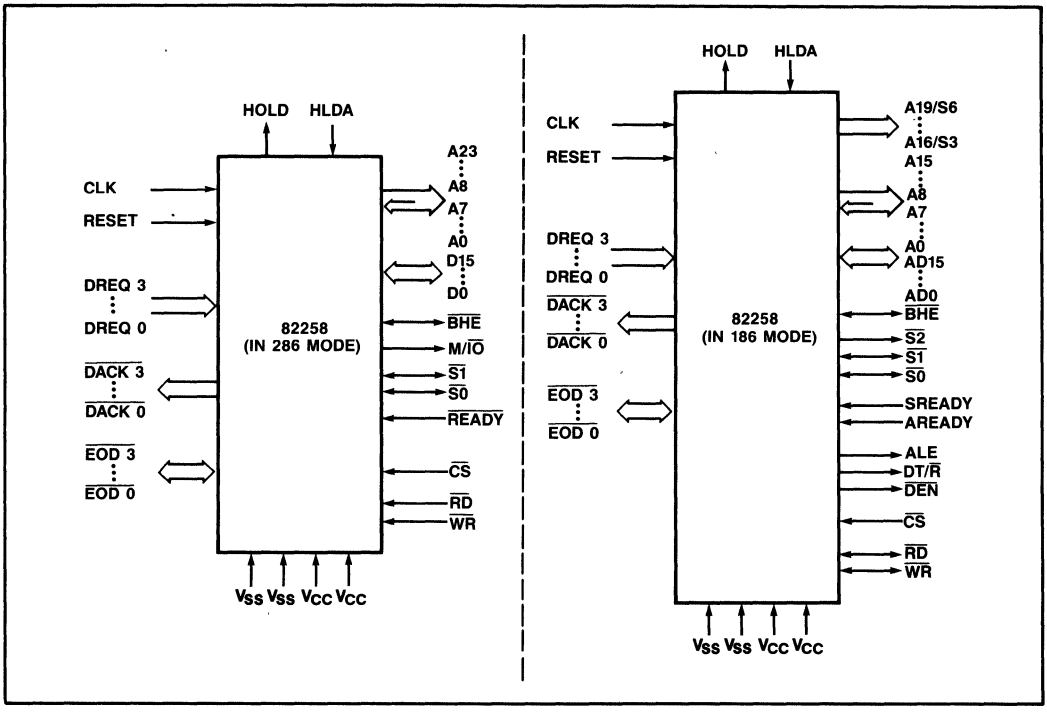


Figure 1. 82258

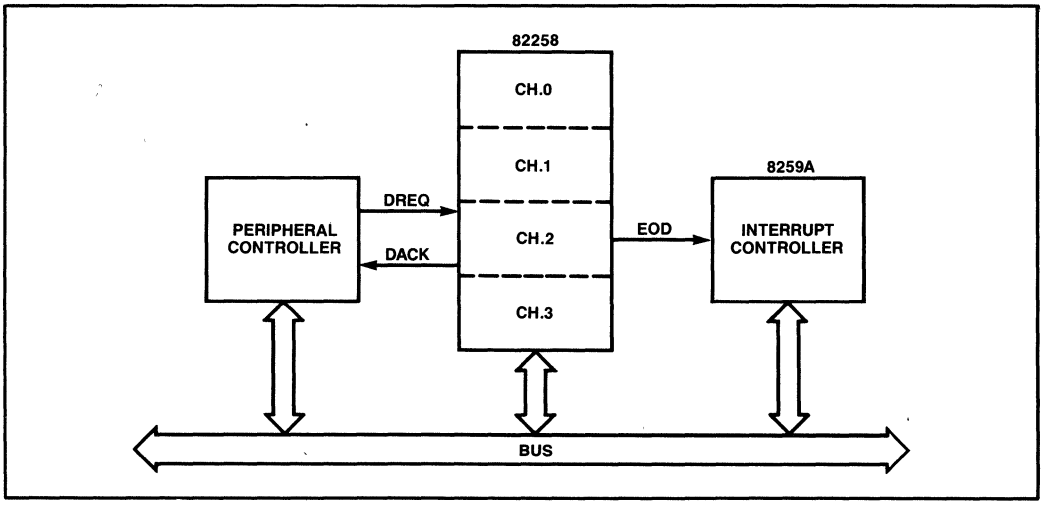


Figure 2. Each Channel with 3 line Interface

MULTIPLEXOR CHANNEL

For supporting a large number of relatively slow equipment (e.g. 9600 baud) like CRT terminals and line printers etc., channel 3 of the 82258 can be programmed to be a multiplexor channel. In this mode channel 3 can service up to 32 subchannel request lines. The multiplexor network is implemented using 8259A interrupt controllers. The multiplexor channel has two modes of operation:

a) Byte or Word Multiplex Operation: In this case the channel is enabled for another device after each byte or word transfer. The maximum cumulative data rate for multiplexor channel in this mode is approximately 275 K byte per second.

b) Block Multiplex Operation: Here the channel is enabled for another device only after transfer of complete data block. The actual block transfer is carried out at a rate of 4M Byte/sec (maximum).

Each individual device can be programmed into its own operating mode so various combinations of byte/word and block multiplex are possible. Each device on the multiplexor channel has its own program. Command chaining is supported on the multiplexor channel. Single cycle transfers are not allowed on the multiplexor channel.

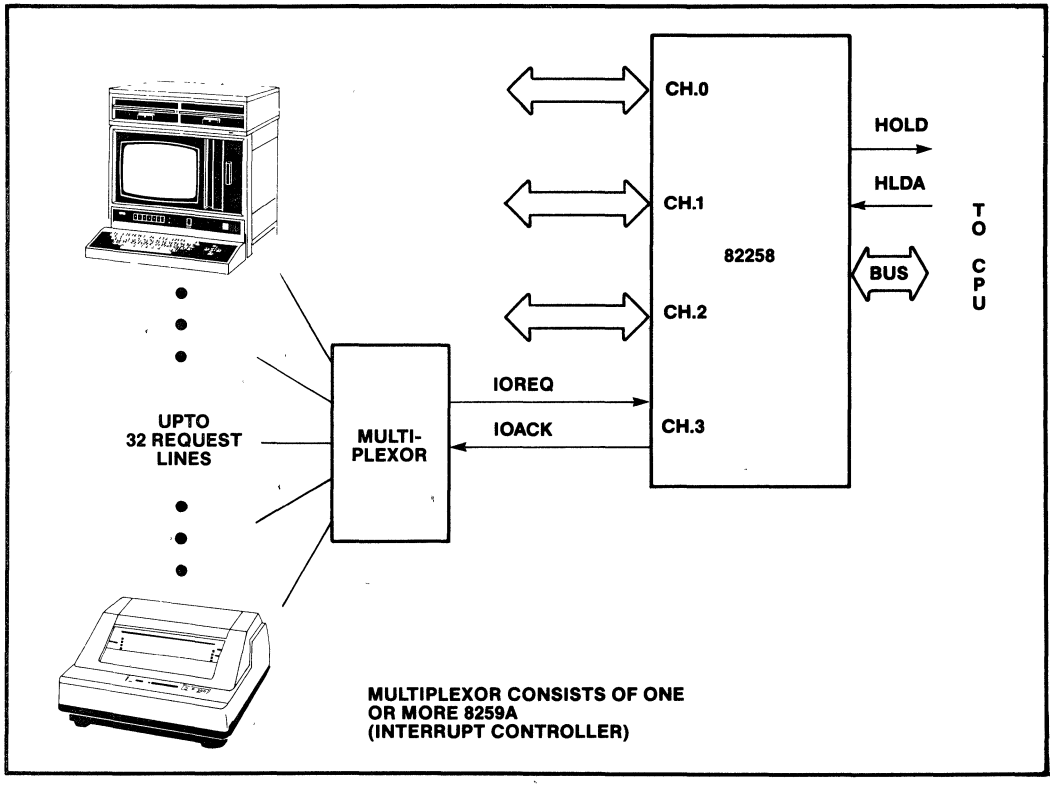


Figure 3. Multiplexor Channel Configuration

TRANSFER IN 1 OR 2 BUS CYCLES

The 82258 is capable of doing single cycle and two cycle data transfers. See Figure 4. In a single cycle transfer, the data is transferred between a peripheral and memory (in either direction) in a single bus cycle. Single cycle transfers provide the maximum DMA throughput. In a two cycle operation, the data is read by the 82258 using a normal bus operation before sending it out to the destination using a normal bus operation.

The 82258 can do "on the fly" operations like translate and compare during a two cycle transfer. The data being transferred can be compared with a given pattern (mask-compare) and the DMA transfer can be optionally stopped on encountering that pattern. The 82258 supports this mask-compare operation on 8- and 16-bit patterns. The data can also be translated on the fly before sending it to the destination. Data can be transferred from one memory region to another in a two cycle transfer mode—this is not possible with single cycle transfer. Another feature of two cycle transfer is automatic assembly/disassembly of data. This means that data can be read as one word (16 bits) and written as two bytes or vice versa. Automatic assembly/disassembly is very often desirable when using 8-bit wide peripherals in a 16-bit system. When writing data to the 8-bit peripheral from memory, data could be fetched as a 16-bit word and written out as two 8-bit bytes. The reverse is true for reading data out of an 8-bit peripheral. This feature saves time and reduces the number of bus cycles to transfer a given block of data.

FAST CHANNEL SWITCHING

For high speed DMA controllers with multiple channels, it is very important to minimize overhead for switching from one channel to another. The 82258 imposes no performance penalty for switching channels. Therefore, the maximum cumulative transfer rate of an 82258 is 8 MBytes/second even if multiple channels are used. The priority of different channels can be programmed to be fixed or rotating. It is also possible to have a combination of the two, i.e., channels 0 and 1, having higher priority than channels 2 and 3, and rotating priority within each pair. The channel with the highest priority gets processed first when multiple channels have to be serviced at the same time.

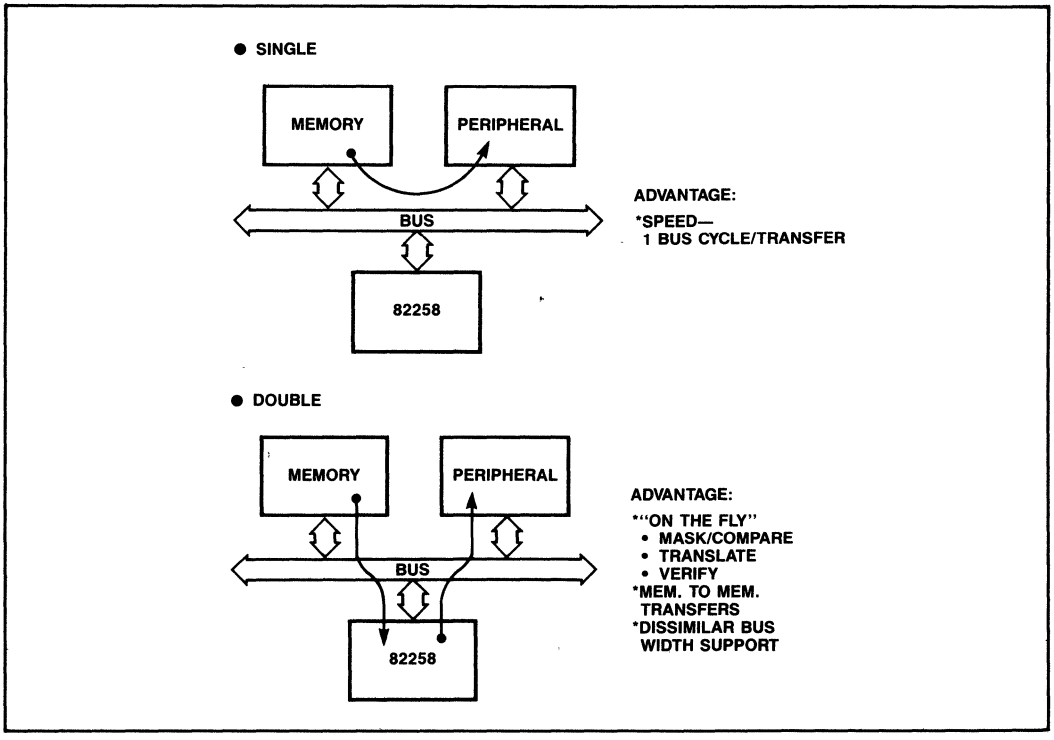


Figure 4. Single/double cycle transfer

16 MBYTE ADDRESSING

Being the DMA controller for the 80286, the 82258 supports an address range of 16 MBytes in memory and I/O space. Thus, the source and destination pointers for a DMA are each twenty-four bits long. The 82258 can transfer data blocks as big as 16 MBytes. That is, the byte count for the transfer is also twenty-four bits long.

Figure 5 shows the user visible registers of the 82258. A set of five registers, called the General Registers, is used for all four channels. There is a set of channel registers for each of the four channels. The mode register is written first after reset and describes the 82258 environment—bus widths, priorities, etc. The General Command Register (GCR) is used to start and stop the DMA transfer on different channels. The General Status Register (GSR) shows the status of all four channels: i.e., if the channel is running, if an interrupt is pending, etc. The General Burst Register (GBR) and the General Delay Register (GDR) are used to specify the bus load which is permissible for the 82258.

MEMORY BASED COMMANDS

The 82258 and the CPU communicate via memory based commands. All relevant data (parameters) for DMA transfer is written by the CPU in a "command block" in memory which is accessible to the CPU and the 82258. See Figure 6. the command pointer on the 82258 is loaded with the based address of the command block by the CPU. After getting a start channel command from the CPU, the 82258 loads the contents of the command block into its registers and starts the DMA. After completing operation on a command block, the 82258 writes back the channel status in the command block in memory. Optionally, source pointer, destination pointer and byte counter register may also be written out to the command block. Although all channel registers are user accessible, they need to be directly accessed only for diagnostic purposes.

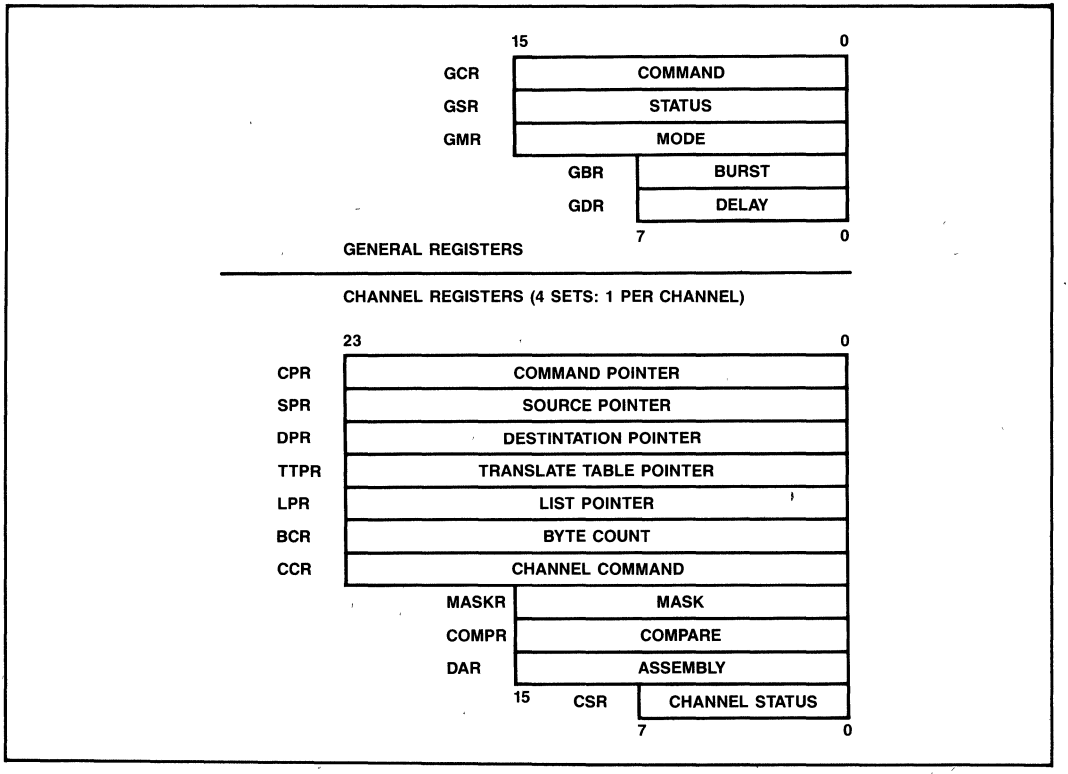


Figure 5. 82258 Register Set

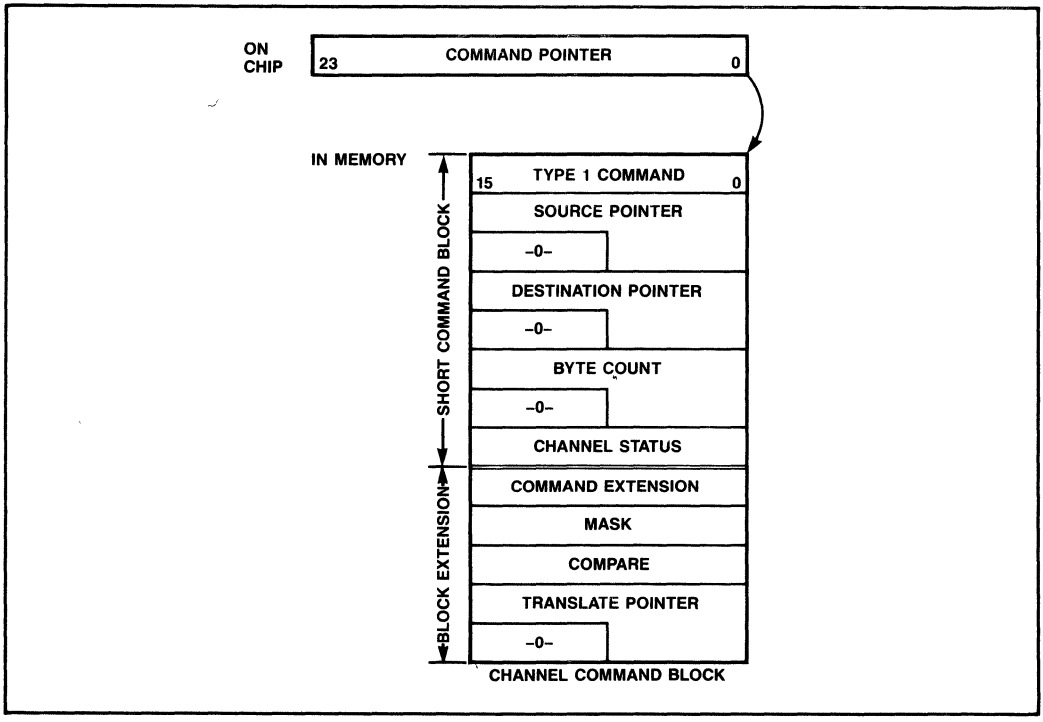


Figure 6. Command Block

COMMAND CHAINING

It is possible to have command blocks in a chain as shown in Figure 7. After completing one block, the 82258 starts processing the next block automatically, This is called command chaining. It is therefore possible to execute a sequence of different types of DMAs without CPU intervention. Besides standard DMA command blocks, there are special command blocks which execute unconditional and conditional branching (depending on the type of DMA termination), and (un) conditional stop. Conditional command chaining offers tremendous flexibility since it is possible to do conditional DMA processing with a degree of flexibility previously found only in microprocessors.

The shortest and simplest DMA implementation consists of a normal command block followed by a "STOP" command block. A simple auto-reload DMA can be implemented with a DMA command block followed by a JUMP to top command block. More complex structures are also easily implemented. See Figure 8.

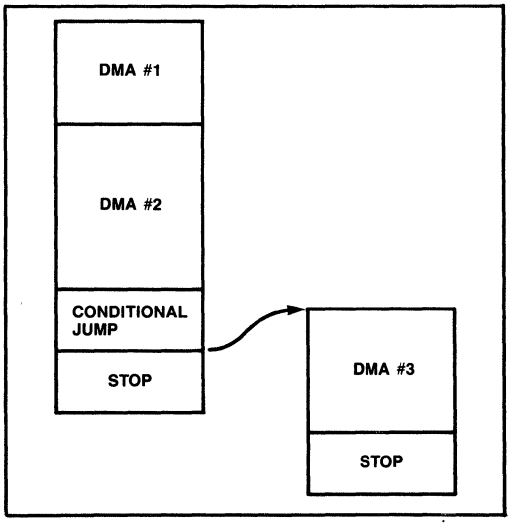


Figure 7. Command Chaining

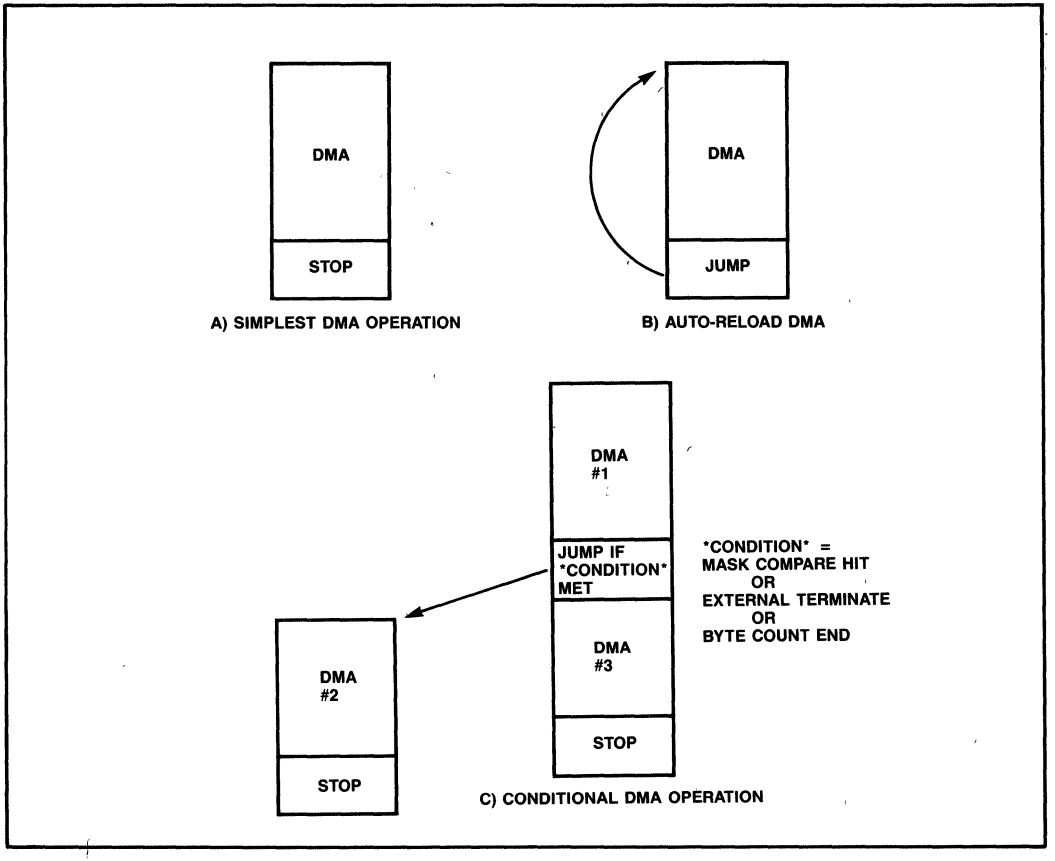


Figure 8. Complexities in Command Chaining

DATA CHAINING

Very often it is necessary to collect data from different data blocks and send it out to a peripheral. This can be done very efficiently by data chaining. Figure 9 shows an example of data chaining. Each block on the right consists of byte count, data pointer and link pointer. As soon as the data from block 1 has been sent out to the destination, the next data block is located through the link pointer and is sent out. This process continues until a zero is encountered in the byte count field. The big advantage here is that the data blocks can be included, removed, or their sequence altered dynamically, allowing the CPU to manipulate the link pointers. Data chaining can be used for serial data communications controllers where different blocks represent different types of information (e.g., header, address, tail, etc.). Such linking of data blocks can also be implemented using command chaining. Using command blocks for data chaining is comparatively slower because the whole command block must be loaded before the DMA can start.

Another form of data chaining called list chaining (as opposed to "linked list" data chainin described above) is still faster. It is illustrated in Figure 10. In this case, there is only one list containing details on all the data blocks to be chained. The latency of going from one data block to the next is of the order of 1 microsecond. List chaining is preferred to linked list chaining when speed of going to the next block is of prime importance. Linked list chaining offers greater flexibility, however.

Data chaining enables "gathering" of data from various data blocks to one destination and "scattering" of data from one source to several data blocks.

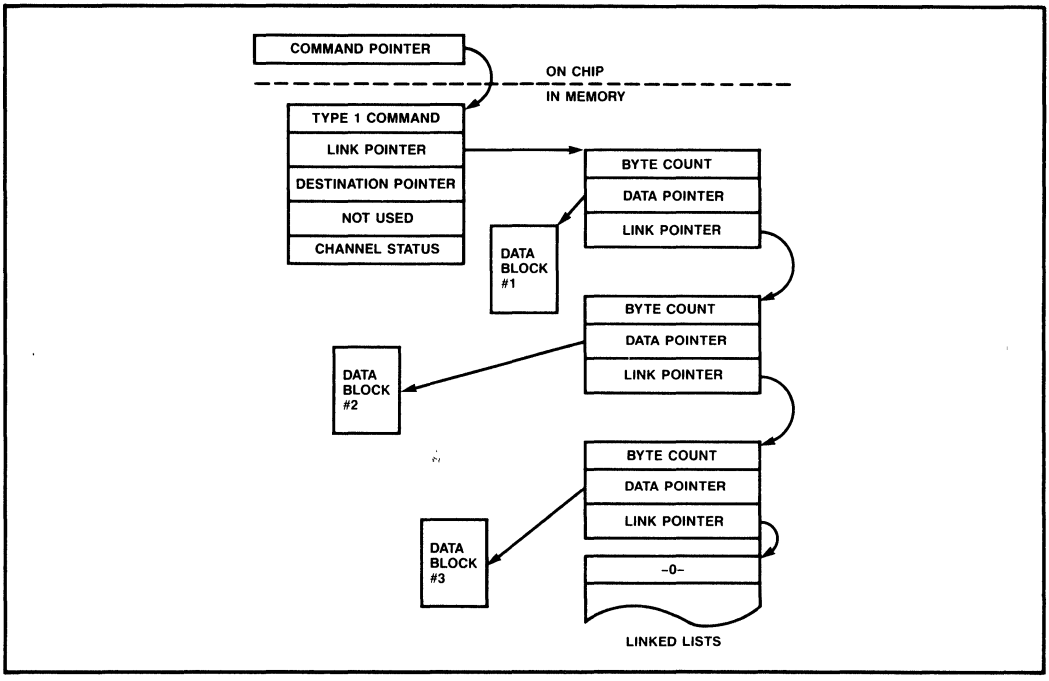


Figure 9. Data Chaining (Linked List)

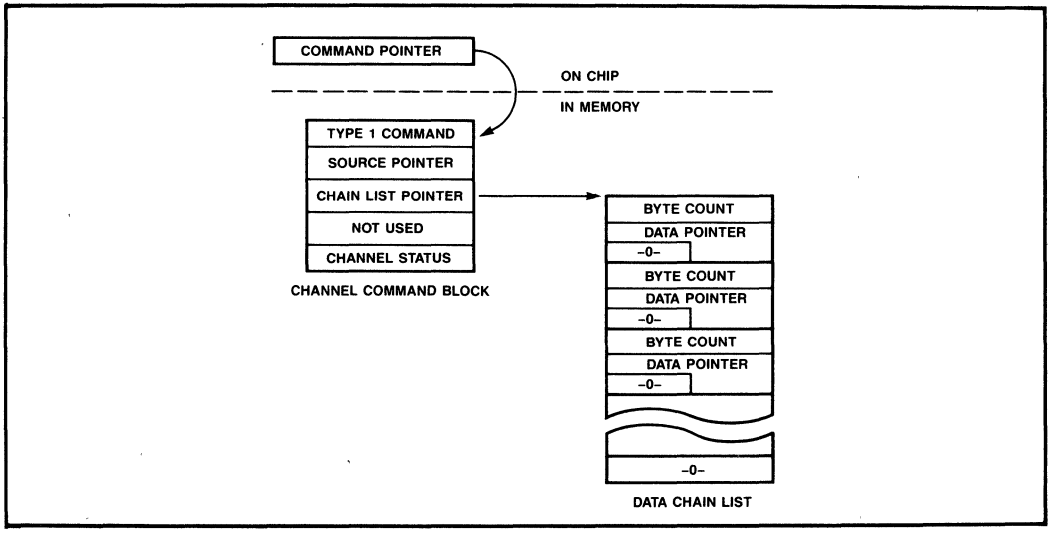


Figure 10. Data Chaining (List Chaining)

VERIFY

Verification of data read from the peripheral is also an often required function. It is like a block-compare operation. See Figure 11. A byte (or word) read from peripheral and a memory block are compared. The transfer can be halted if a mismatch occurs. It is very useful to compare the data on a disk sector with a memory block.

The "verify and save" mode of the 82258 also supports a simultaneous data transfer (in single cycle mode) followed by a verify. Here the byte (or word) being transferred is read by the 82258, saved in memory, and then compared with another byte/word fetched from memory.

CHANNEL COMMANDS

The channel commands are contained in the channel command block (Figure 5). Up to 22 bits are used to specify the command. There are two types of channel commands:

- Type 1: for data movement
- Type 2: for command chaining control

Type 1 command bits contain information on:

a. Source and destination bus width
b. If source and/or destination address should be incremented or decremented or kept constant during the transfer
c. If source/destination is in memory or I/O space
d. If data chaining (list or linked list) is to be performed
e. If the data transfer is synchronized (source or destination)
f. If an "on the fly" match operation and /or translate operation has to be performed
g. If a verify operation has to be performed

For certain type 1 transfers which, for example, do not use "on the fly" match, translate or verify feature, the command is only 16 bits long and only a short command block is necessary (Figure 6).

Type 2 command blocks are 6 bytes long of which the first 2 bytes form the command and the rest is either a relative displacement or an absolute address for the JUMP operation. There are two basic type 2 commands:

1. JUMP—conditional and non-conditional
2. STOP—conditional and non-conditional

The conditional case tests for either of the four condition bits which are altered at the termination of any DMA operation:

- Termination due to byte count being zero
- Termination due to mask-compare
- Termination due to external terminate
- Verify operation resulting in mismatch

It is thus possible to JUMP or STOP further execution of commands based on any of these conditions and optionally generate EOD or an interrupt signal.

A combination of type 1 and 2 commands gives the 82258 a high degree of "programmability". It can thus execute quite complex algorithms with a fairly low demand for CPU service.

82258—80286 SYSTEM

Figure 12 shows the 82258 in an 80286 system. As described earlier, it is tailored to fit not only in an 80286 system but also in 8086, 8088, 80186, and 80188 systems. The 80286 can synchronously access the DMA controller through status lines. The 82258 and 80286 share the same local bus and support components, 82284, 82288, 82289, and 8287 and 8283. The bus arbitration between the 82258 and 80286 is done through HOLD and HOLDA lines. The protocol ensures that only one of them possesses the bus at any time.

REMOTE MODE

To achieve a yet higher throughput it is imperative that both the processor and the DMA controller possess a private bus. This is called the remote mode of operation for the 82258 and is illustrated in Figure 13. When operating in the remote mode, the 82258 can work in parallel with the main processor (which may also possess its own private bus) accessing resources on the resident bus most of the time. Only for communication with the processor does the 82258 have to access the system bus. The registers of the 82258 are always accessible to the processor over the system bus. In addition to achieving a higher throughput due to parallel processing, remote mode is also preferred

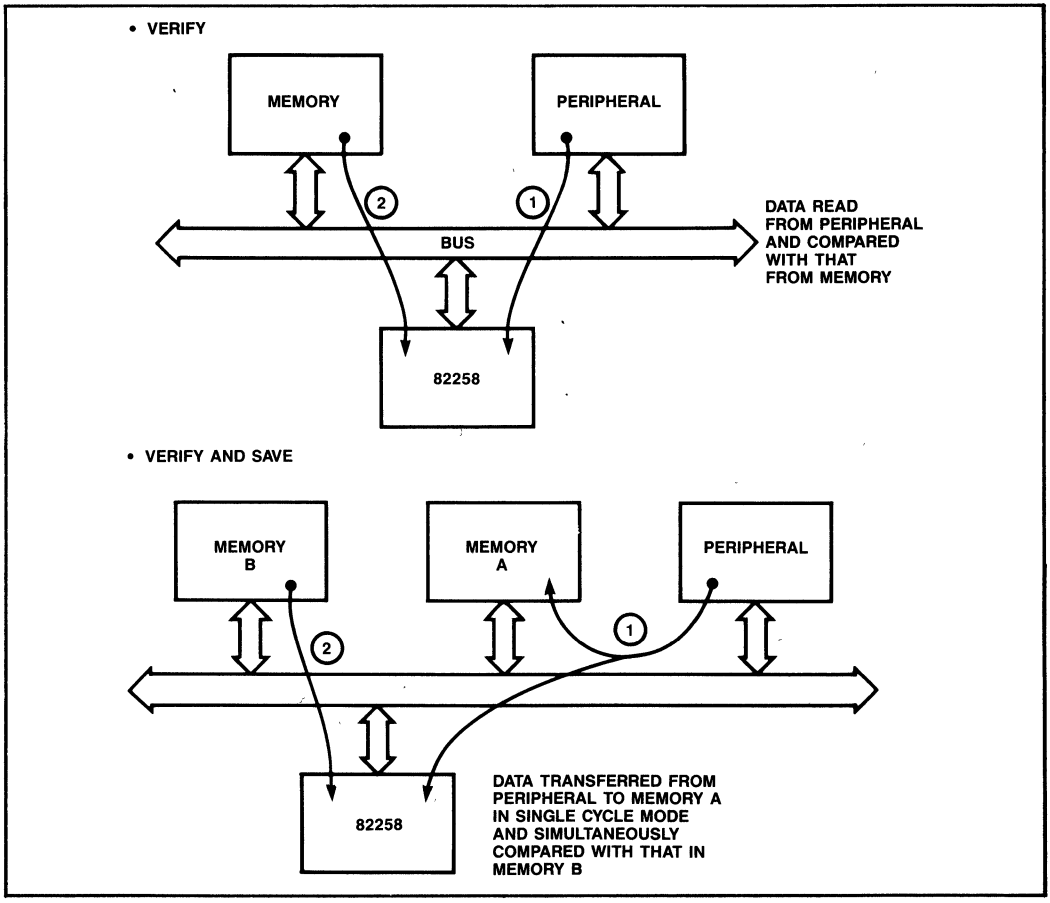


Figure 11. Verify Operation

for good system partitioning. For example, subsystem dealing with winchester and floppy disk could be put separately on a board with the 82258, sufficient resident memory, and the necessary controllers.

PROTECTION ISSUE

The 80286 is a processor with memory management and protection functions on-chip. The 82258 does not directly support the protected virtual addressing mechanism of the 80286. This does not hurt the protection, since the 80286 has to program the ADMA with real (physical) addresses for source, destination and other pointers. These real addresses are, as such, generated through the protection mechanism of the 80286 and are hence checked against protection violation. Normally, an I/O utility routine is provided by the Operating System to service the 82258. No direct user access should be allowed to the 82258 from lower privilege levels. The OS utility converts the virtual addresses to real addresses when programming the 82258.

IMPLEMENTATION

The 82258 will be implemented in Intel's HMOS II process technology and packaged in a 68-pin JEDEC type A leadless chip carrier.

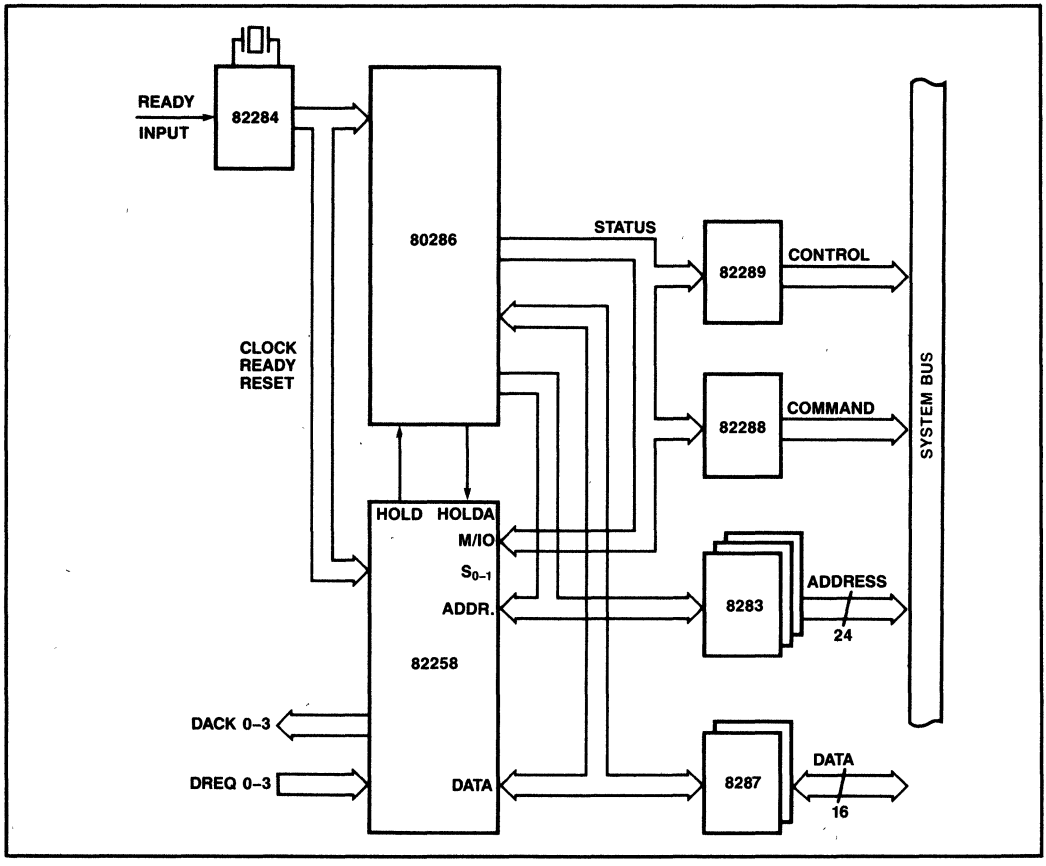


Figure 12. 82258 in 80286 System

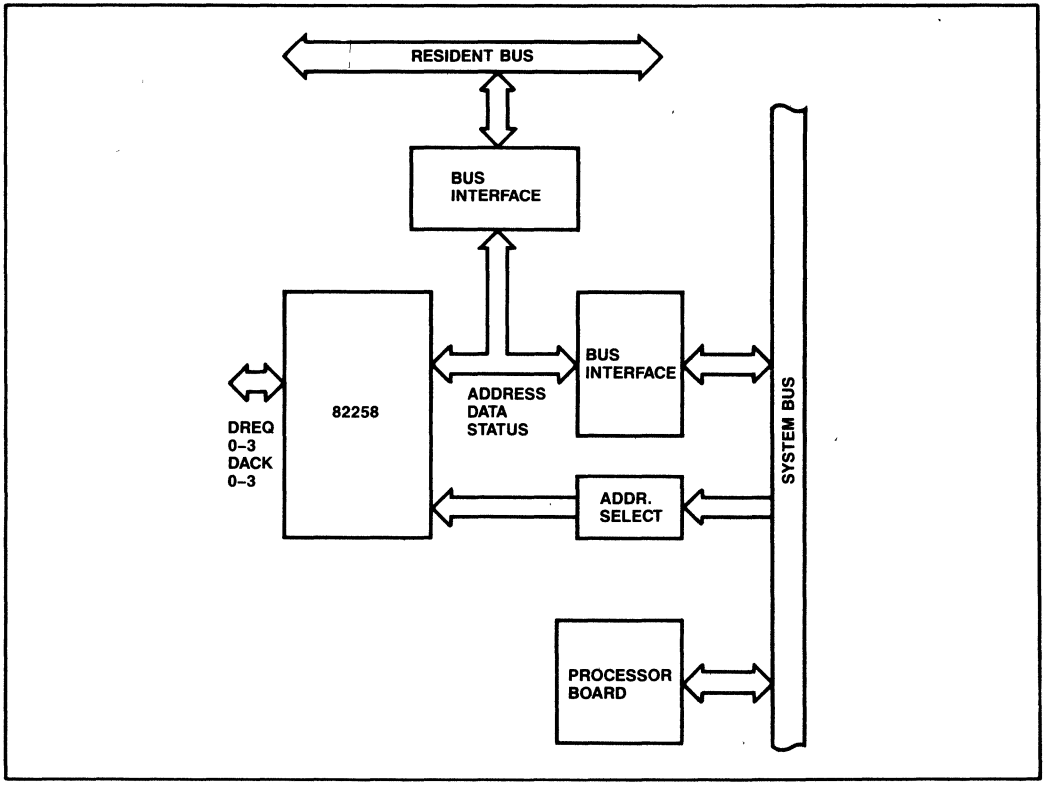


Figure 13. 82258 in Remote or Stand-Alone Configuration

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.

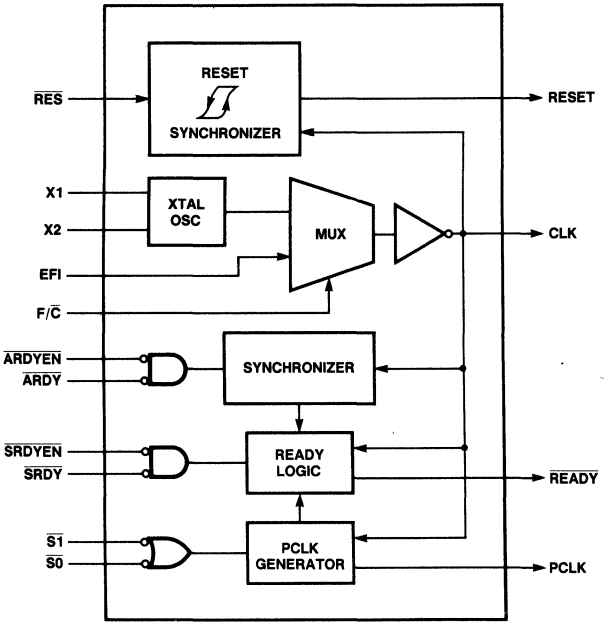


Figure 1. 82284 Block Diagram

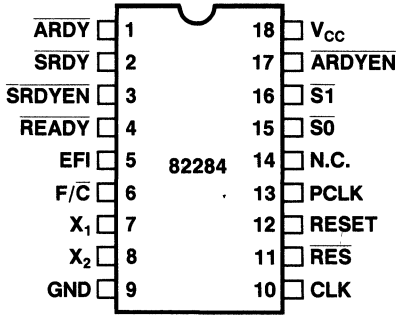


Figure 2. 82284 Pin Configuration

* Multibus is a patented bus of Intel

January 1985
ORDER NUMBER: 210453-003

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

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.

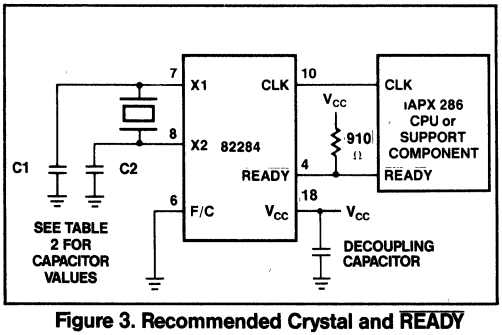


Figure 3. Recommended Crystal and $\overline{READY}$ Connections

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.

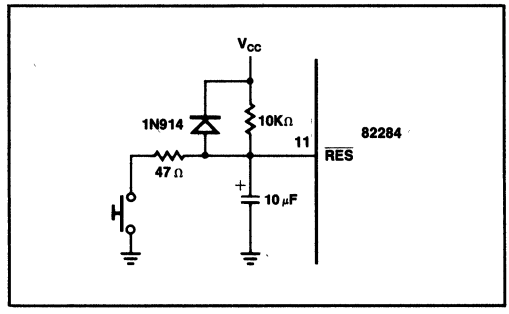


Figure 4. Typical RC $\overline{RES}$ Timing Circuit

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.

$\overline{\text{READY}}$ 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.

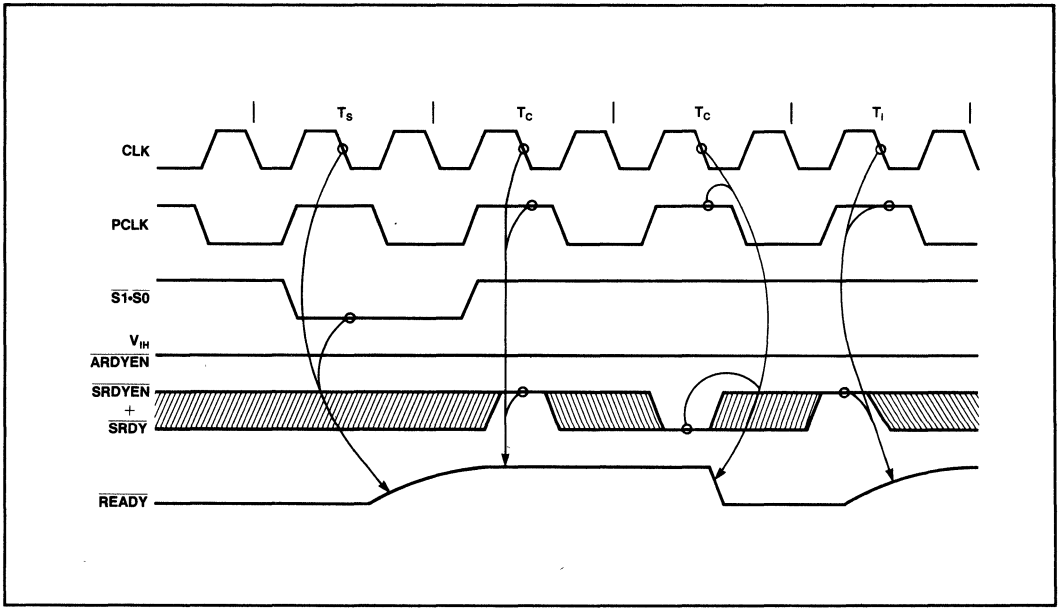


Figure 5. Synchronous Ready Operation

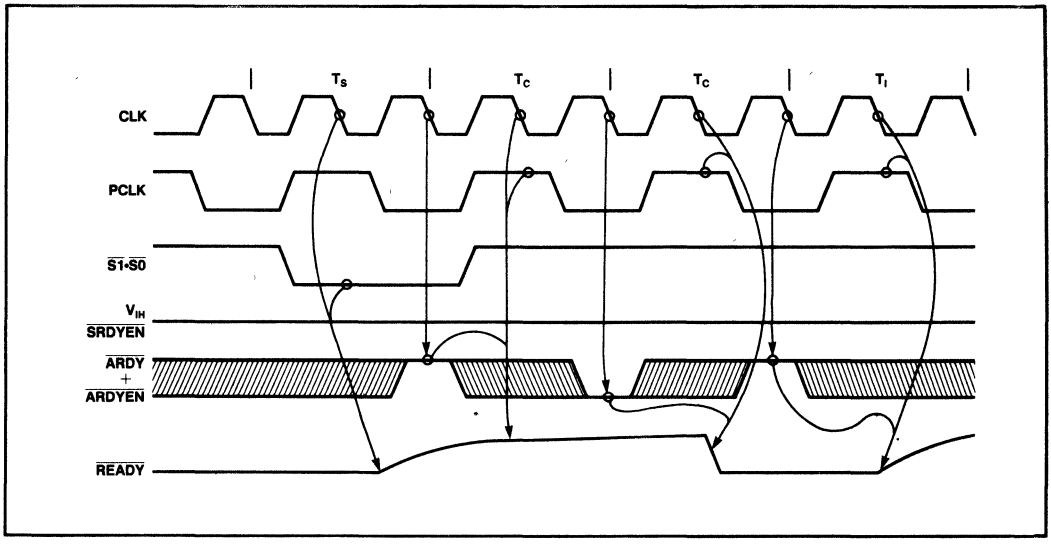


Figure 6. Asynchronous Ready Operation

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, ± 5%)

		6 MHz		8 MHz			
Sym	Parameter	-6 Min	-6 Max	-8 Min	-8 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, ± 5%)

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

		6MHz		8MHz			
Sym	Parameter	-6 Min	-6 Max	-8 Min	-8 Max	Unit	Test Condition
1	EFI to CLK Delay		35		30	ns	at 1.5V Note 1
2	EFI LOW Time	40		25		ns	at 1 5V Note 1 Note 7
3	EFI HIGH Time	35		25		ns	at 1.5V Note 1 Note 7
4	CLK Period	83	500	62	500	ns	
5	CLK LOW Time	20		15		ns	at 1.0V Note 1 Note 2,8
6	CLK HIGH Time	25		25		ns	at 3.6V Note 1 Note 2,8
7	CLK Rise Time		10		10	ns	1.0V to 3.6V Note 1
8	CLK Fall Time		10		10	ns	3.6V to 1.0V Note 1
9	Status Setup Time	28		22		ns	Note 1
10	Status Hold Time	1		1		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		30		ns	Note 1 Note 3
15	$\overline{RES}$ Setup Time	25		20		ns	Note 1 Note 3
16	$\overline{RES}$ Hold Time	10		10		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	45	ns	Note 5
20	RESET Delay	5	50	5	34	ns	Note 5
21	PCLK LOW Time	t4-20		t4-20		ns	Note 5 Note 6
22	PCLK HIGH Time	t4-20		t4-20		ns	Note 5 Note 6

NOTE 1: CLK loading. C_L = 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. PCLK also has 750 ohm pullup.

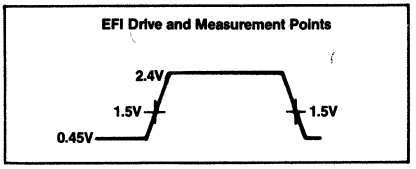
NOTE 6: t4 refers to any allowable CLK period.

NOTE 7: When driving the 82284 with EFI, provide minimum EFI HIGH and LOW times as follows:

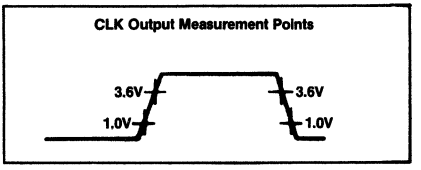
CLK Output Frequency:	8MHz CLK	12MHz CLK	16MHz CLK*
Min. required EFI HIGH time	52ns	35ns	25ns
Min. required EFI LOW time	52ns	40ns	25ns

* At CLK frequencies above 12MHz, CLK output HIGH and LOW times are guaranteed only when using crystal with recommended capacitive loading per Table 2, *not* when driving component from EFI All features of the 82284 remain functional whether EFI or crystal is used to drive the 82284.

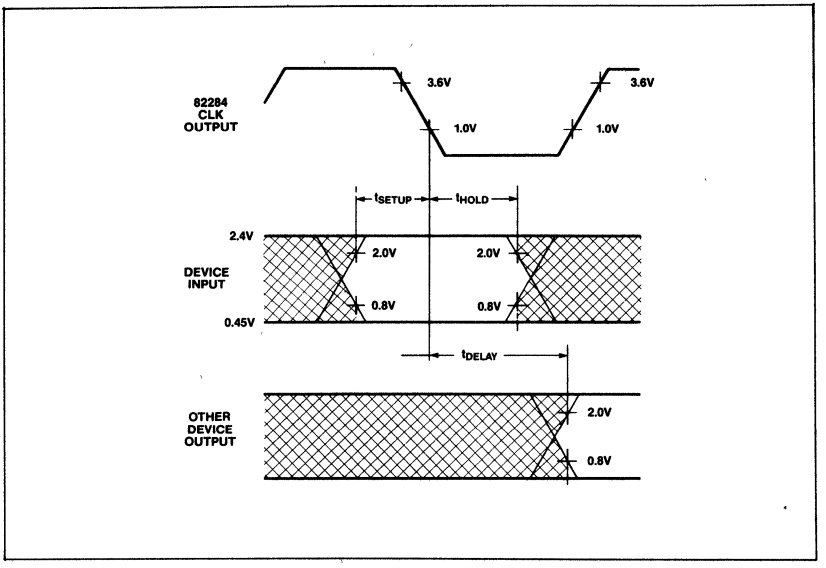
NOTE 8: When using crystal (with recommended capacitive loading per Table 2) appropriate for speed of 80286, CLK output HIGH and LOW times guaranteed to meet 80286 requirements.



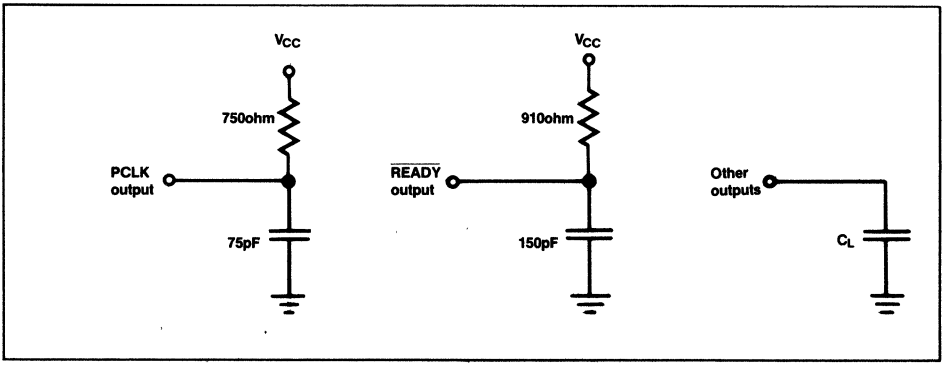
NOTE 9:



NOTE 10:



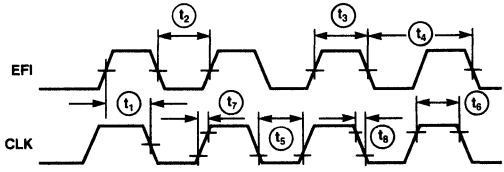
NOTE 11. AC Setup, Hold and Delay Time Measurement — General



NOTE 12. AC Test Loading on Outputs

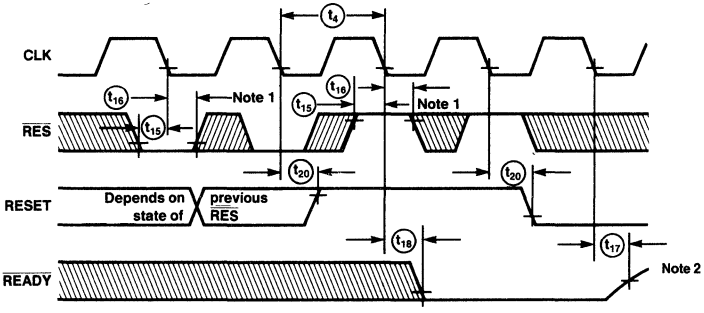
Waveforms

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.

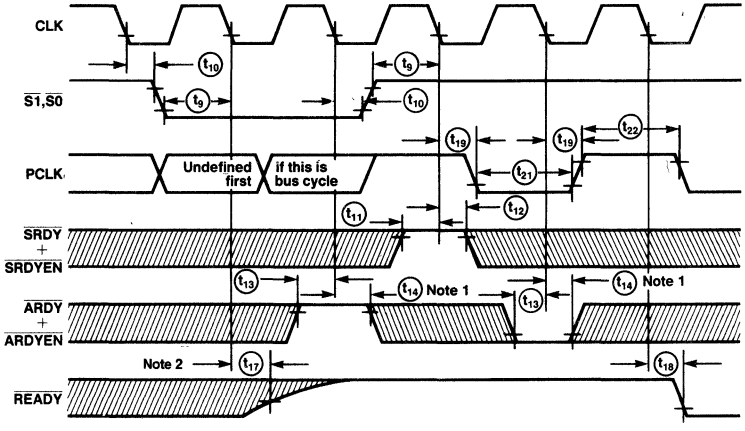
RESET and $\overline{\text{READY}}$ Timing as a Function of $\overline{\text{RES}}$ with $\overline{\text{S1}}$ and $\overline{\text{S0}}$ HIGH



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

(82288-8, 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.

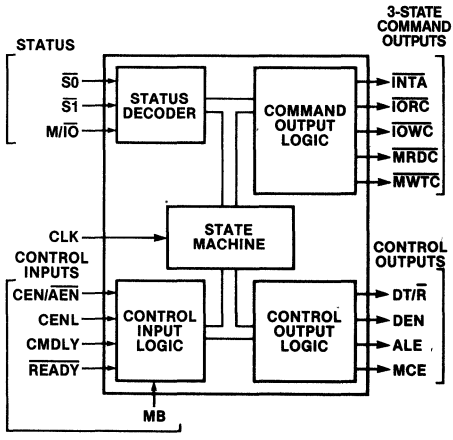


Figure 1. 82288 Block Diagram

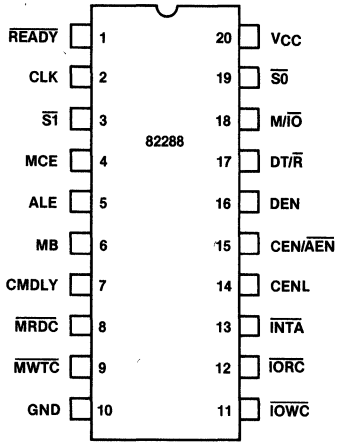


Figure 2. 88228 Pin Configuration

*Multibus is a patented bus of Intel.

January 1985
ORDER NUMBER: 210471-004

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. 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 typically a strapping option and not dynamically changed.
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.

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 $\overline{S0}$.

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

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.

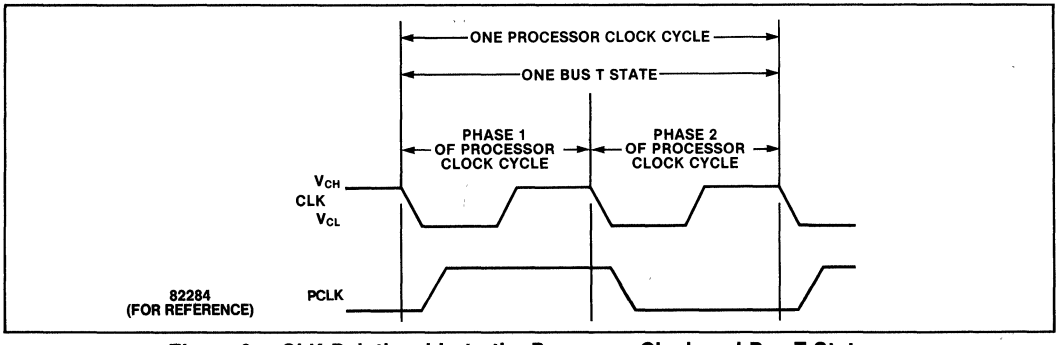


Figure 3. CLK Relationship to the Processor Clock and Bus T-States

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.

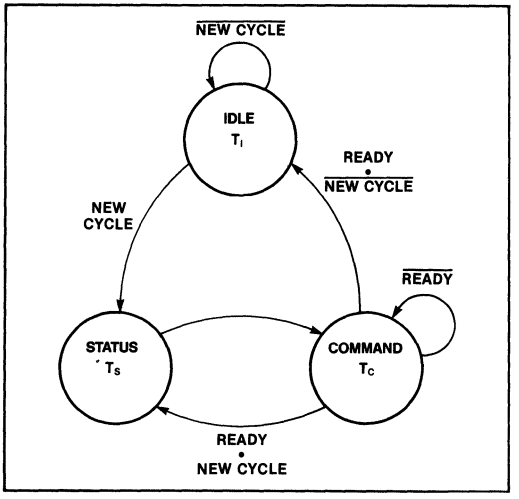


Figure 4. 82288 Bus States

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.

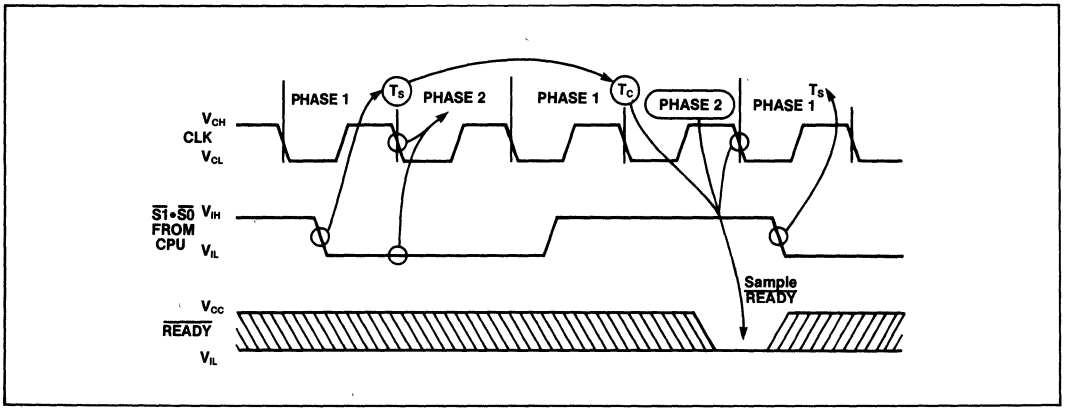


Figure 5. Bus Cycle Definition

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.

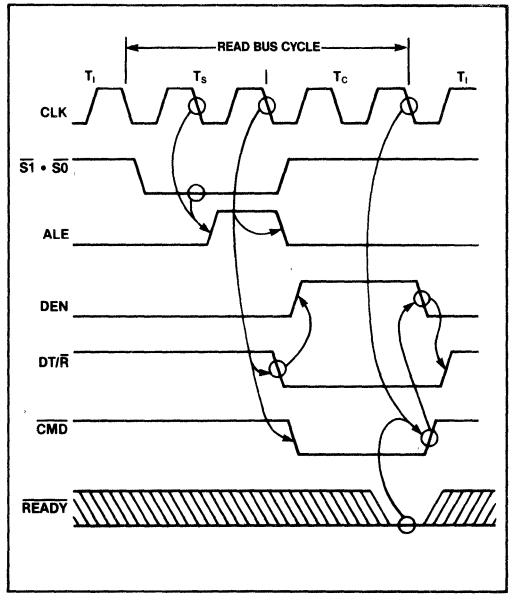


Figure 6. Idle-Read-Idle Bus Cycles with MB = 0

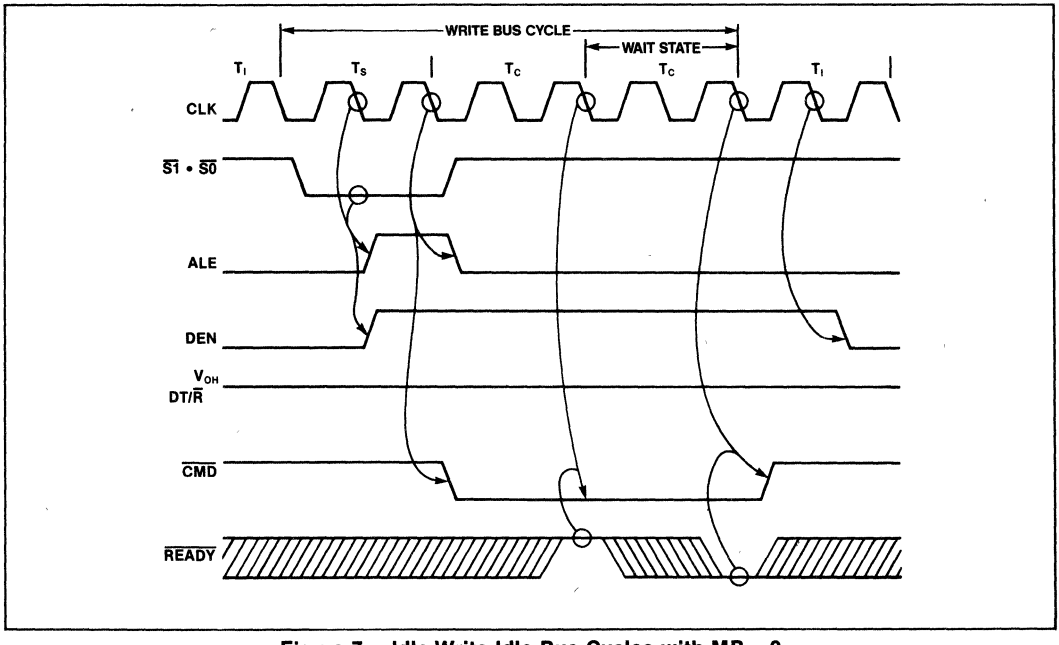


Figure 7. Idle-Write-Idle Bus Cycles with MB = 0

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.

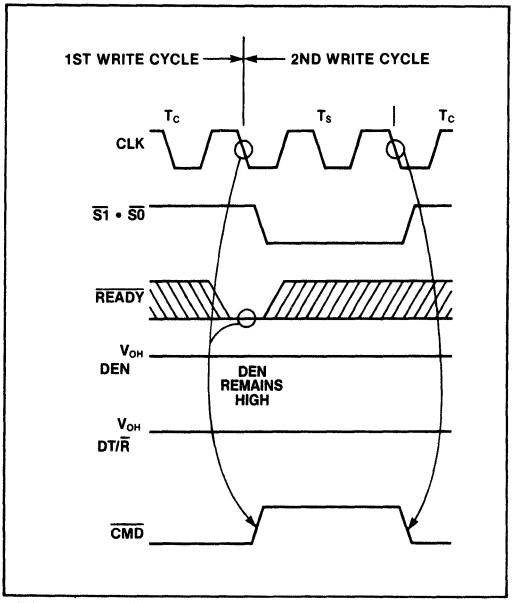


Figure 8. Write-Write Bus Cycles with MB = 0

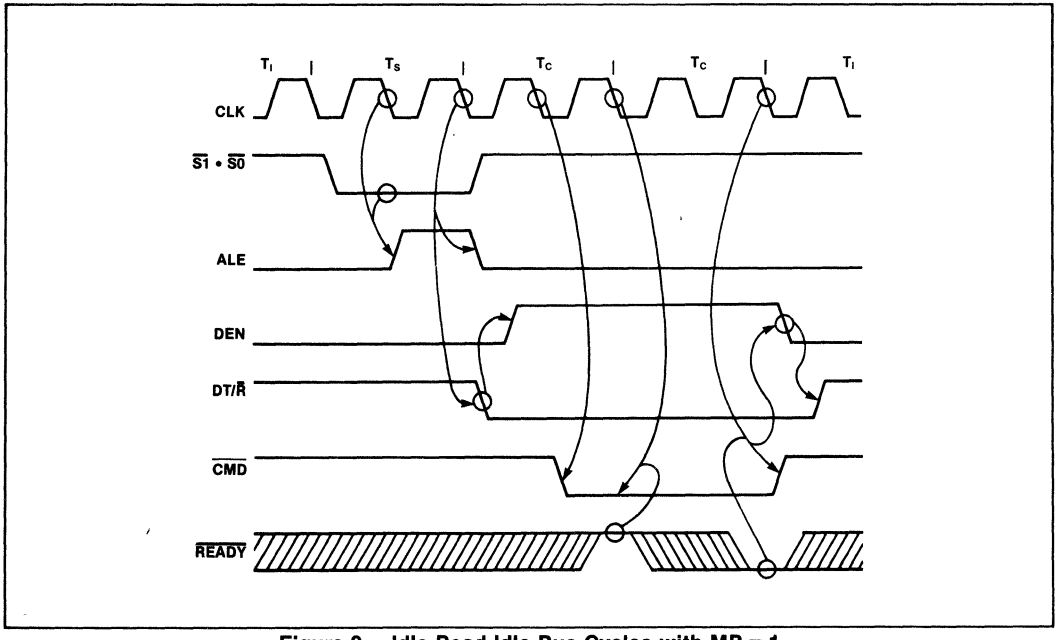


Figure 9. Idle-Read-Idle Bus Cycles with MB = 1

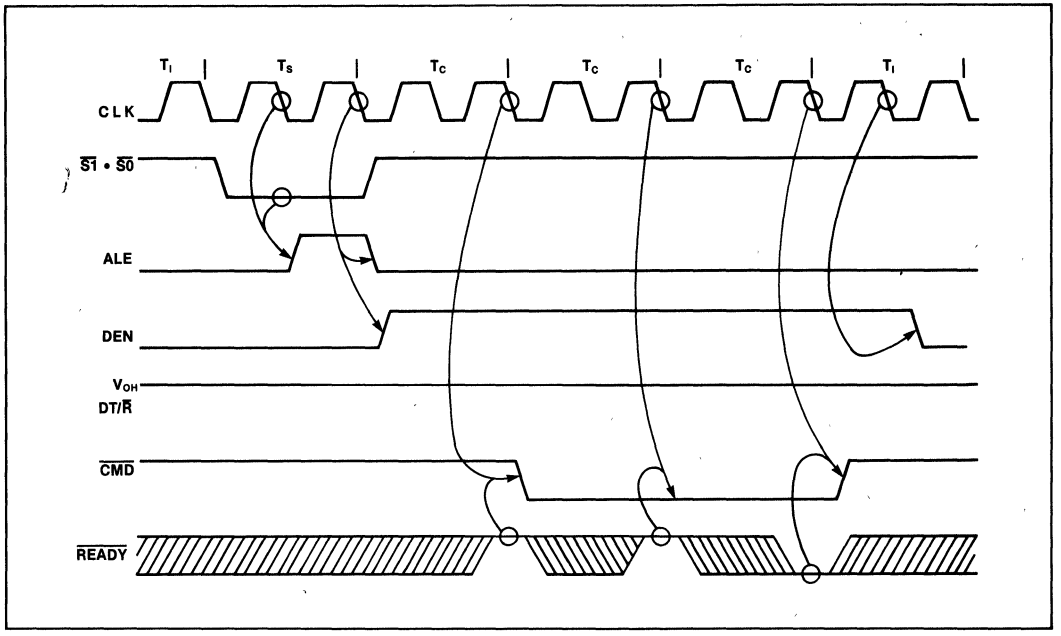


Figure 10. Idle-Write-Idle Bus Cycles with MB = 1

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.

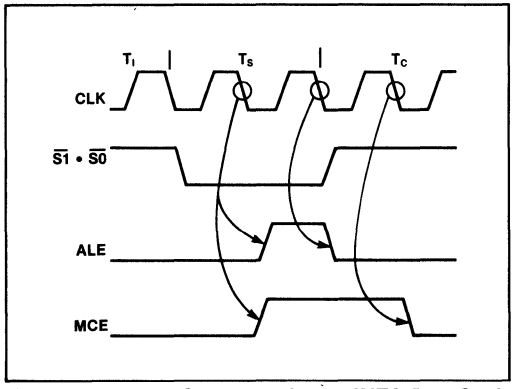


Figure 11. MCE Operation for an INTA Bus Cycle

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). $\overline{\text{READY}}$ 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/$\overline{\text{R}}$.

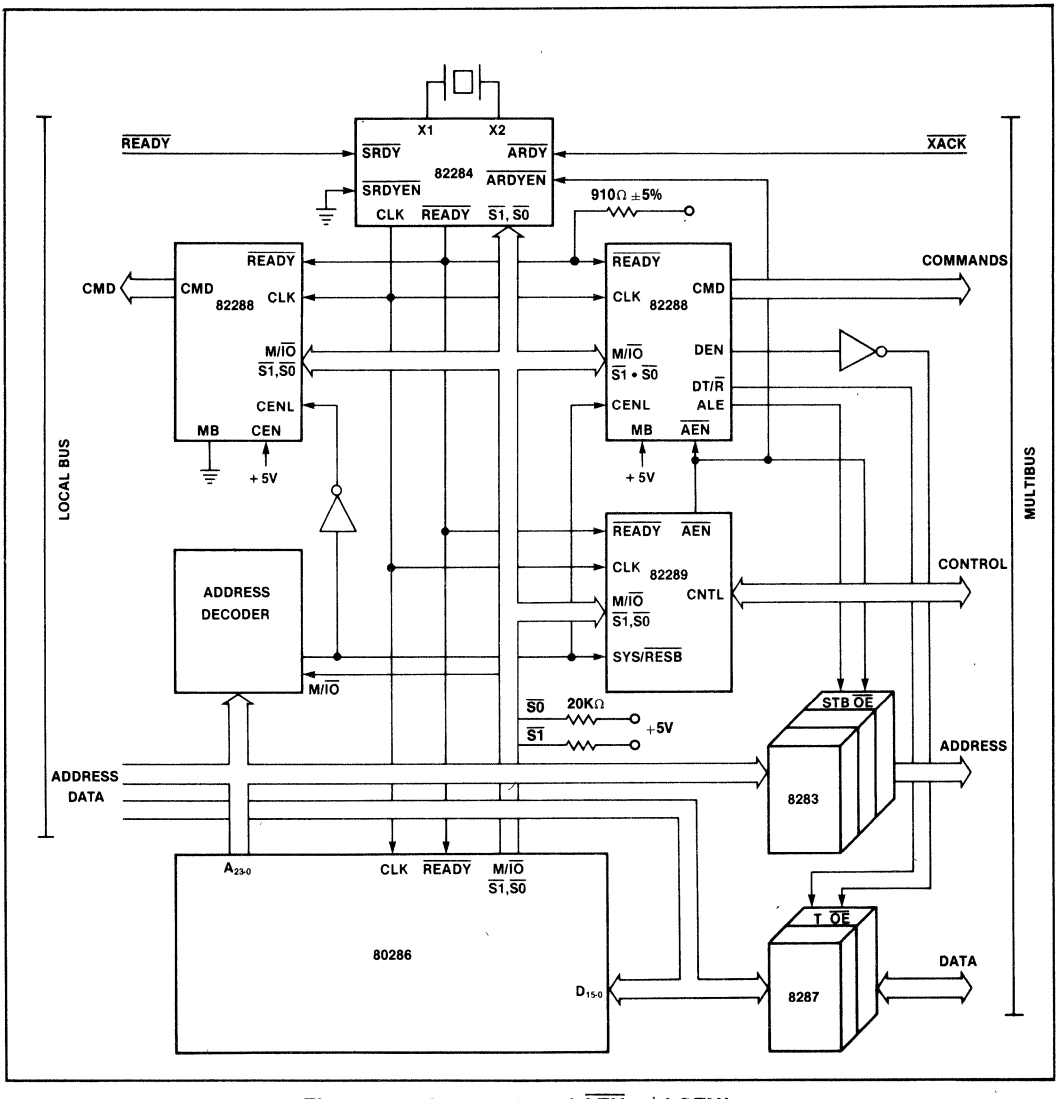


Figure 12. System Use of $\overline{\text{AEN}}$ and CENL

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.

$\overline{\text{READY}}$ 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/$\overline{\text{R}}$ 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, ± 5%)

		6 MHz		8 MHz			
Symbol	**Parameter**	**-6 Min.**	**-6 Max.**	**-8 Min.**	**-8 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		120		120	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

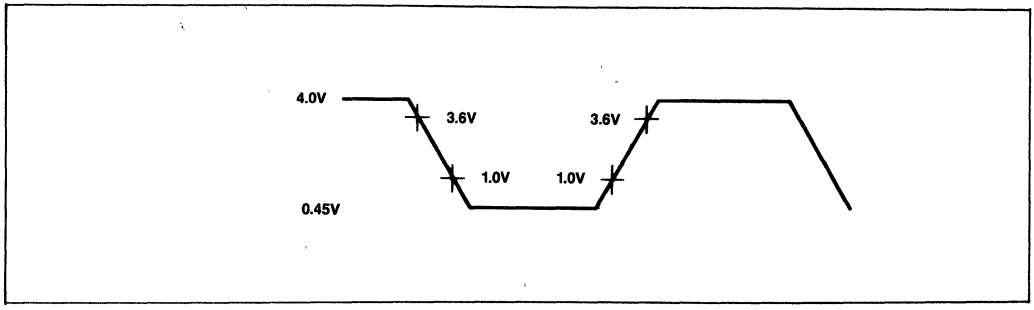
NOTE: 1. Command Outputs are $\overline{INTA}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{MRDC}$, $\overline{MWRC}$.
2. Control Outputs are DT/$\overline{R}$, DEN, ALE and MCE.

A.C. CHARACTERISTICS

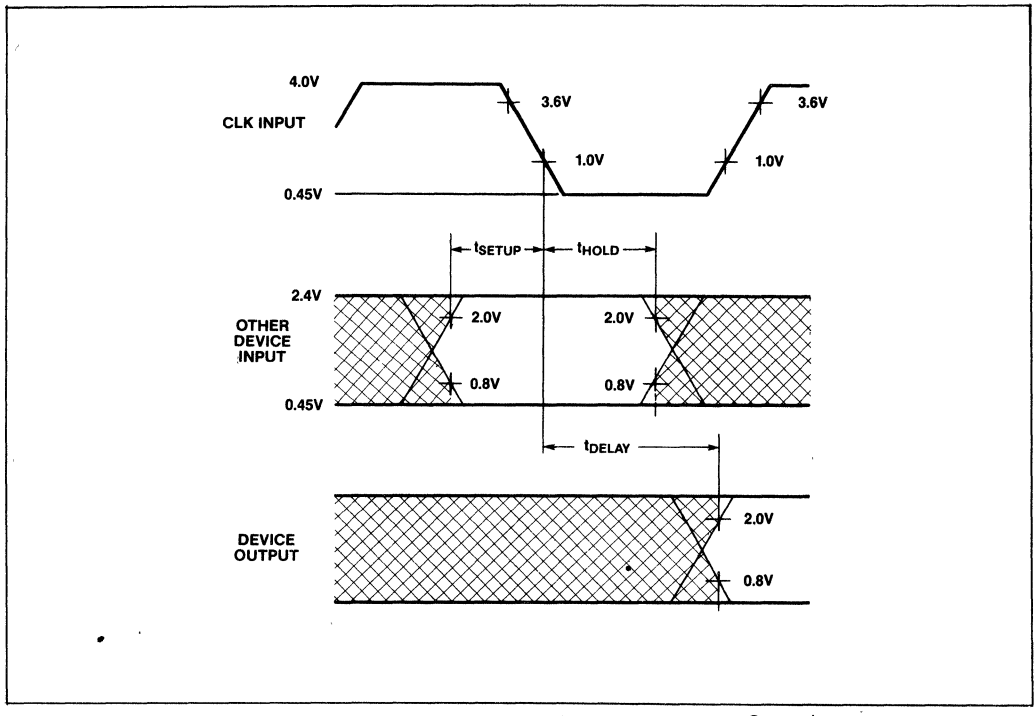
(T_A = 0°C to 70°C, V_{CC} = 5V, ± 5%)
AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in data sheet waveforms, unless otherwise noted

		6 MHz		8 MHz			
Sym	Parameter	-6 Min.	-6 Max.	-8 Min.	-8 Max.	Unit	Test Condition
1	CLK Period	83	250	62	250	ns	
2	CLK HIGH Time	25	230	20	235	ns	at 3 6V
3	CLK LOW Time	20	225	15	230	ns	at 1 0V
4	CLK Rise Time		10		10	ns	1 0V to 3 6V
5	CLK Fall Time		10		10	ns	3 6V to 1 0V
6	M/$\overline{IO}$ and Status Setup Time	28		22		ns	
7	M/$\overline{IO}$ and Status Hold Time	1		1		ns	
8	CENL Setup Time	30		20		ns	
9	CENL Hold Time	1		1		ns	
10	$\overline{READY}$ Setup Time	50		38		ns	
11	$\overline{READY}$ Hold Time	35		25		ns	
12	CMDLY Setup Time	25		20		ns	
13	CMDLY Hold Time	1		1		ns	
14	$\overline{AEN}$ Setup Time	25		20		ns	Note 3
15	$\overline{AEN}$ Hold Time	0		0		ns	Note 3
16	ALE, MCE Active Delay from CLK	3	25	3	20	ns	Note 4
17	ALE, MCE Inactive Delay from CLK		35		25	ns	Note 4
18	DEN (Write) Inactive from CENL		35		35	ns	Note 4
19	DT/$\overline{R}$ LOW from CLK		40		25	ns	Note 4
20	DEN (Read) Active from DT/$\overline{R}$	5	50	5	35	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	5	35	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		30	ns	Note 4
26	DEN Active from CEN		35		30	ns	Note 4
27	DT/$\overline{R}$ HIGH from CLK (when CEN = LOW)		50		35	ns	Note 4
28	DEN Active from $\overline{AEN}$		35		30	ns	Note 4
29	$\overline{CMD}$ Active Delay from CLK	3	40	3	25	ns	Note 5
30	$\overline{CMD}$ Inactive Delay from CLK	3	30	3	25	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
35	MB Setup Time	25		20		ns	
36	MB Hold Time	0		0		ns	
37	Command Inactive Enable from MB↓		40		40	ns	Note 5
38	Command Float Time from MB↑		40		40	ns	Note 6
39	DEN Inactive from MB↑		40		30	ns	Note 4
40	DEN Active from MB↓		35		30	ns	Note 4

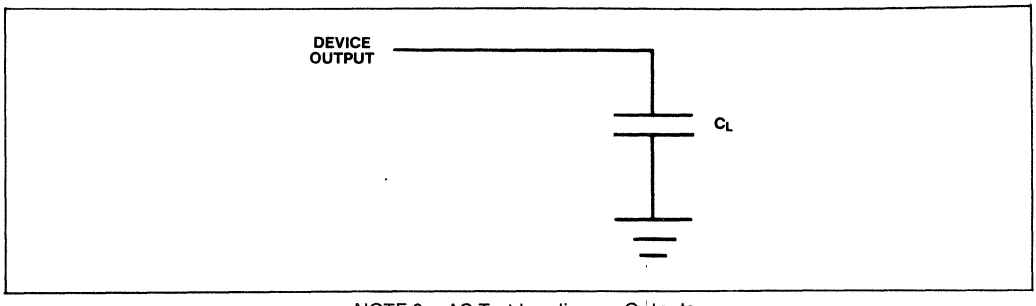
NOTE: 3 $\overline{AEN}$ is an asynchronous input This specification is for testing purposes only, to assure recognition at a specific CLK edge
4 Control output load CI = 150pF
5 Command output load CI = 300pF
6 Float condition occurs when output current is less then I_{LO} in magnitude



NOTE 7: AC Drive and Measurement Points — CLK Input



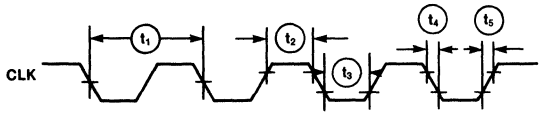
NOTE 8: AC Setup, Hold and Delay Time Measurement — General



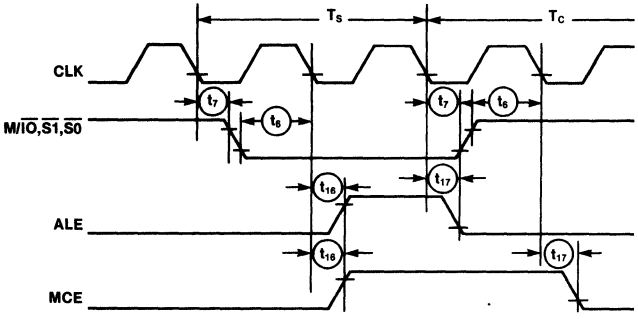
NOTE 9: AC Test Loading on Outputs

WAVEFORMS

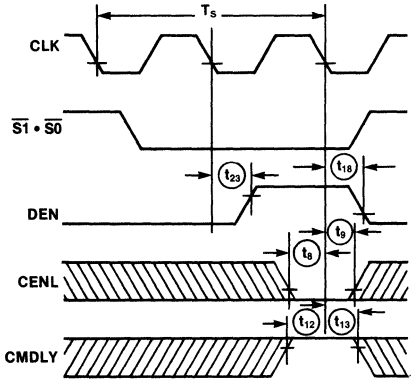
CLK CHARACTERISTICS



STATUS, ALE, MCE, CHARACTERISTICS

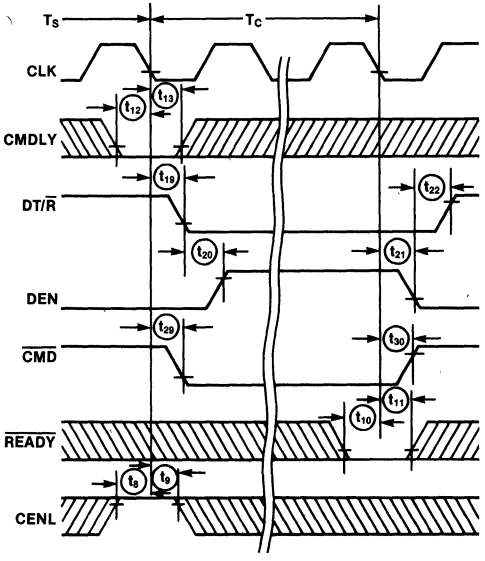


CENL, CMDLY, DEN CHARACTERISTICS WITH MB = 0 AND CEN = 1 DURING WRITE CYCLE

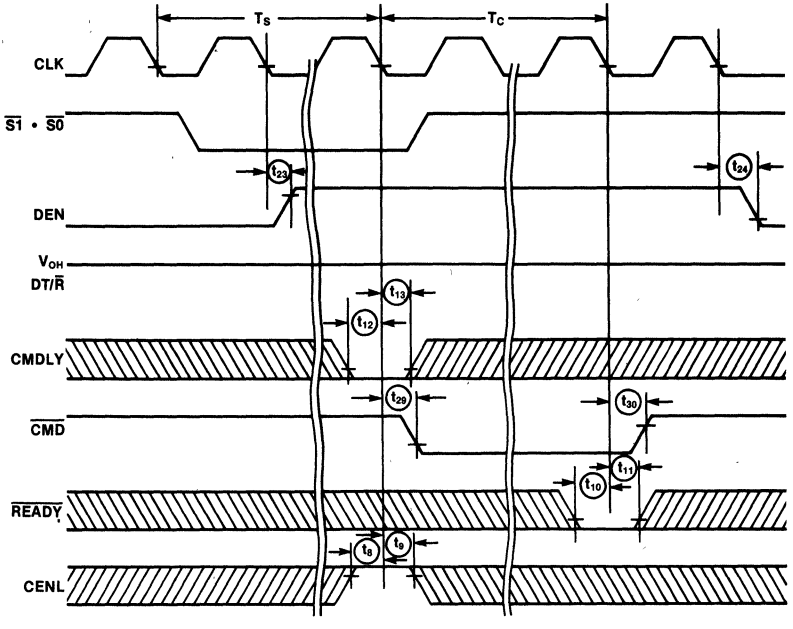


WAVEFORMS (Continued)

READ CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1

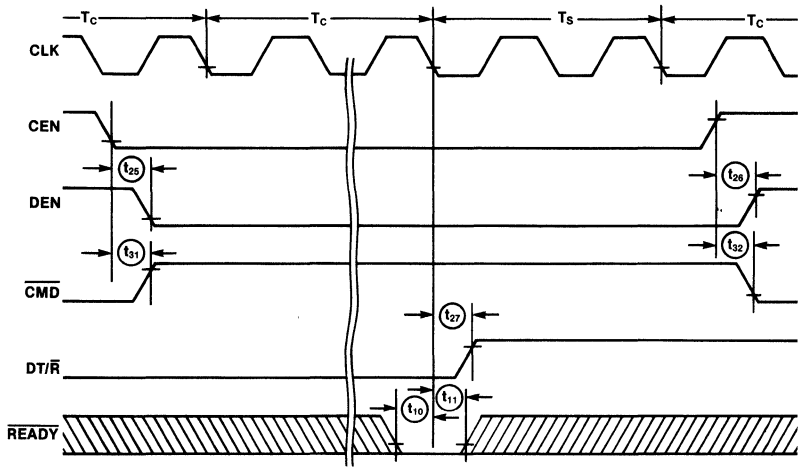


WRITE CYCLE CHARACTERISTICS WITH MB = 0 AND CEN = 1

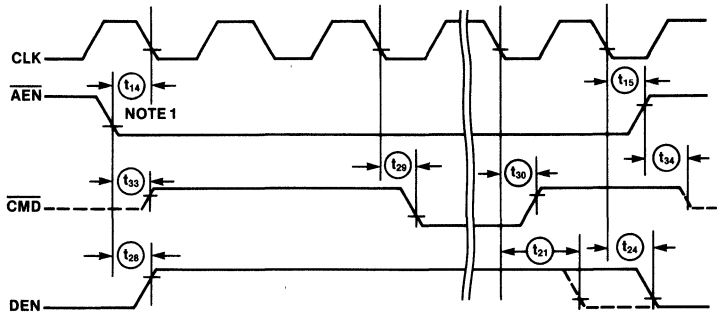


WAVEFORMS (Continued)

CEN CHARACTERISTICS WITH MB = 0



AEN CHARACTERISTICS WITH MB = 1



NOTE 1: $\overline{AEN}$ is an asynchronous input. $\overline{AEN}$ setup and hold time is specified to guarantee the response shown in the waveforms.

82289
BUS ARBITER
FOR iAPX 286 PROCESSOR FAMILY

- **Supports Multi-master System Bus Arbitration Protocol**
- **Synchronizes 80286 Processor with Multi-master Bus**
- **Compatible With Intel Bus Standard Multibus®* (IEEE 796 Standard)**
- **Three Modes of Bus Release Operation for Flexible System Configuration**
- **Supports Parallel, Serial, and Rotating Priority Resolving Schemes**
- **Available in EXPRESS - Standard Temperature Range**

The Intel 82289 Bus Arbiter is a 5-Volt, 20-pin HMOS III component for use in multiple bus master iAPX 286 systems. The 82289 provides a compact solution to system bus arbitration for the 80286 CPU.

The complete IEEE 796 Standard bus arbitration protocol is supported. Three modes of bus release operation support a number of bus usage models.

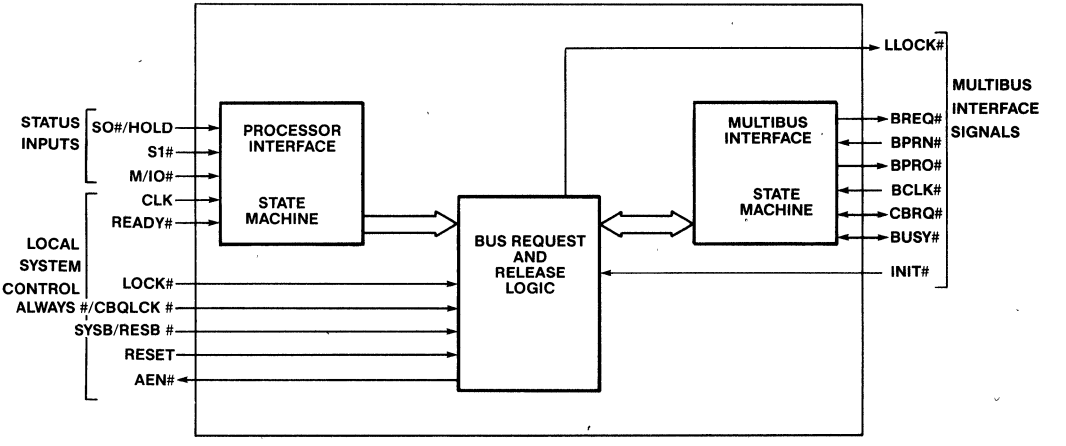
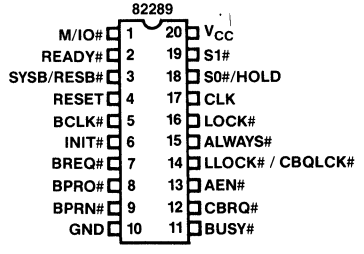


Figure 1. 82289 Block Diagram



# INDICATES FUNCTION IS ACTIVE LOW

Figure 2. 82289 Pin Diagram

Table 1. 82289 Pin Definition

Symbol	Pin(s)	Type	Name and Function
CLK	17	I	**SYSTEM CLOCK** accepts the CLK signal from the 82284 Clock Generator chip as the timing reference for the bus arbiter and processor interface signals.
S0#/HOLD	18	I	**STATUS INPUT S0# or HOLD** is either the S0# status signal from 80286 or the HOLD signal from some other bus master. The function of this input is established during the processor reset of the 82289 Bus Arbiter. The 80286 S0# pin meets the setup and hold time requirements of this pin. The S0# pin function is selected by forcing this input high during the falling edge of processor reset. If the 82289 is used to support an 80286 processor, the S0# output of the processor will be high during reset. In supporting the 80286 processor, the 82289 decodes the S0# pin together with the other status input pins, S1# and M/IO#, to determine the beginning of a processor bus cycle and initiate bus request and surrender actions. The HOLD function of the S0#/HOLD pin is selected by holding this input low during the falling edge of processor reset. When supporting a bus master other than 80286, the 82289 monitors the HOLD signal to initiate bus request and surrender actions.
S1#, M/IO#	19, 1	I	**STATUS INPUTS** are the status input signal pins from the 80286 processor. The arbiter decodes these inputs together with S0#/HOLD input to initiate bus request and surrender actions. A bus cycle is started when either S1# or S0# is sampled LOW at the falling edge of CLK. The 80286 S1# and M/IO# pins meet the setup and hold time requirements of these pins. 80286 Bus Cycle Status Encoding **M/IO# S1# S0#/HOLD 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; bus idle 1 0 0 Halt or shutdown 1 0 1 Memory read 1 1 0 Memory write 1 1 1 None; bus idle When supporting the HOLD output of another bus master, the S1# and M/IO# pins must be held HIGH during T_S, the Status Cycle, for proper operation.
SYSB/RESB#	3	I	**SYSTEM BUS/RESIDENT BUS#** is an input signal which determines when the multi-master system bus is required for the current bus cycle. The signal can originate from address mapping circuitry such as a decoder or PROM attached to the processor address and status pins. The arbiter will request or retain control of the multi-master system bus when the SYSB/RESB# pin is sampled HIGH at the end of the T_S bus state. During an interrupt acknowledge cycle, this input is sampled on every falling edge of CLK starting at the end of the T_S state until either SYSB/RESB# is sampled HIGH or the bus cycle is terminated by the READY# signal. Setup and hold times for this pin must be met for proper operation.

Table 1. 82289 Pin Definition (continued)

Symbol	Pin(s)	Type	Name and Function
READY#	2	I	**READY#** is an active-LOW signal which indicates the end of the bus cycle. The 80286 halt or shutdown cycle does not require READY# to terminate the bus cycle. Setup and hold times for this pin must be met for proper operation.
LOCK#	16	I	**LOCK** # is a processor-generated signal which when asserted (LOW) prevents the arbiter from surrendering the multi-master system bus to any other bus arbiter, regardless of its priority. LOCK# is sampled by the arbiter at the end of the T_S (status) bus state. Setup and hold times for this pin must be met for proper operation.
ALWAYS#/ CBQLCK#	15	I	**ALWAYS RELEASE# or COMMON BUS REQUEST LOCK#** can be programmed at processor reset to be either the ALWAYS RELEASE (ALWAYS#) strapping option or the COMMON BUS REQUEST LOCK (CBQLCK#) control input. Setup and hold times for this pin must be met for proper programming. When this pin is LOW during the falling edge of processor reset (ALWAYS# option) the arbiter is programmed to surrender the multi-master system bus after each bus transfer cycle. The 82289 will remain in the ALWAYS RELEASE mode until it is reprogrammed during the next processor reset. The bus arbiter is programmed to support the COMMON BUS REQUEST LOCK function by forcing this input pin HIGH during the falling edge of the processor reset. CBQLCK# itself is an active-LOW signal which when active prevents the arbiter from surrendering the multi-master system bus to a common bus request through the CBRQ# input pin.
RESET	4	I	**PROCESSOR RESET** is an active-HIGH input synchronous to the system clock (CLK). RESET is the processor initialization of the arbiter to release the multi-master bus and clear any pending request.
INIT#	6	I	**INITIALIZE#** is an active-low Multibus signal used to reset all arbiters on the Multibus system. It will cause the release of the multi-master bus, but will not clear the pending bus master request so that the arbiter can again request the multi-master bus. No arbiters have the use of the multi-master bus immediately after initialization. INIT# is an asynchronous signal to CLK.
BCLK#	5	I	**BUS CLOCK#** is the multi-master system bus clock to which the multi-master bus interface signals are synchronized. BCLK# can be asynchronous to CLK.
BREQ#	7	O	**BUS REQUEST#** is an active-LOW output signal used in the parallel and rotating priority resolving schemes. The arbiter activates BREQ# to request the use of the multi-master system bus. The arbiter holds BREQ# active as long as it is requesting or has possession of the multi-master system bus.
CBRQ#	12	I/O (open-drain)	**COMMON BUS REQUEST#** is a Multibus signal that indicates when an arbiter is requesting the Multibus. This pin is an open-drain input/output requiring an external pullup resistor. As an input CBRQ# indicates that another arbiter is requesting the multi-master system bus. The input function of this pin is enabled by the CBQLCK# signal. Setup and hold times for this pin must be met for proper operation. As an output CBRQ# is asserted to indicate that this arbiter is requesting the Multibus. The arbiter pulls CBRQ# low when it issues a BREQ#. The arbiter release CBRQ# when it obtains the Multibus.

Table 1. 82289 Pin Definition (continued)

Symbol	Pin(s)	Type	Name and Function
BPRN#	9	I	**BUS PRIORITY IN#** is an active-low input indicating that this arbiter has the highest priority of any arbiter requesting the system bus. BPRN# HIGH signals the arbiter that a higher priority arbiter is requesting or has possession of the system bus. Setup and hold times for this pin must be met for proper operation.
BPRO#	8	O	**BUS PRIORITY OUT#** is an active-low output signal used in the serial priority resolving scheme. BRPO# is connected to BPRN# of the next lower priority to grant or revoke priority from that arbiter.
BUSY#	11	I/O (open-drain)	**BUSY#** is a Multibus signal which is asserted when the system bus is in use. BUSY# is an open drain input/output requiring an external pullup resistor. As an input BUSY# asserted indicates when the Multibus is in use. Setup and hold times must be met for proper operation. As an output BUSY# is asserted to signal when this arbiter has taken control of the Multibus.
AEN#	13	O	**ADDRESS ENABLE#** is the output of the arbiter which goes directly to the processor's address latches, the 82288 Bus Controller and the 82284 Clock Generator. AEN# asserted causes the bus controller and address latches to enable their output drivers. AEN# also drives the clock generator ARDYEN# input to enable its asynchronous ready input (ARDY#). AEN# can also be used as an active-LOW Hold Acknowledge to a bus master other than 80286. It signals to the bus master that control of the system bus has been relinquished when AEN# is inactive (HIGH). Note that AEN# goes active relative to BCLK# and goes inactive relative to CLK.
LLOCK#	14	O	**LEVEL LOCK#** is an active-low output signal decoded from processor LOCK# signal. LLOCK# can be used as Multibus LOCK# when buffered with a tri-state buffer enabled by the AEN# signal. LLOCK# will be cleared by RESET but not by INIT#.
V_{CC}	20	I	+5 volts supply voltage
GND	10	I	Ground

FUNCTIONAL DESCRIPTION

The 82289 Bus Arbiter in conjunction with the 82288 Bus Controller and the 82284 Clock Generator interfaces the 80286 processor or some other bus master to a multi-master system bus. The arbiter multiplexes a processor onto a multi-master system bus. It avoids contention with other bus masters.

The 82289 has two separate state machines which communicate through bus request and release logic. The processor interface state machine is synchronous with the local system clock (CLK) and the multi-master system bus interface state machine is synchronous with the bus clock (BCLK#).

The 82289 performs all signalling to request, obtain, and release the system bus. External logic is used to determine which bus cycles require the system bus and to resolve priorities of simultaneous requests for control of the system bus.

82289 with 80286

In an iAPX 286 system using an 82289 Bus Arbiter, the 80286 processor is unaware of the arbiter's existence and issues cammands as though it had exclusive use of the multi-master system bus such as Multibus™. If the processor cycle requires Multibus access, the arbiter requests control of the Multibus. Until the request is granted the 82289 keeps AEN# disabled to prevent the 82288 Bus Controller and the address latches from accessing the Multibus. AEN# inactive also disasserts the

asynchronous ready enable (ARDYEN#) input of the 82284 clock chip so that the system bus will appear as "NOT READY" to the 80286 processor.

Once the 82289 Bus Arbiter ha acquired the bus, it will assert AEN# allowing the 82288 Bus Controller and the address latches to access the system bus and asserting the ARDYEN# input of the 82284 Clock chip.

Typically, once the data transfer command has been issued by the 82288 and the data transfer has taken place, a transfer acknowledge (XACK#) signal is returned to the processor on the multi-master system bus to indicate "Ready" from the accessed slave device. The processor remains in a series of "Wait States" (Repeated Tc states) unitl the addressed device responds with XACK# asserted signal to the 82284 ARDY# input and the 82284 asserts READY# to the processor. The processor then completes its bus cycle.

82289 with other Bus Masters

When supporting other bus masters, the S0#/HOLD and READY# pins of the bus arbiter can be connected to the 'Hold' pin of that master. The inverted AEN# signal from the 82289 can be used as the hold acknowledge (HLDA) input for the other bus master.

The bus master sends a HOLD signal to the bus arbiter when it needs the system bus for a memory access. If the arbiter currently controls the system bus, AEN# will be active. Otherwise, AEN# will be inactive and the arbiter will request control of the system bus. The bus master will have to wait until the 82289 has asserted AEN# (LOW), before it starts its bus cycle.

When the bus master no longer requires the Multibus it will have to inactivate the HOLD signal. The arbiter interprets the Multibus access as a single bus cycle which is terminated by HOLD going inactive (LOW). Thus the arbiter will not release the Multibus to any other bus master during a bus access cycle.

Processor Cycle Definition

Any iAPX 286 system which gains access to the Multibus through the 82289 Bus Arbiter 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 82289 control of the iAPX 286 interface to Multibus. The local bus master informs the bus arbiter of its internal clock phase when it asserts the status signals. The 80286 S0# and S1# status signals are always first asserted in phase 1 of the local bus master's internal clock.

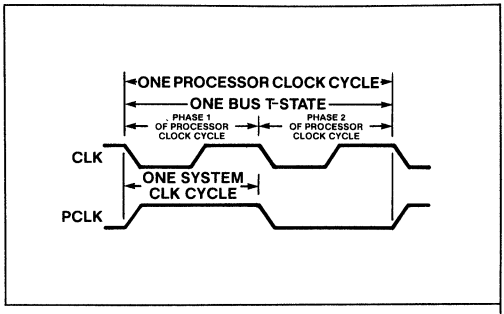


Figure 3: CLK Relationship to Internal Processor Phase, and Bus T-States

Bus State Definition

The 82289 Bus Arbiter has three processor bus states (see figure 4): Idle (T_I), Status (T_S), Command (T_C). Each bus state is two CLK cycles long. Bus state phases correspond to the internal CPU processor clock phases.

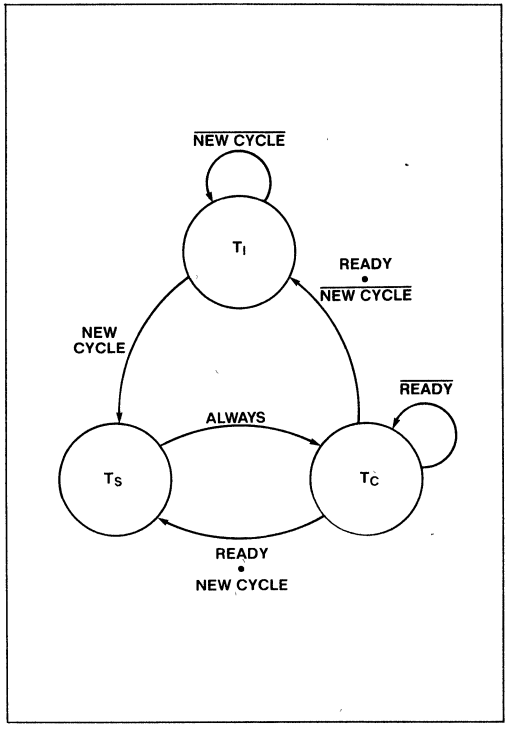


Figure 4: 82289 Processor Bus States

Bus Cycle Definition

The S1# and S0# status inputs are sampled by the 82289 on the falling edge of CLK and signal the start of a bus cycle by going active (LOW). The T_S bus state is defined to be the two CLK cycles during which either S1# or S0# is active (see figure 5). When either S1# or S0# is sampled LOW, the next CLK cycle is considered the second phase of the associated processor clock cycle.

The arbiter 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 READY# input determines whether the current T_C bus state is to be repeated. The READY# input has the same timing and effect for all bus cycles. 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.

When READY# is sampled LOW, the current bus cycle is terminated. Note that the bus arbiter may enter the T_S bus state directly from T_C if the status lines are sampled active (LOW) at the next falling edge of CLK (see Figure 5). If neither of the status lines are sampled active at that time the 82289 will enter the T_I bus state. The T_I bus state will be repeated until the status inputs are sampled active.

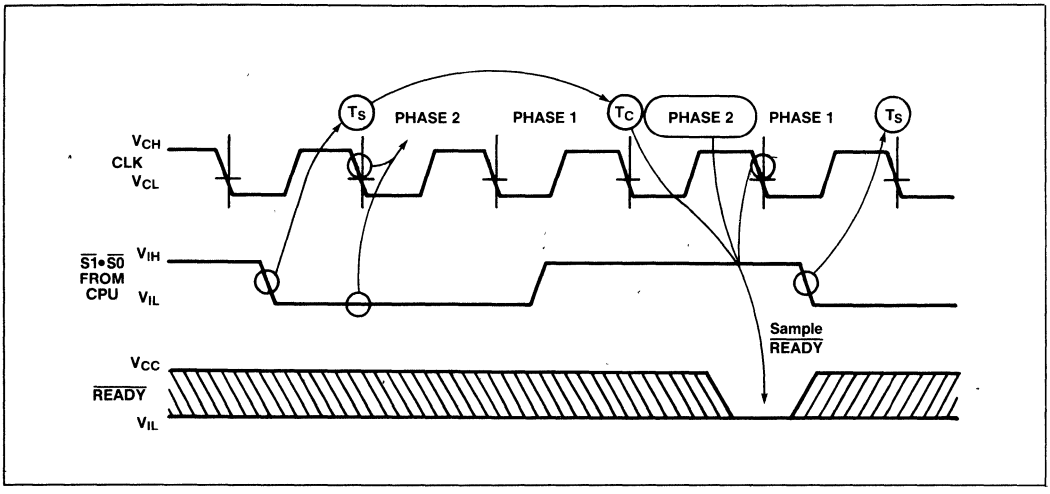


Figure 5: 80286 Bus Cycle Definition (without wait states)

Arbitration Between Bus Masters

The Multibus protocol allows multiple processing elements to compete with each other to access common system resources. Since the local 80286 processor does not have exclusive use of the system bus, if the Multibus is "BUSY" the 80286 processor will have to wait before it can access the system bus.

The 82289 Bus Arbiter provides an integrated solution for controlling access to a multi-master system bus. The bus arbiter allows both higher and lower priority bus masters to acquire the system bus depending on which release mode is used. In general, higher priority masters obtain the bus immediately after any 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 or the proper surrender conditions exist. The 82289 handles this arbitration in a manner completely transparent to the bus master (e.g. 80286 processor).

At the end of each transfer, the arbiter may retain or release the system bus. This decision is controlled by the processor state, bus arbitration inputs and arbiter strapping options. (See Releasing The Multibus, ahead).

Priority Resolving Techniques

Some means of resolving priority between bus masters requesting the multi-master bus simultaneously must be provided. The 82289 Bus Arbiter supports parallel, serial, and rotating system bus priority resolving techniques. All of these techniques are based on the concept that at a given time, one bus master will have priority above all the others.

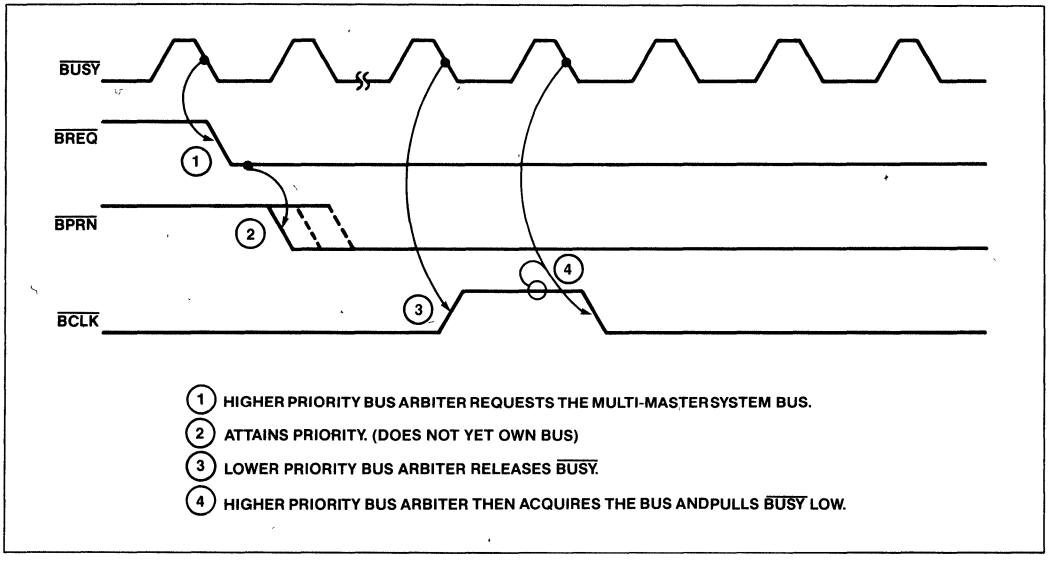


Figure 6: Bus Exchange Timing For The Multibus

An individual arbiter is the highest priority arbiter requesting the Multibus when its BPRN# input is asserted (LOW). The highest priority requesting arbiter cannot immediately seize the system bus. It must wait until the present bus transaction is completed. Upon completing its current transaction the present bus owner surrenders the bus by releasing BUSY#.

BUSY# is an active-low 'Wired-OR' Multibus signal which goes to every bus arbiter on the system bus. When BUSY# goes inactive, the arbiter which has requested the system bus, *and* presently has bus priority (BPRN# LOW), seizes the bus by pulling BUSY# LOW (See waveform in Figure 6).

The generation of a multi-master bus request (BREQ#) is controlled by the type of bus cycle and the SYSB/RESB# input. Whenever the processor signals the status for memory read, memory write, I/O read, I/O write or interrupt acknowledge cycle, and SYSB/RESB# is HIGH at the end of T_S, a bus request is generated.

When the status inputs indicate an interrupt acknowledge bus cycle, the arbiter allows external logic to decide (through the SYSB/RESB# input) whether the interrupt acknowledge cycle should use the Multibus.

Figure 7 shows how SYSB/RESB# is repeatedly sampled until it is sampled HIGH or the bus cycle is terminated. If the bus cycle is completed (READY# is sampled LOW) before SYSB/RESB# is sampled HIGH, the arbiter will not request the Multibus.

The 82289 bus Arbiter does not generate a separate BREQ# for each bus cycle. Instead the 82289 generates BREQ# when it requests the bus and holds BREQ# active during the time that it has possession of the bus. Note that all multi-master system bus requests (via BREQ#) are synchronized to the system bus clock (BCLK#).

Parallel Priority Resolving Technique

The parallel priority resolving technique requires a separate bus request line (BREQ#) for each arbiter on the multi-master system bus (see Figure 8). Each BREQ# line enters a priority encoder which generates the binary address of the highest priority BREQ# line currently active. The binary address is decoded to select the BPRN# line corresponding to the highest priority arbiter requesting the bus. In a parallel scheme, the BPRO# output is not used.

The arbiter receiving priority (BPRN# LOW) then allows its associated bus master onto the multi-master system bus as soon as the bus becomes available (i.e., the bus is no longer busy). Any number of bus masters may be acomodated in this way, limited only by the complexity of the external priority resolving circuitry. Such circuitry must resolve the priority within one BCLK# period.

Serial Priority Resolving Technique

The serial priority resolving technique eliminates the need for the priority circuitry of the parallel technique by daisy-chaining the bus arbiters together, that is, connecting the higher priority

arbiter's BPRO# output to the BPRN# of the next lower priority arbiter (see Figure 9). The highest priority bus arbiter would have its BPRN# tied LOW in this configuration, signifying to the arbiter that it always has the highest priority when requesting the system bus. In a serial scheme, the BREQ# output is not used.

Since arbitration must be resolved within one BCLK# period the number of arbiters connected together in the serial priority is limited by arbiter BPRN# to BPRO# propagation delay (18 ns). For a 10 MHz Multibus BCLK#, five 82289 Bus Arbiters may be connected together in serial configuration.

Maximum number of chained-priority devices =

$$\frac{\text{BCLK\# period}}{\text{BPRN\# to BPRO\# delay}}$$

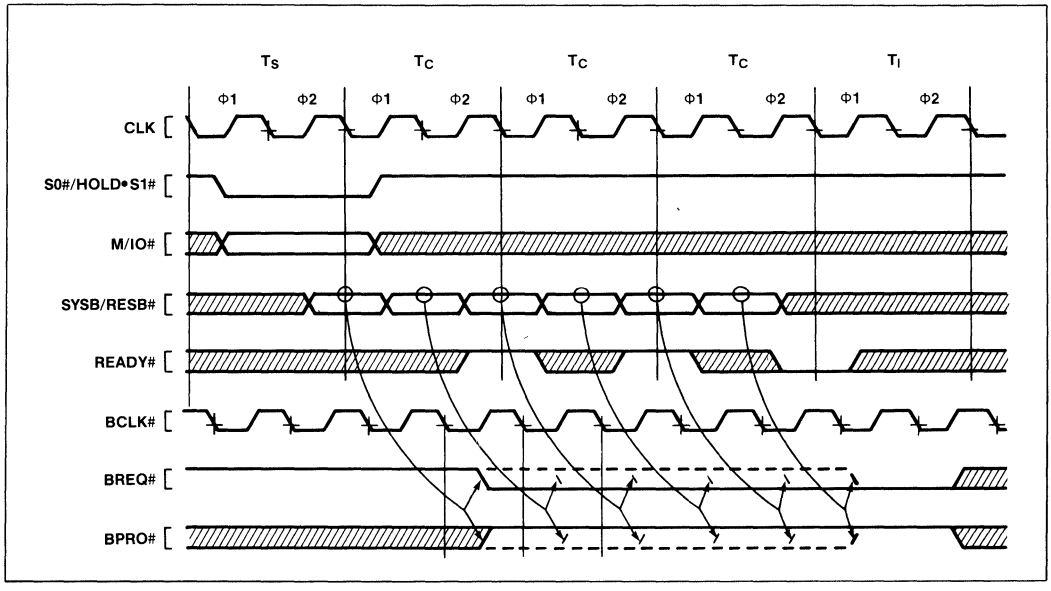


Figure 7: Bus Request Timing During an Interrupt Acknowledge Cycle

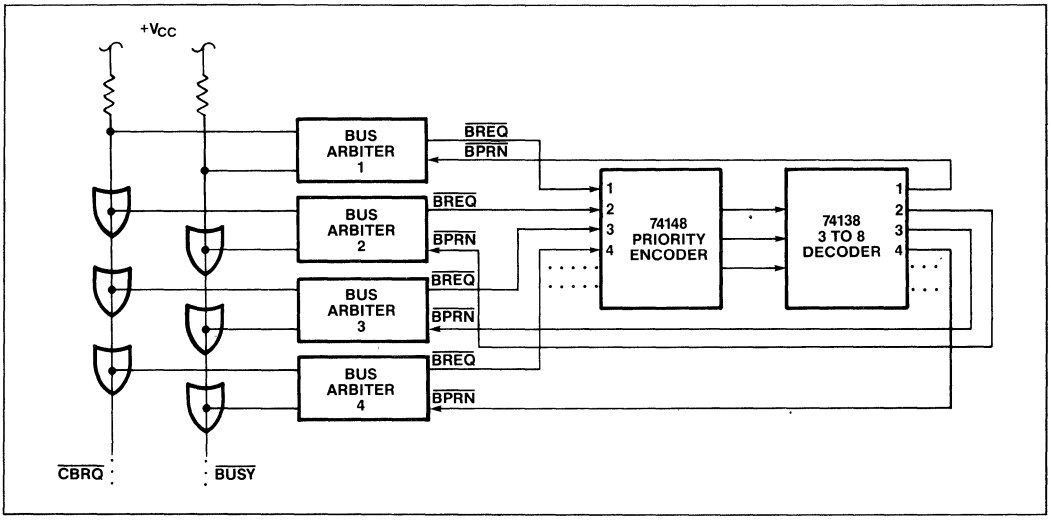


Figure 8: Parallel Priority Resolving Technique

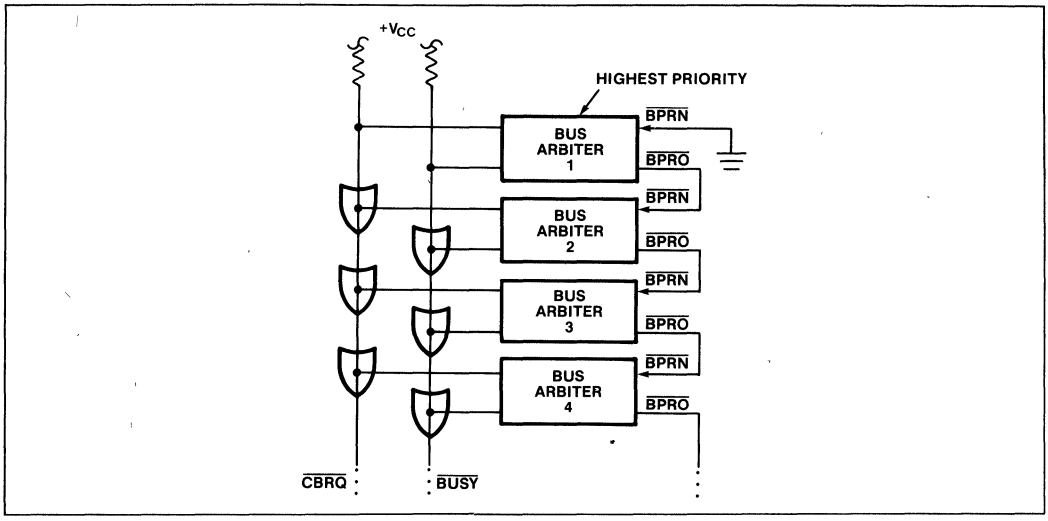
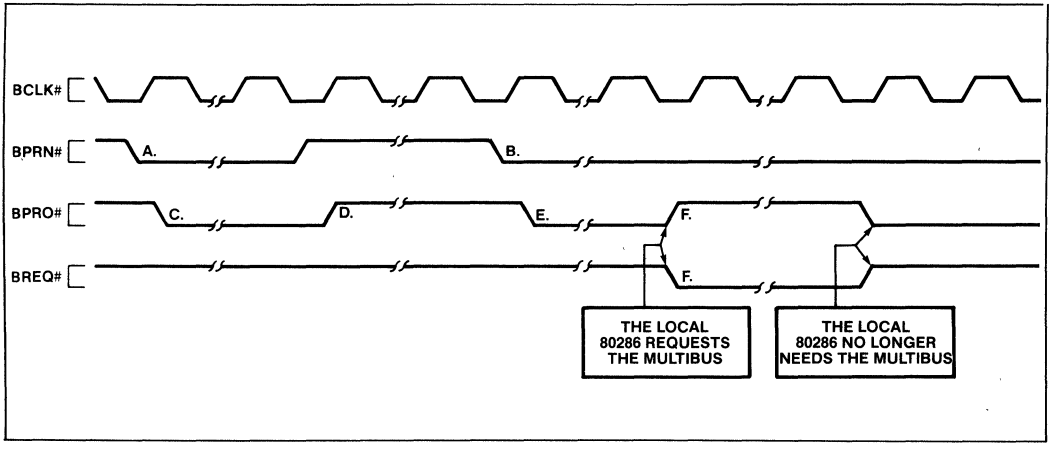


Figure 9: Connections for Serial Priority Resolving Technique



Note: Events A through F described above.

Figure 10: Serial Priority Bus Behavior

When using the serial priority resolving scheme, a higher priority arbiter (for example, arbiter 2, Figure 9) passes priority to the next lower priority arbiter (arbiter 3) by asserting its BPRO# signal (LOW). This asserts BPRN# of next arbiter (arbiter 3) as shown in Figure 10-a & 10-b. An arbiter's BPRO# is asserted if the arbiter has priority (BPRN# is asserted) but is not accessing or requesting the system bus (as indicated by BREQ# inactive as shown in Figure 10-c and 10-e for arbiter 3). Whenever a higher priority arbiter (arbiter 3) issues a bus request its BPRO# goes inactive causing the next lower priority arbiter (arbiter 4) to lose its bus priority (Figure 10-f). Any arbiter (arbiter 3) will also bring its BPRO# inactive if its BPRN# goes inactive (from arbiter 2), thereby passing the loss of bus priority on to the lower priority arbiters (e.g. arbiter 4) as shown in Figure 10-d.

Rotating Priority Resolving Technique

The rotating priority resolving technique is similar to 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 a given period of time.

Selecting the Appropriate Priority Resolving Technique

The choice of a priority resolving technique involves a tradeoff between external logic complexity and ease of Multibus access for the different bus masters in the system. The rotating priority resolving technique requires a substantial amount of external logic, but guarantees all the bus masters an equal opportunity to access the system bus. The serial priority resolving technique uses no external logic but has fixed bus master priority levels and can accommodate only a limited number of bus arbiters. The parallel priority resolving technique is in general a compromise between the other two techniques. (For example parallel priority configuration in Fig. 8 allows up to eight arbiters to be present on the Multibus, with fixed priority levels, while not requiring a large amount of complex external logic to implement.)

Releasing the Multibus

Following a data transfer cycle on the Multibus, the 82289 Bus Arbiter can either retain control of the system bus or release the bus for use by some other bus master. The 82289 can operate in one of three modes, defining different conditions under which the arbiter relinquishes control of the multi-master system bus. These release modes are described in Table 2.

Release Mode	Conditions under which the Bus Arbiter releases the system bus (unless cycles are LOCKed)
Mode 1	The Bus Arbiter always releases the bus at the end of each transfer cycle
Mode 2	The Bus Arbiter retains the bus until: • a higher-priority bus master requests the bus, driving BPRN# HIGH • a lower-priority bus master requests the bus by pulling CBRQ# LOW
Mode 3	The Bus Arbiter retains the bus until: • a higher-priority bus master requests the bus, driving BPRN# HIGH. (CBRQ# LOW ignored)

Table 2: 82289 Release Modes

If the arbiter was programmed to operate in the Always Release mode (Mode 1) during the previous reset, it will surrender the Multibus after each complete transfer cycle. If the arbiter is not in the Always Release mode, it will not surrender the bus until the local 80286 processor enters a halt state, the arbiter is forced off of the bus by the loss of BPRN# (Mode 2 or 3), or by a common bus request when the CBRQ# input is enabled by the CBQLCK# input (Mode 2).

CBRQ# can save the bus exchange overhead in many cases. If CBRQ# is high, it indicates to the bus master that no other master is requesting the bus and therefore the present bus master can retain the bus. Without CBRQ#, only BPRN# indicates whether or not another master is requesting the bus and, that only if the other master is of higher priority. Between the master's bus transfer cycles, in order to allow lower priority masters to take the bus if they need it, the master must give up the bus. At the start of the master's next transfer cycle, the bus must be regained. If no other master has the bus, this can take approximately two BCLK# periods. To avoid this overhead of unnecessarily giving up and regaining the bus when no other masters need it, CBRQ# is extremely useful. Any master that wants but does not have the bus, must assert CBRQ# (LOW). If CBRQ# line is not asserted the bus does not have to be released, thereby eliminating the delay of regaining the bus at the start of the next cycle.

The LOCK# input to the arbiter can be used to override any of the conditions shown in Table 2. While LOCK# is asserted, the arbiter will not surrender control of the Multibus to any other requesting arbiter. Note that the arbiter will surrender the Multibus (synchronous to BCLK#) either in response to RESET or INIT# signals independent of the current release mode or the state of the arbiter inputs.

The three bus release modes have the same operation when supporting either the 80286 processor or some other bus master.

Selecting the Appropriate Release Mode

The choice of which release mode to use may affect the bus utilization of the individual subsystems, and the system as a whole. Mode dependent performance variations are due to the bus acquisition/release overhead. The effect of these acquire and release times on system bus efficiency is illustrated in Figure 11.

An isolated transfer on the multi-master system bus is depicted in Figure 11-a. Figure 11-b shows utilization for the bus arbiter operating in Mode 1. The arbiter must request and release the system bus for each transfer cycle. Lower priority arbiters have easy access to the system bus, but overall bus efficiency is low. Bus utilization for a bus arbiter operating in Mode 2 or 3 is shown in Figure 11-c. In this situation the arbiter acquires the bus once for a sequence of transfers. The arbiter retains the bus until forced off by another bus master's request as defined in Table 2

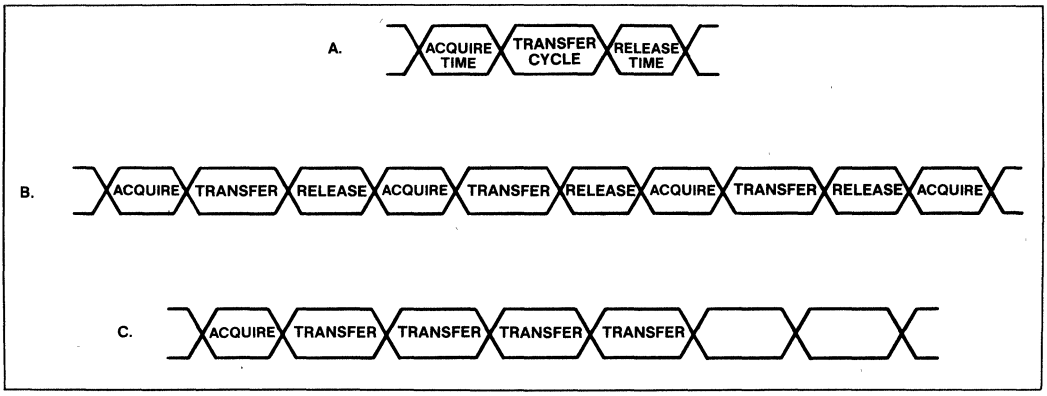


Figure 11: Effects of Bus Contention on Bus Efficiency

The three release modes of the 82289 allow the designer to optimize the system use of the Multibus.

Configuring the 82289 Release Mode

The 82289 Bus Arbiter can be configured in any of its three bus release modes without additional hardware. the 82289 can also be configured to switch between Mode 2 and Mode 3 under software control of the 80286 processor, requiring that a parallel port or addressable latch be used to drive the ALWAYS#/CBQLCK# input pin of the 82289 (see Figure 12).

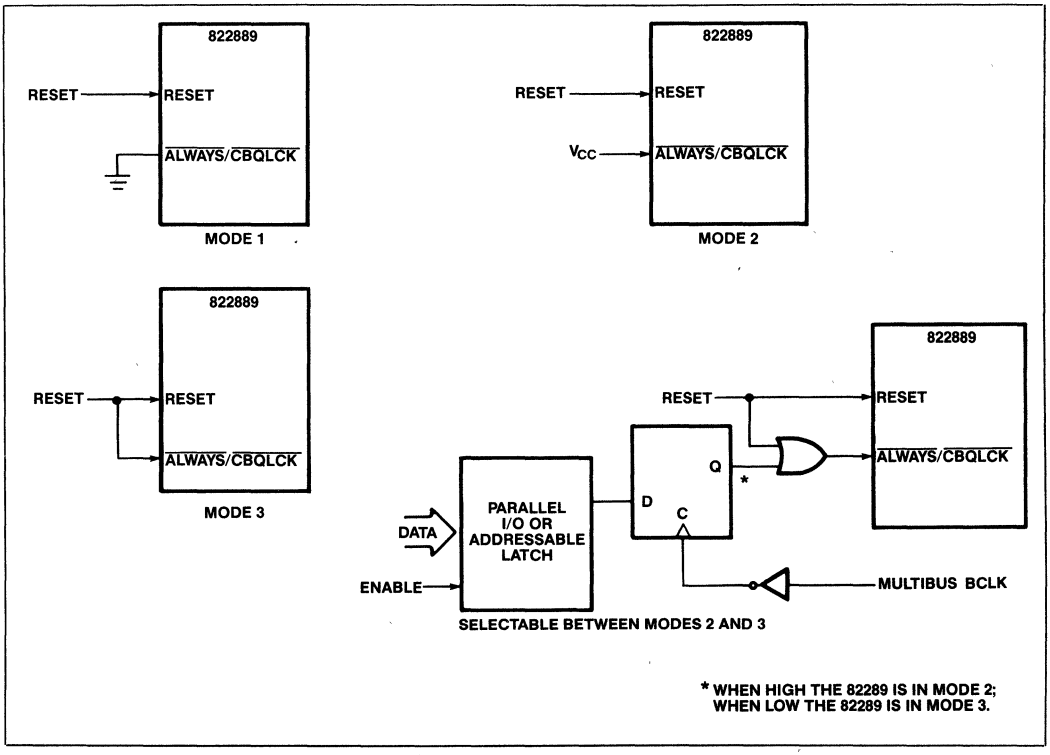


Figure 12: 82289 Release Mode Configurations

Asserting the LOCK# Signal

Independent of the particular release mode of the 82289 Bus Arbiter, the 80286 processor can assert a LOCK# signal synchronously to CLK to prevent the arbiter from releasing the Multibus. This software-controlled LOCK# signal prevents the 82289 from surrendering the system bus to any other bus master, whether that bus master is of higher or lower priority. The LOCK# signal is typically used for implementing software semaphores for shared resources or for critical processes that must run in real-time.

The 82289 LLOCK# output is the Multibus signal asserted during all bus cycles which are locked together. The LLOCK# is set or reset depending on processor LOCK# at the end of the T_S cycle. The LLOCK# will delay going inactive until the termination of the current transfer cycle.

The 82289 will continue to assert the LLOCK# signal, retaining control of the Multibus, until the end of the first 'unLOCKed' 80286 bus cycle (80286 disables its LOCK# output on the last bus cycle indicating that no future locked cycles are needed). While the LOCK# signal will force the arbiter presently in control to hold the system bus, it cannot force another arbiter to surrender the bus any earlier than it normally would.

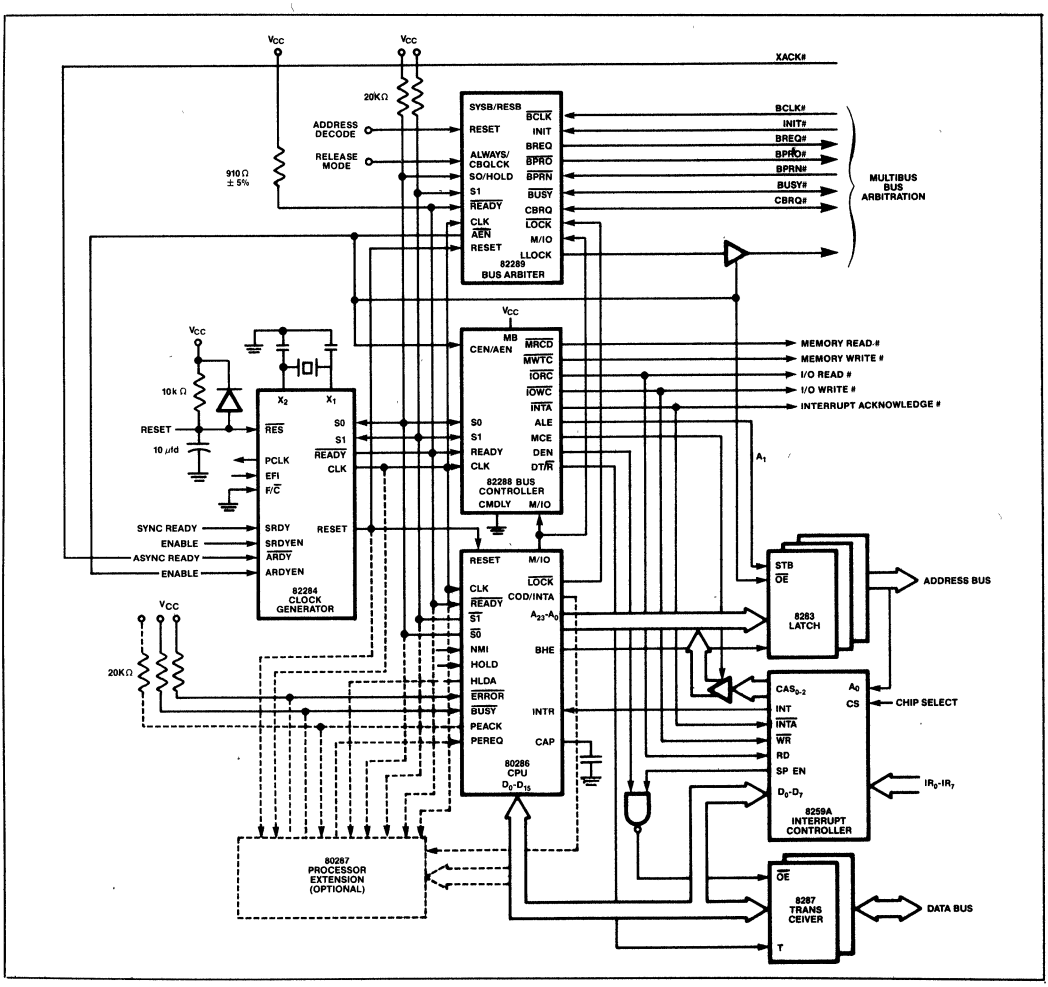
The LLOCK# signal from the 82289 must be connected to a tri-state buffer in order to drive the Multibus LOCK# signal. This tri-state buffer should be enabled by the AEN# signal from the arbiter going active.

82289 Reset and Initialization

The 82289 Bus Arbiter provides the RESET and INIT# pins for initialization. RESET is a CLK synchronous signal from the 80286 processor and INIT# is an asynchronous signal on the multimaster system bus. By having RESET pin high or INIT# pin low, the BREQ#, BUSY#, and AEN# output pins will all be cleared and become inactive. RESET will also clear the LLOCK# signal. Unlike RESET, INIT# will not clear any pending bus request; the bus request would be asserted after the INIT# signal goes inactive.

Note that when the 82289 is initialized by the RESET input it does not wait until the end of the current bus cycle to reset. Any bus cycle in process when RESET goes active will be aborted by the arbiter. Although the INIT# signal will also interrupt an active bus cycle, the arbiter can request the Multibus and complete the bus cycle when INIT# goes inactive.

As mentioned in the Table 1 Pin Description and Figure 12, the functions of the S0#/HOLD pin and the release mode (ALWAYS#/CBQLCK# pin) are programmed at the falling edge of RESET.



Schematic 1: Typical iAPX 286 Subsystem MULTIBUS 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 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.*

Electrical Characteristics and Waveforms

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

Symbol	Parameter	Preliminary		Units	Test Conditions
		Min.	Max.		
V_{IL}	Input Low Voltage	-0.5	.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.5	V	
V_{ILC}	CLK Input Low Voltage	-0.5	.6	V	
V_{IHC}	CLK Input High Voltage	3.8	V_{CC} + 1.0	V	
V_{OL}	Output Low Voltage: BUSY#, CBRQ# BPRO#, BREQ#, AEN# LLOCK#		 .45 .45 .45	 V V V	 I_{OL} = 32mA I_{OL} = 16mA I_{OL} = 5mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = 400 μA
I_{LI}	Input Leakage Current		±10 ±1	μA mA	$0.45V \leq V_{IN} \leq V_{CC}$ $0V \leq V_{IN} < 0.45V$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
I_{CC}	Power Supply Current		120	mA	
C_{CLK}	CLK, BCLK# Input Capacitance		12	pF	F_C = 1 MHz
C_{IN}	Input Capacitance		10	pF	F_C = 1 MHz
C_O	Input/Output Capacitance		20	pF	F_C = 1 MHz

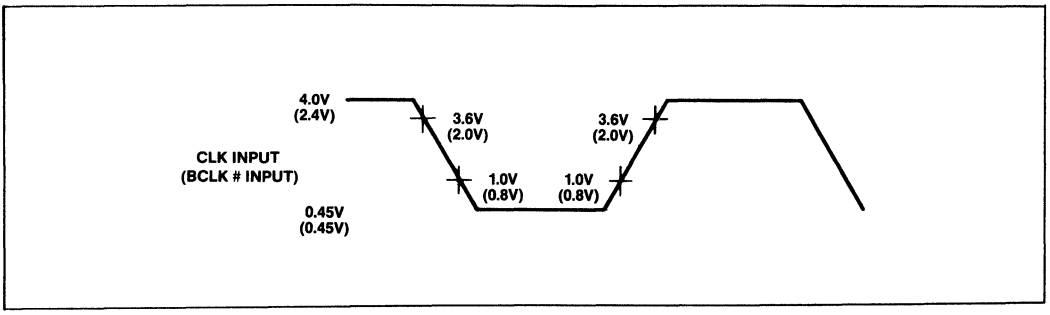
A.C. Characteristics (T_A = 0°C to 70°C, V_{CC} = 5V ± 5%)

AC timings are referenced to 0.8V and 2.0V points of signals as illustrated in datasheet waveforms, unless otherwise noted.

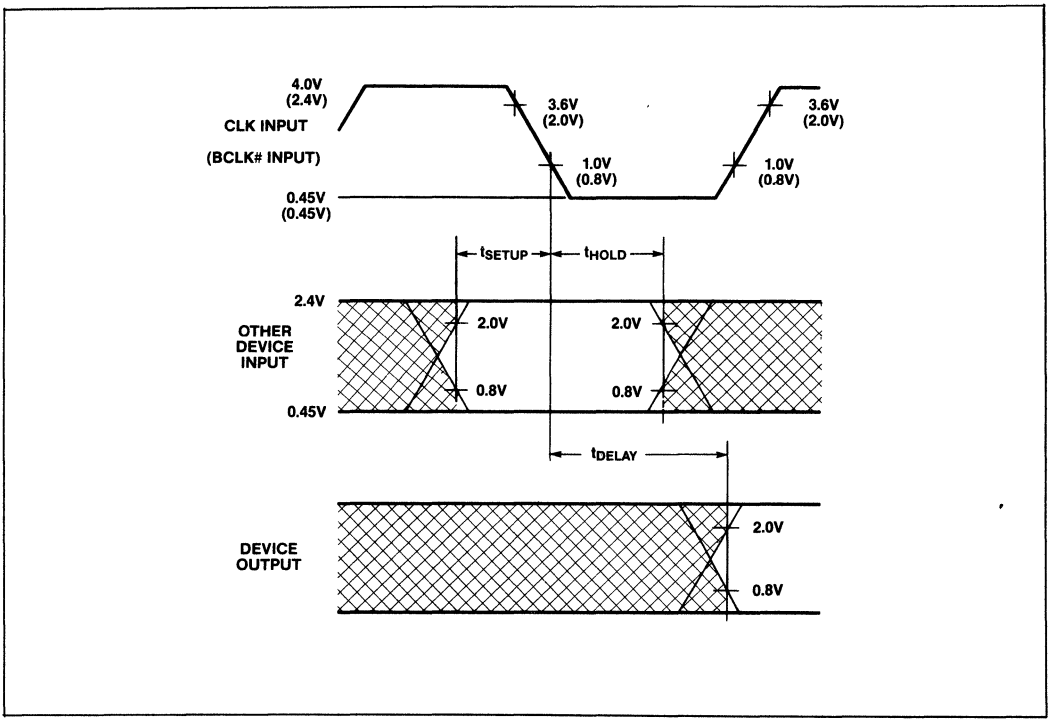
Sym	Parameter	Preliminary 6MHz		Preliminary 8MHz		Unit	Test Conditions	Shown in Figure
		Min.	Max.	Min.	Max.			
1	CLK Cycle Period	83	t5+ 50	62	t5+ 50	ns		13
2	CLK Low Time	20	225	15	230	ns	at 1.0 V	13
3	CLK High Time	25	230	20	235	ns	at 3.6 V	13
4	CLK Rise/Fall Time		10		10	ns	1.0 to 3.6 V	13
5	BCLK# Cycle Time	100	∞	100	∞	ns		13
6	BCLK# High/Low Time	30		30		ns		13
7	S0#/HOLD, S1#, M/IO# Setup	28		22		ns		13
8	S0#/HOLD, S1#, M/IO# Hold	1		1		ns		13
9	READY# Setup	50		38		ns		13
10	READY# Hold Time	35		25		ns		13
11	LOCK#, SYSB/RESB# Setup Time	28		20		ns		13, 18
12	LOCK#, SYSB/RESB# Hold Time	1		1		ns		13, 18
13	RESET Setup Time	28		20		ns		19
14	RESET Hold Time	1		1		ns		19
15	RESET ACTIVE Pulse Width	16		16		CLKs		19
16	INIT# Setup Time	45		45		ns	Note 9	20
17	INIT# Hold Time	1		1		ns	Note 9	20
18	INIT# Active Pulse Width	3(t1) +3(t14)		3(t1) +3(t14)		ns		20
19	BUSY#, BPRN#, CBRQ#, CBQLCK#/ALWAYS# Setup to BCLK# (or to RESET)	20		20		ns		13, 15, 21
20	BUSY#, BPRN#, CBRQ#, CBQLCK#/ALWAYS# Hold to BCLK# (or to RESET)	1		1		ns		13, 15, 21
21	BCLK# to BREQ# Delay		30		30	ns	Note 1	13, 14
22	BCLK# to BPRO# Delay		35		35	ns	Note 2	17
23	BPRN# to BPRO# Delay		25		25	ns	Note 2	17
24	BCLK# to BUSY# Active Delay	1	60	1	60	ns	Note 3	13
25	BCLK# to BUSY# Float Delay		35		35	ns	Note 4	13, 14
26	BCLK# to CBRQ# Active Delay		55		55	ns	Note 5	13
27	BCLK# to CBRQ# Float Delay		35		35	ns	Note 4	13, 20
28	BCLK to AEN# Active Delay	1	25	1	25	ns	Note 6	13
29	CLK to AEN# Inactive Delay	3	25	3	25	ns	Note 6	13, 14
30	CLK to LLOCK# Delay		20		20	ns	Note 7	18
31	RESET to LLOCK# Delay		35		35	ns	Note 7	19
32	CLK to BCLK# Setup Time	38		38		ns	Note 8	13, 16, 20

NOTES:

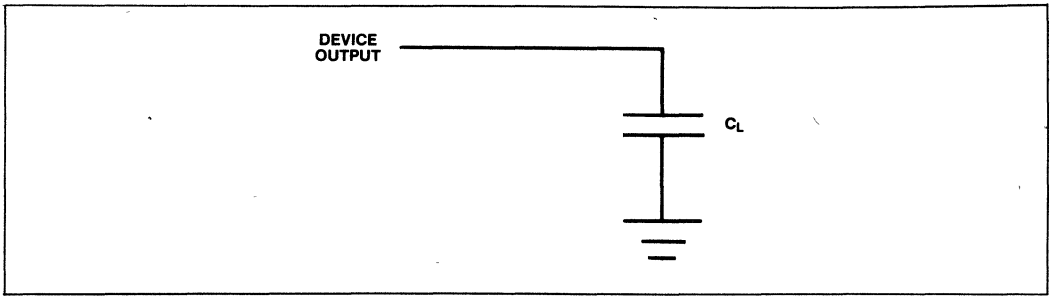
NOTE 1. BREQ# load C_L = 60pF
NOTE 2 BPRO# load C_L = 60pF
NOTE 3 BUSY# load C_L = 300pF
NOTE 4 Float condition occurs when output current is less that I_{LO} in magnitude
NOTE 5 CBRQ# load C_L = 300pF
NOTE 6 AEN# load C_L = 150pF
NOTE 7 LLOCK# load C_L = 60pF
NOTE 8 In actual use, CLK and BCLK# are usually asynchronous to each other However, for component testing purposes, this specification is required to assure signal recognition at specific CLK and BCLK# edges
NOTE 9 INIT# is asynchronous to CLK and to BCLK# However for component testing purposes, this specification is required to assure signal recognition at specific CLK and BCLK# edges



NOTE 10: AC Drive and Measurement Points — CLK Input (BCLK# Input)



NOTE 11: AC Setup, Hold and Delay Time Measurement — General



NOTE 12: AC Test Loading on Outputs

Waveforms

The waveforms (Figure 13-21) show the timing relationships of the inputs and the 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.

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.

The 82289 Bus Arbiter serves as an interface between the iAPX 286 subsystem which operates synchronous to the CLK signal and Multibus which operates synchronous to BCLK# signal. CLK and BCLK# generally operate asynchronously to each other and at different frequencies. Thus, the exact clock period in which an input synchronous to one clock will cause a response synchronous to the other clock depends on the relative phase and frequency of CLK and BCLK# at the time the input is sensed.

One strict relation between CLK and BCLK# must be maintained for proper Multibus arbitration. If the CLK period is too long relative to BCLK# period (t1 greater than t5 + 50ns), another arbiter could gain control of the system bus before this arbiter has released AEN# synchronous to its CLK. This situation arises since the release of AEN# is synchronous to the next falling CLK edge after the processor cycle ends but the release of BREQ# and BUSY# is synchronous to the next falling BCLK# edge after the processor cycle ends. In practice, any CLK frequency greater than 6.66MHz (ie. 80286 processor speeds greater than 3.33MHz) will avoid conflict with a 10MHz BCLK#. Therefore all 80286 speed selections are Multibus compatible.

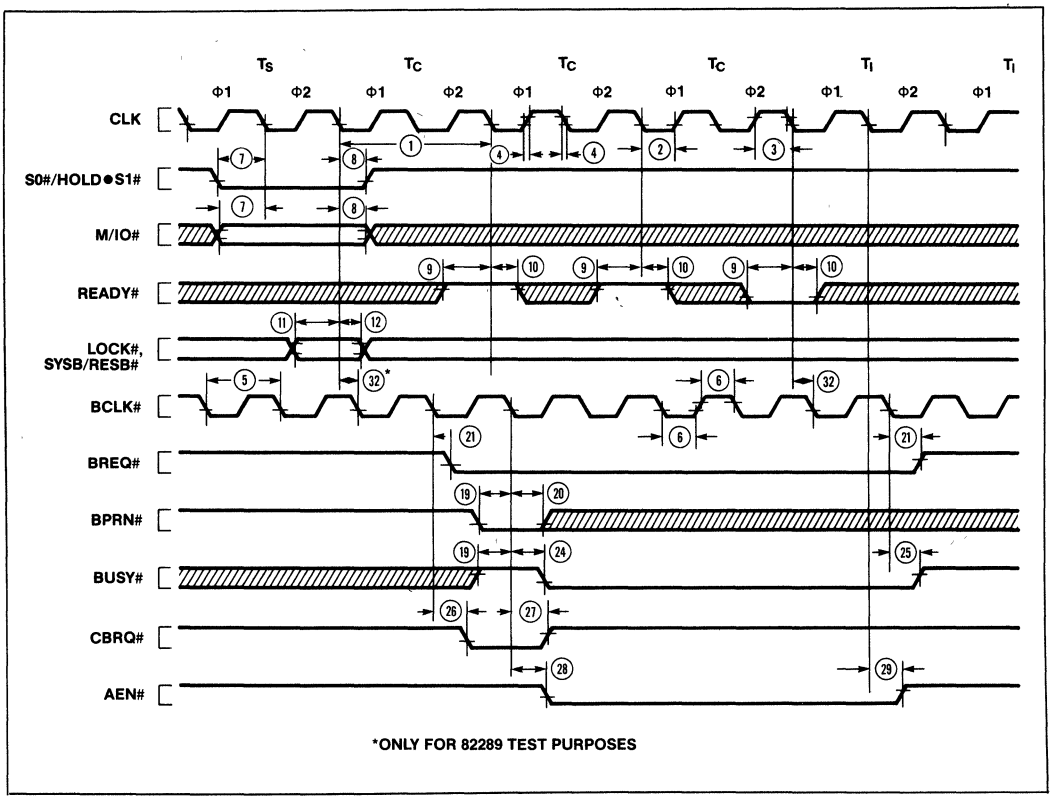


Figure 13: Multibus Acquisition and Always-Release Operation

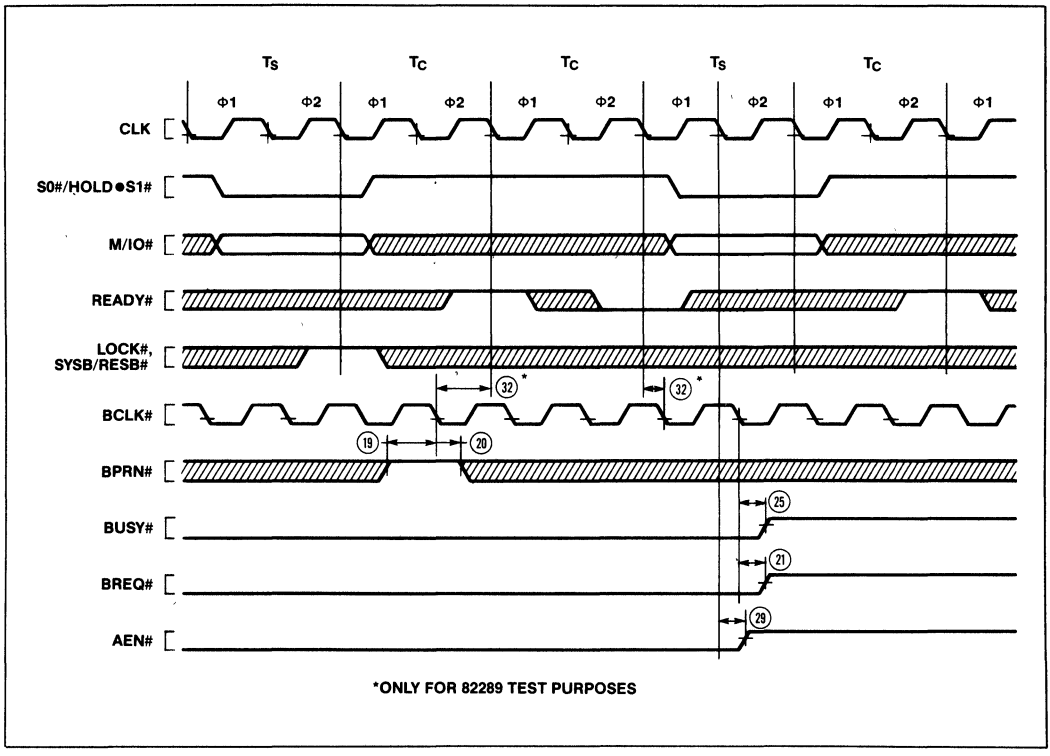


Figure 14: Multibus Release due to BPRN# Inactive

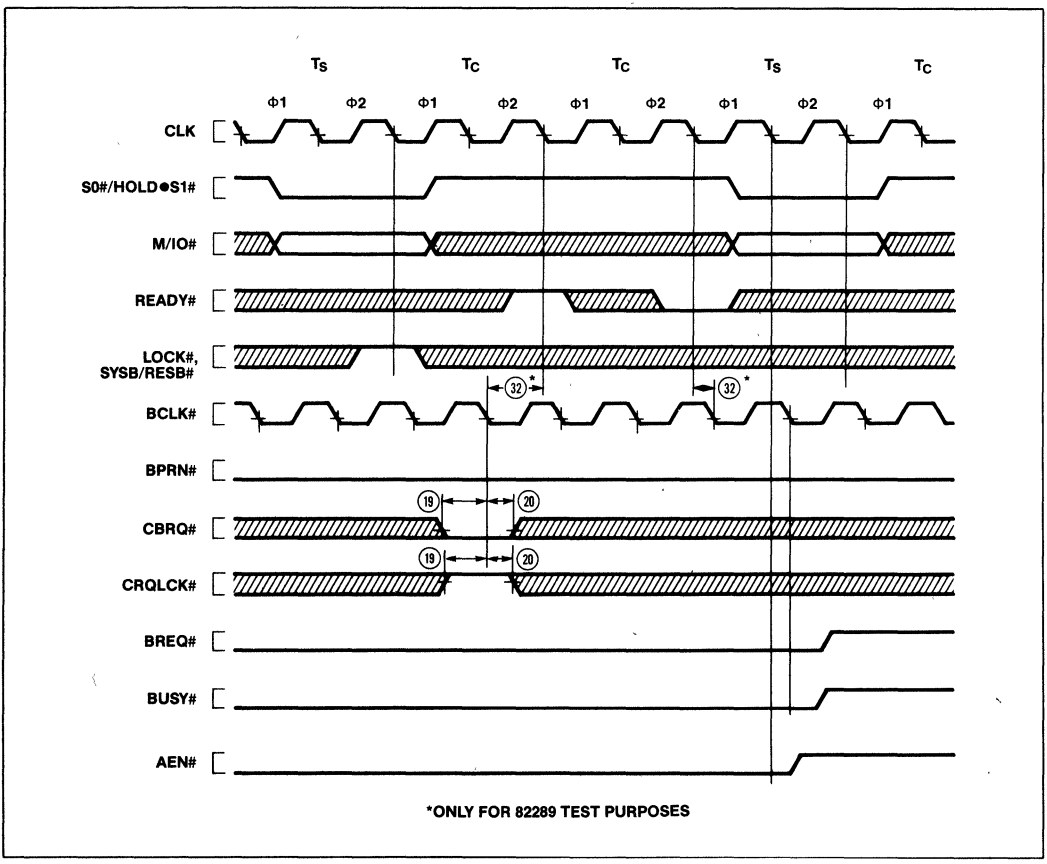


Figure 15: Multibus Release due to CBRQ# Active

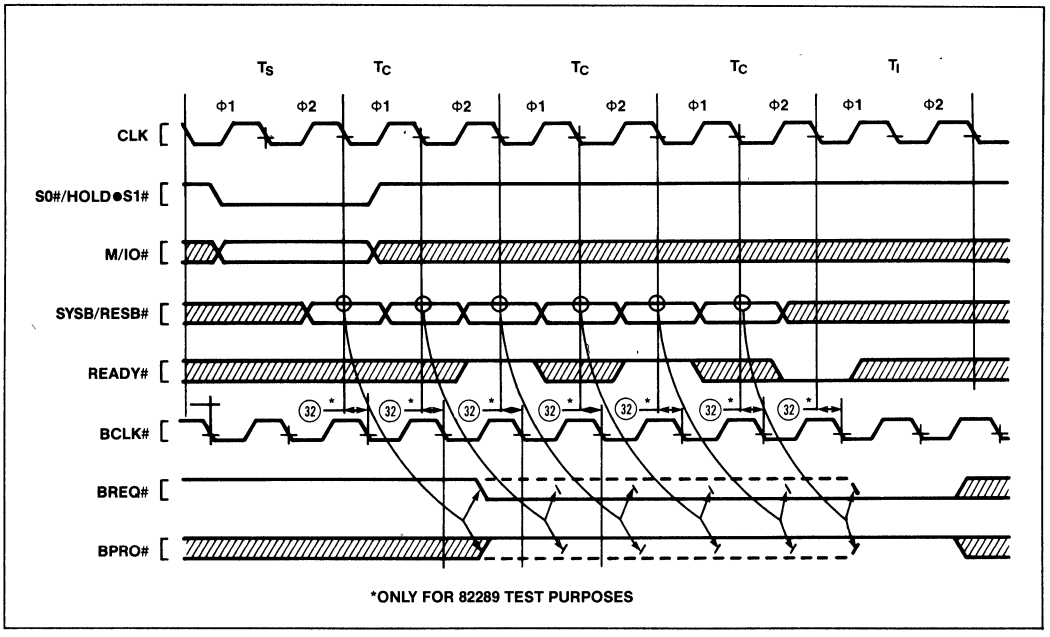


Figure 16: Multibus Acquisition During 80286 INTA Cycles

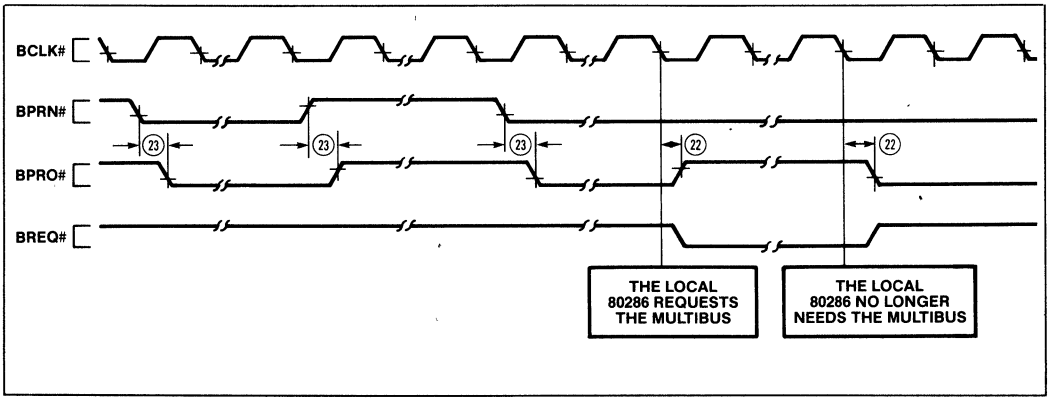


Figure 17: BPRN# to BPRO# Timing Relationship

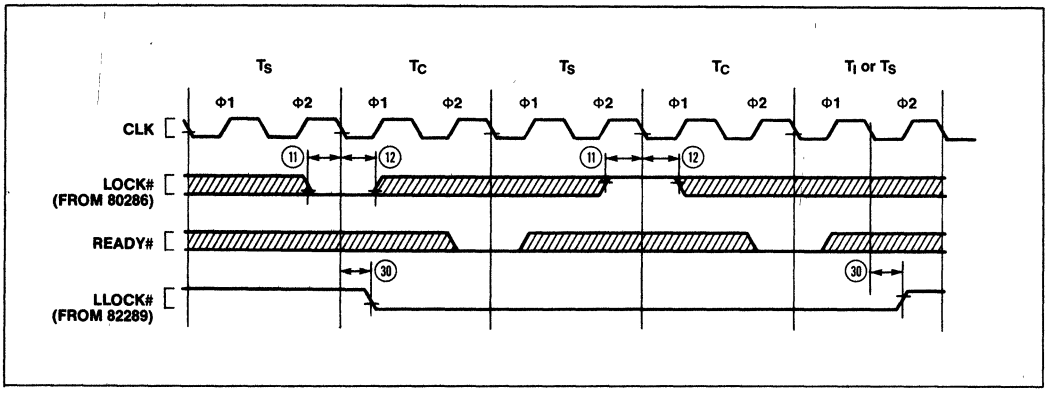


Figure 18: 80286 LOCK# and 82289 LLOCK# Relationship

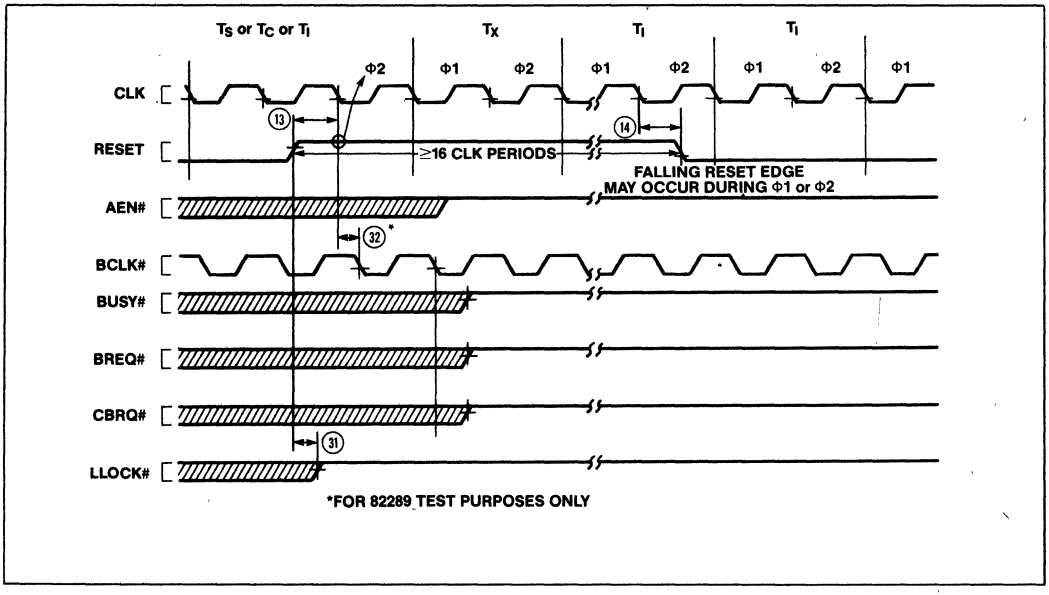


Figure 19: RESET Active Pulse

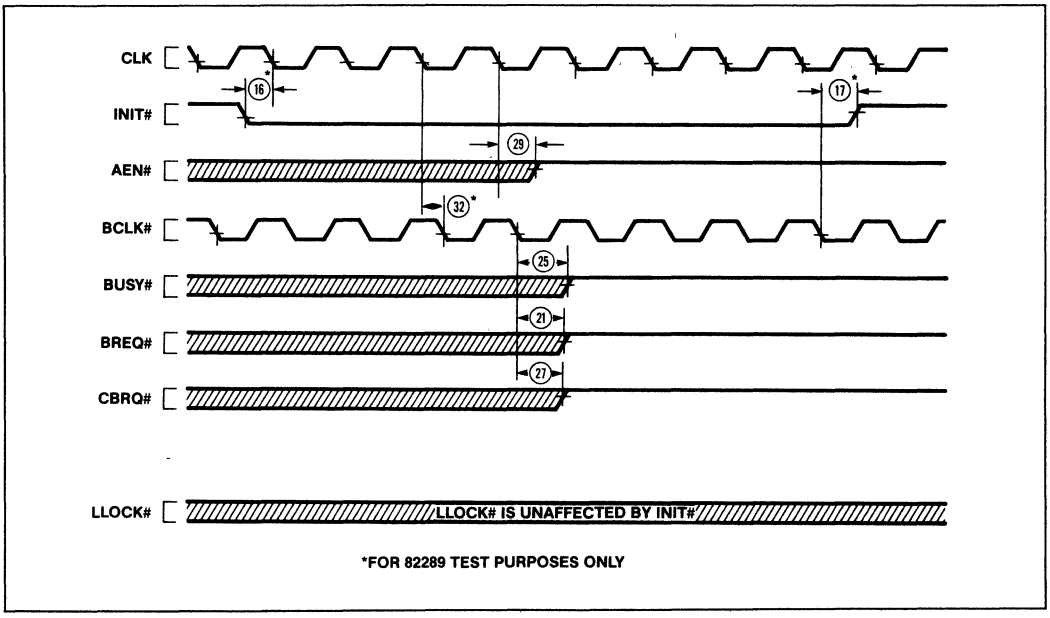


Figure 20: INIT# Active Pulse

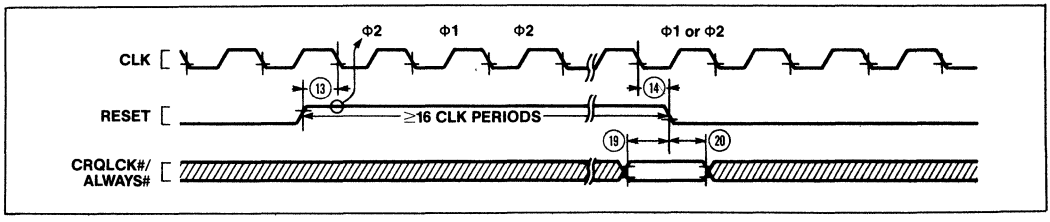


Figure 21: Programming the Always-Release/Common-Bus-Request-Release Option

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp
5015 Bradford Drive
Suite 2
Huntsville 35805
Tel (205) 830-4010

ARIZONA

Intel Corp
11225 N 28th Drive
Suite 214D
Phoenix 85029
Tel (602) 869-4980

Intel Corp
1161 N El Dorado Place
Suite 301
Tucson 85715
Tel (602) 299-6815

CALIFORNIA

Intel Corp
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel (818) 704-8500

Intel Corp
2250 E Imperial Highway
Suite 218
El Segundo 90245
Tel (213) 640-6040

Intel Corp
1010 Hurley Way
Suite 300
Sacramento 95825
Tel (916) 929-4078

Intel Corp
4350 Executive Drive
Suite 150
San Diego 92111
(619) 452-5880

Intel Corp *
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel (714) 835-9642
TWX 910-595-1114

Intel Corp *
1350 Shorebird Way
Mt View 94043
Tel (415) 968-8086
TWX 910-339-9279
910-338-0255

COLORADO

Intel Corp
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel (303) 594-6622

Intel Corp *
650 S Cherry Street
Suite 720
Denver 80222
Tel (303) 321-8086
TWX 910-931-2289

CONNECTICUT

Intel Corp
26 Mill Plain Road
Danbury 06810
Tel (203) 748-3130
TWX 710-456-1199

EMC Corp
222 Summer Street
Stamford 06901
Tel (203) 327-2934

FLORIDA

Intel Corp
242 N Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel (305) 869-5588

Intel Corp
1500 N W 62nd Street
Suite 104
Ft Lauderdale 33309
Tel (305) 771-0600
TWX 510-956-9407

FLORIDA (Cont'd)

Intel Corp
11300 4th Street South
Suite 170
St Petersburg 33702
Tel (813) 577-2413

GEORGIA

Intel Corp
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel (404) 449-0541

ILLINOIS

Intel Corp *
2550 Gulf Road
Suite 815
Rolling Meadows 60008
Tel (312) 981-7200
TWX 910-651-5881

INDIANA

Intel Corp
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel (317) 875-0623

IOWA

Intel Corp
St Andrews Building
1930 St Andrews Drive N E
Cedar Rapids 52402
Tel (319) 393-5510

KANSAS

Intel Corp
8400 W 110th Street
Suite 170
Overland Park 66210
Tel (913) 642-8080

LOUISIANA

Industrial Digital Systems Corp
Tel (504) 899-1654

MARYLAND

Intel Corp *
7321 Parkway Drive South
Suite C
Hanover 21076
Tel (301) 796-7500
TWX 710-862-1944

Intel Corp
7833 Walker Drive
Greenbelt 20770
Tel (301) 441-1020

MASSACHUSETTS

Intel Corp *
27 Industrial Avenue
Chelmsford 01824
Tel (617) 256-1800
TWX 710-343-6333

MICHIGAN

Intel Corp
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel (313) 851-8096

MINNESOTA

Intel Corp
3500 W 80th Street
Suite 360
Bloomington 55431
Tel (612) 835-6722
TWX 910-576-2867

MISSOURI

Intel Corp
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel (314) 291-1990

NEW JERSEY

Intel Corp *
Raritan Plaza III
Raritan Center
Edison 08837
Tel (201) 225-3000
TWX 710-480-6238

NEW MEXICO

Intel Corp
8500 Menual Boulevard N E
Suite B 295
Albuquerque 87112
Tel (505) 292-8086

NEW YORK

Intel Corp *
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel (516) 231-3300
TWX 510-227-6236

Intel Corp
Suite 2B Hollowbrook Park
15 Myers Corners Road
Wappinger Falls 12590
Tel (914) 297-6161
TWX 510-248-0060

Intel Corp *
211 White Spruce Boulevard
Rochester 14623
Tel (716) 424-1050
TWX 510-253-7391

T-Squared
6443 Ridings Road
Syracuse 13206
Tel (315) 463-8592
TWX 710-541-0554

T-Squared
7353 Pittsford-Victor Road
Victor 14564
Tel (716) 924-9101
TWX 510-254-8542

NORTH CAROLINA

Intel Corp
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel (919) 781-8022

OHIO

Intel Corp *
6500 Poe Avenue
Dayton 45414
Tel (513) 890-5350
TWX 810-450-2528

Intel Corp *
Chagrin-Brainard Bldg , No 300
28001 Chagrin Boulevard
Cleveland 44122
Tel (216) 464-2736
TWX 810-427-9298

OKLAHOMA

Intel Corp
4157 S Harvard Avenue
Suite 123
Tulsa 74135
Tel (918) 749-8688

OREGON

Intel Corp
10700 S W Beaverton
Hillsdale Highway
Suite 22
Beaverton 97005
Tel (503) 641-8086
TWX 910-467-8741

PENNSYLVANIA

Intel Corp *
455 Pennsylvania Avenue
Fort Washington 19034
Tel (215) 641-1000
TWX 510-661-2077

Intel Corp *
400 Penn Center Boulevard
Suite 610
Pittsburgh 15235
Tel (412) 823-4970

PENNSYLVANIA (Cont'd)

Q E D Electronics
139 Terwood Road
Willow Grove 19090
Tel (215) 657-5600

TEXAS

Intel Corp *
12300 Ford Road
Suite 380
Dallas 75234
Tel (214) 241-8087
TWX 910-860-5617

Intel Corp *
7322 S W Freeway
Suite 1490
Houston 77074
Tel (713) 988-8086
TWX 910-881-2490

Industrial Digital Systems Corp
5925 Sovereign
Suite 101
Houston 77036
Tel (713)988-9421

Intel Corp
313 E Anderson Lane
Suite 314
Austin 78752
Tel (512) 454-3628

UTAH

Intel Corp
5201 Green Street
Suite 290
Salt Lake City 84123
Tel (801) 263-8051

VIRGINIA

Intel Corp
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel (804) 282-5668

WASHINGTON

Intel Corp
110 110th Avenue N E
Suite 510
Bellevue 98004
Tel (206) 453-8086
TWX 910-443-3002

Intel Corp
408 N Mullan Road
Suite 102
Spokane 99206
Tel (509) 928-8086

WISCONSIN

Intel Corp
450 N Sunnyslope Road
Suite 130
Chancellory Park I
Brookfield 53005
Tel (414) 784-8087

CANADA

ONTARIO

Intel Semiconductor of Canada, Ltd
Suite 202, Bell Mews
39 Highway 7
Nepean K2H 8R2
Tel (613) 829-9714
TELEX 053-4115

Intel Semiconductor of Canada, Ltd
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel (416) 675-2105
TELEX 06983574

QUEBEC

Intel Semiconductor of Canada, Ltd
3860 Cote Vertu Rd
Suite 210
St Laurent H4R 1V4
Tel (514) 334-0560
TELEX 05-824172

*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

†Arrow Electronics, Inc
3611 Memorial Parkway So
Huntsville 35801
Tel (205) 882-2730

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel (205) 837-7210
TWX 810-726-2162

†Pioneer Electronics
1207 Putnam Drive N W
Huntsville 35805
Tel (205) 837-9300
TWX 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S Madison Drive
Tempe 85281
Tel (602) 231-5140
TWX 910-950-0077

†Wyle Distribution Group
8155 N 24th Avenue
Phoenix 85021
Tel (602) 249-2232
TWX 910-951-4282

CALIFORNIA

†Arrow Electronics, Inc
521 Weddell Drive
Sunnyvale 94086
Tel (408) 745-6600
TWX 910-339-9371

†Arrow Electronics, Inc
19748 Dearborn Street
Chatsworth 91311
Tel (213) 701-7500
TWX 910-493-2086

Arrow Electronics, Inc
2961 Dow Avenue
Tustin 92680
Tel. (714) 838-5422
TWX 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel (714) 754-6051
TWX 910-595-1928

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel (408) 743-3300
TWX 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel (619) 571-7500
TWX 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel (213) 700-6271
TWX 910-494-2207

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel (916) 920-3150

Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel (714) 989-9411

Hamilton/Avnet Electronics
19515 So Vermont Avenue
Torrance 90502
Tel (213) 615-3913
TWX 910-349-6263

†Hamilton Electro Sales
10912 W Washington Boulevard
Culver City 20230
Tel (213) 558-2458
TWX 910-340-6364

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel (714) 641-4150
TWX 910-595-2638

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel (818) 700-6500

Kieruliff Electronics, Inc
1180 Murphy Avenue
San Jose 95131
Tel (408) 947-3471
TWX 910-379-6430

CALIFORNIA (Cont'd)

Kieruliff Electronics, Inc
14101 Franklin Avenue
Tustin 92680
Tel (714) 731-5711
TWX 910-595-2599

Kieruliff Electronics, Inc
5650 Jillson Avenue
Commerce 90040
Tel (213) 725-0325
TWX 910-580-3106

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel (213) 322-8100
TWX 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel (714) 843-9953
TWX 910-595-1572

†Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel (619) 565-9171
TWX 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel (408) 727-2500
TWX 910-338-0296

Wyle Military
17810 Teller Avenue
Irvine 92750
Tel (714) 851-9958
TWX 310-371-9127

Wyle Systems
15292 Bolsa Chica
Huntington Beach 92649
Tel (714) 851-9953
TWX 910-595-2642

COLORADO

†Wyle Distribution Group
451 E 124th Avenue
Thornton 80241
Tel (303) 457-9953
TWX 910-936-0770

†Hamilton/Avnet Electronics
8765 E Orchard Road
Suite 708
Englewood 80111
Tel (303) 740-1017
TWX 910-935-0787

CONNECTICUT

†Arrow Electronics, Inc
12 Beaumont Road
Wallingford 06492
Tel (203) 265-7741
TWX 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel (203) 797-2800
TWX 710-456-9974

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel (203) 853-1515
TWX 710-468-3373

FLORIDA

†Arrow Electronics, Inc
1001 N W 62nd Street
Suite 108
Ft Lauderdale 33309
Tel (305) 776-7790
TWX 510-955-9456

†Arrow Electronics, Inc
1530 Bottlebrush Drive N E
Palm Bay 32905
Tel (305) 725-1480
TWX 510-959-6337

†Hamilton/Avnet Electronics
6801 N W 15th Way
Ft Lauderdale 33309
Tel (305) 971-2900
TWX 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech Drive North
St Petersburg 33702
Tel (813) 576-3930
TWX 810-863-0374

FLORIDA (Cont'd)

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel (305) 628-3888
TWX 810-853-0322

†Pioneer Electronics
221 N Lake Boulevard
Suite 412
Alta Monte Springs 32701
Tel (305) 834-9090
TWX 810-853-0284

†Pioneer Electronics
1500 62nd Street N W
Suite 506
Ft Lauderdale 33309
Tel (305) 771-7520
TWX 510-955-9653

GEORGIA

†Arrow Electronics, Inc
2979 Pacific Drive
Norcross 30071
Tel (404) 449-8252
TWX 810-766-0439

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel (404) 447-7500
TWX 810-766-0432

†Pioneer Electronics
58358 Peachtree Corners E
Norcross 30092
Norcross 30092
Tel (404) 448-1711
TWX 810-766-4515

ILLINOIS

†Arrow Electronics, Inc
2000 E Alonquin Street
Schaumberg 60195
Tel (312) 397-3440
TWX 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel (312) 860-7780
TWX 910-227-0060

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel (312) 437-9680
TWX 910-222-1834

INDIANA

†Arrow Electronics, Inc
2718 Rand Road
Indianapolis 46241
(317) 243-9353
TWX 810-341-3119

†Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel (317) 844-9333
TWX 810-260-3966

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel (317) 849-7300
TWX 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel (913) 888-8900
TWX 910-743-0005

MARYLAND

Arrow Electronics, Inc
8300 Guilford Road #H
Rivers Center
Columbia 21046
Tel (301) 995-0003
TWX 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel (301) 995-3500
TWX 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel (301) 948-4350
TWX 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel (301) 948-0710
TWX 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc
1 Arrow Drive
Woburn 01801
Tel (617) 933-8130
TWX 710-393-6770

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel (617) 935-9700
TWX 710-393-0382

†Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel (617) 863-1200
TWX 710-326-6617

MICHIGAN

†Arrow Electronics, Inc
3810 Varsity Drive
Ann Arbor 48104
Tel (313) 971-8220
TWX 810-223-6020

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel (313) 525-1800
TWX 810-242-3271

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel (313) 522-4700
TWX 810-242-8775

†Hamilton/Avnet Electronics
2215 29th Street S E
Space A5
Grand Rapids 49508
Tel (616) 243-8805
TWX 810-273-6921

MINNESOTA

†Arrow Electronics, Inc
5230 W 73rd Street
Edina 55435
Tel (612) 830-1800
TWX 910-576-3125

†Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel (612) 932-0600
TWX (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel (612) 935-5444
TWX 910-576-2738

MISSOURI

†Arrow Electronics, Inc
2380 Schuetz
St Louis 63141
Tel (314) 567-6888
TWX 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel (314) 344-1200
TWX 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc
1 Perimeter Road
Manchester 03103
Tel (603) 668-6968
TWX 710-220-1684

NEW JERSEY

†Arrow Electronics, Inc
6000 Lincoln East
Marlton 08053
Tel (215) 928-1800
TWX 710-897-0829

†Arrow Electronics, Inc
2 Industrial Road
Fairfield 07006
Tel (201) 575-5300
TWX 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg 36
Cherry Hill 08003
Tel (609) 424-0110
TWX 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel (201) 575-3390
TWX 710-734-4388

†Microcomputer System Technical Demonstrator Centers

DOMESTIC DISTRIBUTORS

NEW JERSEY (Cont'd)

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel (201) 575-3510
TWX 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel (201) 227-5552

NEW MEXICO

†Alliance Electronics Inc
11030 Cochiti S E
Albuquerque 87123
Tel (505) 292-3360
TWX 910-989-1151

†Hamilton/Avnet Electronics
2524 Baylor Drive S E
Albuquerque 87106
Tel (505) 765-1500
TWX 910-989-0614

NEW YORK

†Arrow Electronics, Inc
25 Hub Drive
Melville 11735
Tel (516) 694-6800
TWX 510-224-6126

†Arrow Electronics, Inc
3000 South Winton Road
Rochester 14623
Tel (716) 275-0300
TWX 510-253-4766

†Arrow Electronics, Inc
7705 Maltage Drive
Liverpool 13088
Tel (315) 652-1000
TWX 710-545-0230

†Arrow Electronics, Inc
20 Oser Avenue
Hauppauge 11788
Tel (516) 231-1000
TWX 510-227-6623

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel (716) 475-9130
TWX 510-253-5470

†Hamilton/Avnet Electronics
16 Corporate Circle
E Syracuse 13057
Tel (315) 437-2641
TWX 710-541-1560

†Hamilton/Avnet Electronics
5 Hub Drive
Melville, Long Island 11747
Tel (516) 454-6000
TWX 510-224-6166

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel (607) 748-8211
TWX 510-252-0893

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel (516) 921-8700
TWX 510-221-2184

†Pioneer Northeast Electronics
840 Fairport Park 14450
Tel (716) 381-7070
TWX 510-253-7001

†MTI Systems Sales
38 Harbor Park Drive
P O Box 271
Port Washington 11050
Tel (516) 621-6200
TWX 510-223-0846

NORTH CAROLINA

†Arrow Electronics, Inc
5240 Greendairy Road
Raleigh 27604
Tel (919) 876-3132
TWX 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel (919) 878-0819
TWX 510-928-1836

†Pioneer Electronics
9801 A-Southern Pine Boulevard
Charlotte 28210
Tel (704) 524-8188
TWX 810-621-0366

OHIO

†Arrow Electronics, Inc
7620 McEwen Road
Centerville 45459
Tel (513) 435-5563
TWX 810-459-1611

†Arrow Electronics, Inc
6238 Cochran Road
Solon 44139
Tel (216) 248-3990
TWX 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel (513) 433-0610
TWX 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel (216) 831-3500
TWX 810-427-9452

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel (513) 236-9900
TWX 810-459-1622

†Pioneer Electronics
4800 E 131st Street
Cleveland 44105
Tel (216) 587-3600
TWX 810-422-2211

OKLAHOMA

†Arrow Electronics, Inc
4719 S Memorial Drive
Tulsa 74145
Tel (918) 665-7700

OREGON

†Almac Electronics Corporation
8022 S W Nimbus, Bldg 7
Beaverton 97005
Tel (503) 641-9070
TWX 910-467-8743

†Hamilton/Avnet Electronics
6024 S W Jean Road
Bldg C, Suite 10
Lake Oswego 97034
Tel (503) 635-7848
TWX 910-455-8179

PENNSYLVANIA

†Arrow Electronics, Inc
650 Seco Road
Monroeville 15146
Tel (412) 856-7000

†Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel (412) 782-2300
TWX 710-795-3122

PENNSYLVANIA (Cont'd)

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel (215) 674-4000
TWX 510-665-6778

TEXAS

†Arrow Electronics, Inc
3220 Commander Drive
Carrollton 75006
Tel (214) 380-6464
TWX 910-860-5377

†Arrow Electronics, Inc
10899 Kinghurst
Suite 100
Houston 77099
Tel (713) 530-4700
TWX 910-880-4439

†Arrow Electronics, Inc
2227 W Braker Lane
Austin 78758
Tel (512) 835-4180
TWX 910-874-1348

†Hamilton/Avnet Electronics
2401 Rutland
Austin 78757
Tel (512) 837-8911
TWX 910-874-1319

†Hamilton/Avnet Electronics
2111 W Walnut Hill Lane
Irving 75062
Tel (214) 659-4100
TWX 910-860-5929

†Hamilton/Avnet Electronics
8750 West Park
Hosuton 77063
Tel (713) 780-1771
TWX 910-881-5523

†Pioneer Electronics
9901 Burnet Road
Austin 78758
Tel (512) 835-4000
TWX 910-874-1323

†Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel (214) 386-7300
TWX 910-850-5563

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel (713) 988-5555
TWX 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel (801) 972-2800
TWX 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel (801) 974-9953

WASHINGTON

†Almac Electronics Corporation
14360 S E Eastgate Way
Bellevue 98007
Tel (206) 643-9992
TWX 910-444-2067

†Arrow Electronics, Inc
14320 N E 21st Street
Bellevue 98007
Tel (206) 643-4800
TWX 910-444-2017

†Hamilton/Avnet Electronics
14212 N E 21st Street
Bellevue 98005
Tel (206) 453-5874
TWX 910-443-2469

WISCONSIN

†Arrow Electronics, Inc
430 W Rausson Avenue
Oakcreek 53154
Tel (414) 764-6600
TWX 910-262-1193

†Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel (414) 784-4510
TWX 910-262-1182

CANADA

ALBERTA

†Hamilton/Avnet Electronics
2816 21st Street N E
Calgary T2E 6Z2
Tel (403) 230-3586
TWX 03-827-642

Zentronics
Bay No 1
3300 14th Avenue N E
Calgary T2A 6J4
Tel (403) 272-1021

BRITISH COLUMBIA

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel (604) 273-5575
TWX 04-5077-89

MANITOBA

Zentronics
590 Berry Street
Winnipeg R3H 0S1
Tel (204) 775-8661

ONTARIO

Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel (416) 677-7432
TWX 610-492-8867

Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel (613) 226-1700
TWX 05-349-71

Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel (416) 451-9600
TWX 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel (519) 884-5700

Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel (613) 225-8840
TWX 06-976-78

QUEBEC

Hamilton/Avnet Electronics
2670 Sabourin Street
St Laurent H4S 1M2
Tel (514) 331-6443
TWX 610-421-3731

Zentronics
505 Locke Street
St Laurent H4T 1X7
Tel (514) 735-5361
TWX 05-827-535

†Microcomputer System Technical Demonstrator Centers

EUROPEAN SALES OFFICES

BELGIUM

Intel Corporation S A
Parc Seny
Rue du Moulin a Papier 51
Borte 1
B-1160 Brussels
Tel (02)661 07 11
TELEX 24814

DENMARK

Intel Denmark A/S*
Glentevej 61 - 3rd Floor
DK-2400 Copenhagen
Tel (01) 19 80 33
TELEX 19567

FINLAND

Intel Finland OY
Hameentie 103
SF - 00550 Helsinki 55
Tel 0/716 955
TELEX 123 332

FRANCE

Intel Corporation, S A R L *
5 Place de la Balance
Silic 223
94528 Rungis Cedex
Tel (01) 687 22 21
TELEX 270475

FRANCE (Cont'd)

Intel Corporation, S A R L
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel (7) 842 40 89
TELEX 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Munchen 2
Tel (89) 53891
TELEX 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzer Strasse 75
D-6200 Wiesbaden 1
Tel (6121) 70 08 74
TELEX 04186183 INTW D

Intel Semiconductor GmbH
Brueckstrasse 61
7012 Fellbach
Stuttgart
Tel (711) 58 00 82
TELEX 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollern Strasse 5*
3000 Hannover 1
Tel (511) 34 40 81
TELEX 923625 INTH D

ISRAEL

Intel Semiconductor Ltd *
P O Box 1659
Hafa
Tel 4/524 261
TELEX 46511

ITALY

Intel Corporation Italia Spa*
Milanofiori, Palazzo E
20094 Assago (Milano)
Tel (02) 824 00 06
TELEX 315183 INTMIL

NETHERLANDS

Intel Semiconductor Nederland B V *
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel (10) 21 23 77
TELEX 22283

NORWAY

Intel Norway A/S
P O Box 92
Hvamveien 4
N-2013
Skjetten
Tel (2) 742 420
TELEX 18018

SPAIN

Intel Iberia
Calle Zurbaran 28
Madrid 04
Tel (34) 1410 40 04
TELEX 46880

SWEDEN

Intel Sweden A B *
Dalvagen 24
S-17136 Solna
Tel (08) 734 01 00
TELEX 12261

SWITZERLAND

Intel Semiconductor A G *
Talackerstrasse 17
8152 Glattbrugg postfach
CH-8065 Zurich
Tel (01) 829 29 77
TELEX 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U K) Ltd *
5 Hospital Street
Nantwich, Cheshire CW5 5RE
Tel (0270) 626 560
TELEX 36620

Intel Corporation (U K) Ltd *
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel (0793) 488 388
TELEX 444447 INT SWN

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotemuehlgasse 26
A 1120 Vienna
Tel (222) 83 56 46
TELEX 11532 BASAT A

BELGIUM

Inelco Belgium S A
Ave des Croix de Guerre 94
B1120 Brussels
Tel (02) 216 01 60
TELEX 25441

DENMARK

ITT MultiKomponent A/S
Naverland 29
DK-2600 Glostrup
Tel (02) 45 66 45
TX 33355

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel (0) 692 60 22
TELEX 124 224 Ftron SF

FRANCE

Generim
Z I de Courtaboeuf
Avenue de la Baltique
91943 Les Ulis Cedex-B P 88
Tel (1) 907 78 78
TELEX F691700

Jermyn S A
16, Avenue Jean-Jaures
94600 Choisy-Le-Roi
Tel (1) 853 12 00
TELEX 260967

Metrologie
La Tour d' Asnieres
4, Avenue Laurent Cely
92606-Asnieres
Tel (1) 790 62 40
TELEX 611-448

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet B P 2
92310 Sevres
Tel (1) 534 75 35
TELEX 204552

WEST GERMANY

Electronic 2000 Vertriebs A G
Stahlgruberring 12
D-8000 Munich 82
Tel (89) 42 00 10
TELEX 522561 EIEC D

Jermyn GmbH
Postfach 1180
Schulstrasse 84
D-6277 Bad Camberg
Tel (06434) 231
TELEX 484426 JERM D

CES Computer Electronics Systems GmbH
Gutenbergstrasse 4
2359 Henstedt-Ulzburg
Tel (04193) 4026
TELEX 2180260

Metrologie GmbH
Hansastrasse 15
8000 Munich 21
Tel (89) 57 30 84
TELEX D 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich bei Frankfurt
Tel (6103) 33564
TELEX 417983

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co Dublin
Tel (1) 85 62 88
TELEX 31584

ISRAEL

Eastronics Ltd
11 Rozanis Street
P O Box 39300
Tel Aviv 61390
Tel (3) 47 51 51
TELEX 33638

ITALY

Eledra 3S S P A
Viale Elvezia, 18
I 20154 Milano
Tel (2) 34 97 51
TELEX 332332

ITALY (Cont'd)

Intesi
Milanofiori Pal E/5
20090 Assago
Milano
Tel (02) 82470
TELEX 311351

NETHERLANDS

Koning & Hartman
Koperwerf 30
P O Box 43220
2544 EN's Gravenhage
Tel 31 (70) 210 101
TELEX 31528

NORWAY

Nordisk Elektronic (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel (2) 846 210
TELEX 17546

PORTUGAL

Ditram
Componentes E Electronica LDA
Av Miguel Bombarda, 133
P1000 Lisboa
Tel (19) 545 313
TELEX 14182 Brieks-P

SPAIN

Interface S A
Av Pompeu Fabra 12
08024 Barcelona
Tel (3) 219 80 11
TELEX 51508

ITT SESA
Miguel Angel 21, 6 Piso
Madrid 10
Tel (34) 14 1954 00
TELEX 27461

SWEDEN

AB Gosta Backstrom
Box 12009
Alstroemergatan 22
S-10221 Stockholm 12
Tel (8) 541 080
TELEX 10135

Nordisk Electronik AB
Box 27301
Sandhamnsgatan 71
S-10254 Stockholm
Tel (8) 635 040
TELEX 10547

SWEDEN (Cont'd)

Telko AB
Gardsfogdevagen 1
Box 186
S-161 26 Bromma
Tel (8) 98 08 20
TELEX 11941

SWITZERLAND

Industrade AG
Hertistrasse 31
CH-8304 Wallisellen
Tel (01) 830 50 40
TELEX 56788 INDEL CH

UNITED KINGDOM

Bytech Ltd
Unit 57
London Road
Earley, Reading
Berkshire
Tel (0734) 61031
TELEX 848215

Comway Microsystems Ltd
Market Street
UK-Bracknell, Berkshire
Tel 44 (344) 55333
TELEX 847201

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel (0732) 450144
TELEX 95142

M E D L
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel (190) 49307
TELEX 28817

Rapid Recall, Ltd
Rapid House/Denmark St
High Wycombe
Berks, England HP11 2ER
Tel (0494) 26 271
TELEX 837931

YUGOSLAVIA

H R Microelectronics Enterprises
P O Box 5604
San Jose, California 95150
Tel 408/978-8000
TELEX 278-559

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty Ltd *
(Mailing Address)
P O Box 571
North Sydney NSW, 2065

(Shipping Address)
Spectrum Building
200 Pacific Highway
Level 6
Crows Nest, NSW, 2089
Tel 011-61-2-957-2744
TELEX 790-20097
FAX 011-61-2-923-2632

HONG KONG

Intel Semiconductor Ltd *
1701-3/1720 Connaught Centre
1 Connaught Road
Tel 011-852-5-215311
TWX 60410 ITLHK

JAPAN

Intel Japan K K
5-6 Tokodai, Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel 029747-8511
TELEX 03656-160

Intel Japan K K *
2-1-15 Naka-machi
Atsugi, Kanagawa 243
Tel 0462-23-3511

Intel Japan K K *
2-51-2 Kojima-cho
Chofu, Tokyo 182
Tel 0424-88-3151

Intel Japan K K *
2-69 Hon-cho
Kumagaya, Saitama 360
Tel 0485-24-6871

JAPAN (Cont'd)

Intel Japan K K *
2-4-1 Terauchi
Toyonaka, Osaka 560
Tel 06-863-1091

Intel Japan K K
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel 03-201-3621

Intel Japan K K *
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel 03-426-2231

Intel Japan K K *
Mitsui-Seimei Musashi-Kosugi Bldg
915 Shinmaruko, Nakahara-ku
Kawasaki-Shi, Kanagawa 211
Tel 044-733-7011

JAPAN (Cont'd)

Intel Japan K K
1-1 Shibahon-cho
Mishima-shi
Shizuoka-Ken 411
Tel 0559-72-4121

SINGAPORE

Intel Semiconductor Ltd
101 Thomson Road
21-06 Goldhill Square
Singapore 1130
Tel 011-65-2507811
TWX RS 39921
CABLE INTELSGP

*Field Application Location

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

VLC S R L
Sarmiento 1630, 1 Piso
1042 Buenos Aires
Tel 011-54-1-35-1201/9242
TELEX 17575 EDARG

Agent
Soimex International Corporation
15 Park Row, Room #1730
New York, New York 10038
Tel (212) 406-3052
Attn Gaston Briones

AUSTRALIA

Total Electronics
(Mailing Address)
Private Bag 250
Burwood, Victoria 3125

(Shipping Address)
9 Harker Street
Burwood
Victoria 3125
Tel 011-61-3-288-4044
TELEX AA 31261

Total Electronics
P O Box 139
Artarmon, N S W 2064
Tel 011-61-02-438-1855
TELEX 26297

BRAZIL

Icotron S A
05110 Av Mutinga 3650-6 Andar
Pirituba Sao Paulo
Tel 011-55-11-833-2572
TELEX 1122274 ICOTBR

CHILE

DIN
(Mailing Address)
Av VIC, MacKenna 204
Casilla 6055
Santiago
Tel 011-56-2-277-564
TELEX 352-0003

(Shipping Address)
A102 Greenville Center
3801 Kennett Pike
Wilmington, Delaware 19807

HONG KONG

Novel Precision Machinery Co , Ltd
Flat D 20 Kingsford Ind Bldg
Phase 1 26 Kwai Hei Street NT
Tel 011-852-5-0-223222
TWX 39114 JINMI HX

Schmidt & Co Ltd
18/F Great Eagle Centre
Wanchai
Tel 011-852-5-833-0222
TWX 74766 SCHMC HK

INDIA

Micronic Devices
65 ARUN Complex
D V G Road
Basavan Gudi
Bangalore 560004
Tel 011-91-812-600-631
TELEX 011-5947 MDEV

Micronic Devices
104/109C Nirmal Industrial Estate
Sion (E)
Bombay 400022
Tel 011-91-22-48-61-70
TELEX 011-71447 MDEV IN

Micronic Devices
R-694 New Rajinder Nagar
New Delhi 110060

Ramlak International, Inc (Agent)
465 S Mathilda Avenue
Suite 302
Sunnyvale, CA 94086
Tel (408) 733-8767

S & S Corporation
(Mailing Address)
P O Box 1185
Mauldin, South Carolina 29657

(Shipping Address)
308 Green Drive
Liberty, South Carolina 29657

JAPAN

Asahi Electronics Co Ltd
KMM Bldg Room 407
2-14-1 Asano, Kokurakita-Ku
Kitakyushu City 802
Tel (093) 511-6471
TELEX AECKY 7126-16

JAPAN (Cont'd)

Hamilton-Avnet Electronics Japan Ltd
YU and YOU Bldg 1-5-7 Horidome-Cho
Nihonbashi Chuo-Ku, Tokyo 103
Tel (03) 662-9911
TELEX 2523774

Ryoyo Electric Corporation
Konwa Bldg
1-12-22, Tsukiji
Chuo-Ku, Tokyo 104
Tel (03) 543-7711/541-7311

Tokyo Electron Ltd
Shinjuku Nomura Bldg
26-2 Nishi-Shinjuku 1-Chome
Shinjuku-Ku, Tokyo 160
Tel (03) 343-4411
TELEX 232-2220 LABTEL J

KOREA

J-TEK Corporation
2nd Floor, Government Pension Bldg
24-3, Yoido-Dong
Youngdungpo-Ku
Seoul 150
Tel 011-82-2-782-8039
TELEX KODIGIT K25299

Koram Digital USA (Agent)
14066 East Firestone Boulevard
Santa Fe Springs, CA 90670
Tel (714) 739-2204
TWX 194715 KORAM DIGIT LSA

NEW ZEALAND

McLean Information Technology Ltd
459 Kyber Pass Road, Newmarket,
P O Box 9464, Newmarket
Auckland 1, New Zealand
Tel 011-64-9-501-219, 501-801, 587-037
TELEX NZ21570 THERMAL

PAKISTAN

Computer Applications Ltd
7D Gizri Boulevard
Defense
Karachi-46
Tel 011-92-21-530-306/7
TELEX 24434 GAFAR PK

PAKISTAN (Cont'd)

Horizon Training Co , Inc (Agent)
1 Lafayette Center
1120 20th Street N W
Suite 530
Washington, D C 20036
Tel (202) 887-1900
TWX 248890 HORN

SINGAPORE

General Engineers Corporation Pty Ltd
Units 1003-1008 Block 3
10th Floor PSA Multi Storey Complex
Telok Blangahi Pasir
Pan Jang
Singapore 5
Tel 011-65-271-3163
TELEX RS23987 GENERCO
CABLE GENERRCORP

SOUTH AFRICA

Electronic Building Elements, Pty Ltd
P O Box 4609
Pretoria 0001
Tel 011-27-12-46-9221
TELEX 3-22786 SA
TELEGRAM ELBILEM

TAIWAN

Mitac Corporation
3rd Floor #75, Section 4
Nanking East Road
Taipei
Tel 011-886-2-771-0940, 0941
TELEX 11942 TAIAUTO

Mectel International, Inc (Agent)
3385 Viso Court
Santa Clara, CA 95050
Tel (408) 988-4513
TWX 910-338-2201
FAX 408-980-9742

YUGOSLAVIA

H R Microelectronics Enterprises
P O Box 5604
San Jose, California 95150
Tel (408) 978-8000
TELEX 278-559

*Field Application Location

DOMESTIC SERVICE OFFICES

CALIFORNIA

Intel Corp
1350 Shorebird Way
Mt View 94043
Tel (415) 968-8211
TWX 910-339-9279
910-338-0255

Intel Corp
2000 E 4th Street
Suite 110
Santa Ana 92705
Tel (714) 835-5577
TWX 910-595-2475

Intel Corp
4350 Executive Drive
Suite 150
San Diego 92121
Tel (619) 452-5880

Intel Corp
5530 N Corbin Avenue
Suite 120
Tarzana 91356
Tel (213) 708-0333

COLORADO

Intel Corp
650 South Cherry
Suite 720
Denver 80222
Tel (303) 321-8086
TWX 910-931-2289

CONNECTICUT

Intel Corp
26 Mill Plain Road
Danbury 06811
Tel (203) 748-3130

FLORIDA

Intel Corp
1500 N W 62nd Street
Suite 104
Ft Lauderdale 33309
Tel (305) 771-0600
TWX 510-956-9407

FLORIDA (Cont'd)

Intel Corp.
500 N Maitland Avenue
Suite 205
Maitland 32751
Tel (305) 628-2393
TWX 810-853-9219

GEORGIA

Intel Corp
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel (404) 441-1171

ILLINOIS

Intel Corp
2550 Golf Road
Suite 815
Rolling Meadows 60008
Tel (312) 981-7295
TWX 910-253-1825

KANSAS

Intel Corp
9400 W 110th Street
Suite 170
Overland Park 66210
Tel (913) 642-8080

MARYLAND

Intel Corp
5th Floor Product Service
7833 Walker Drive
Greenbelt 20770
Tel (301) 441-1020

MASSACHUSETTS

Intel Corp
27 Industrial Avenue
Chelmsford 01824
Tel (617) 256-1800
TWX 710-343-6333

MICHIGAN

Intel Corp
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel (313) 851-8905

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

NEW JERSEY

Intel Corp
385 Sylvan Avenue
Englewood Cliffs 07632
Tel (201) 567-0820
TWX 710-991-8593

Intel Corp
Raritan Plaza III
Raritan Center
Edison 08817
Tel (201) 225-3000

NORTH CAROLINA

Intel Corp
2306 W Meadowview Road
Suite 206
Greensboro 27407
Tel (919) 294-1541

OHIO

Intel Corp
Chagrin-Brainard Bldg
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel (216) 464-6915
TWX 810-427-9298

Intel Corp
6500 Poe
Dayton 45414
Tel (513) 890-5350

OREGON

Intel Corp
10700 S W Beaverton-Hillsdale
Highway
Suite 22
Beaverton 97005
Tel (503) 641-8086
TWX 910-467-8741

Intel Corp
5200 N E Elam Young Parkway
Hillsboro 97123
Tel (503) 681-8080

PENNSYLVANIA

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel (313) 354-1540

TEXAS

Intel Corp
313 E Anderson Lane
Suite 314
Austin 78752
Tel: (512)454-3628
TWX 910-874-1347

Intel Corp
12300 Ford Road
Suite 380
Dallas 75234
Tel (214) 241-8087
TWX 910-860-5617

WASHINGTON

Intel Corp
110 110th Avenue N E
Suite 510
Bellevue 98004
Tel 1-800-525-5560
TWX 910-443-3002

WISCONSIN

Intel Corp
450 N Sunnyslope Road
Suite 130
Brookfield 53005
Tel (414) 784-8087

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Japan

FRANCE
Intel
5 Place de la Balance
Silic 223
94528 Rungis Cedex
France

UNITED KINGDOM
Intel
Piper's Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel
Seidstrasse 27
D-8000 Munchen 2
West Germany

Printed in U.S.A./C-790/1284/115K/RRD LD-PM
Microprocessors/Peripherals

ISBN 0-917017-22-6