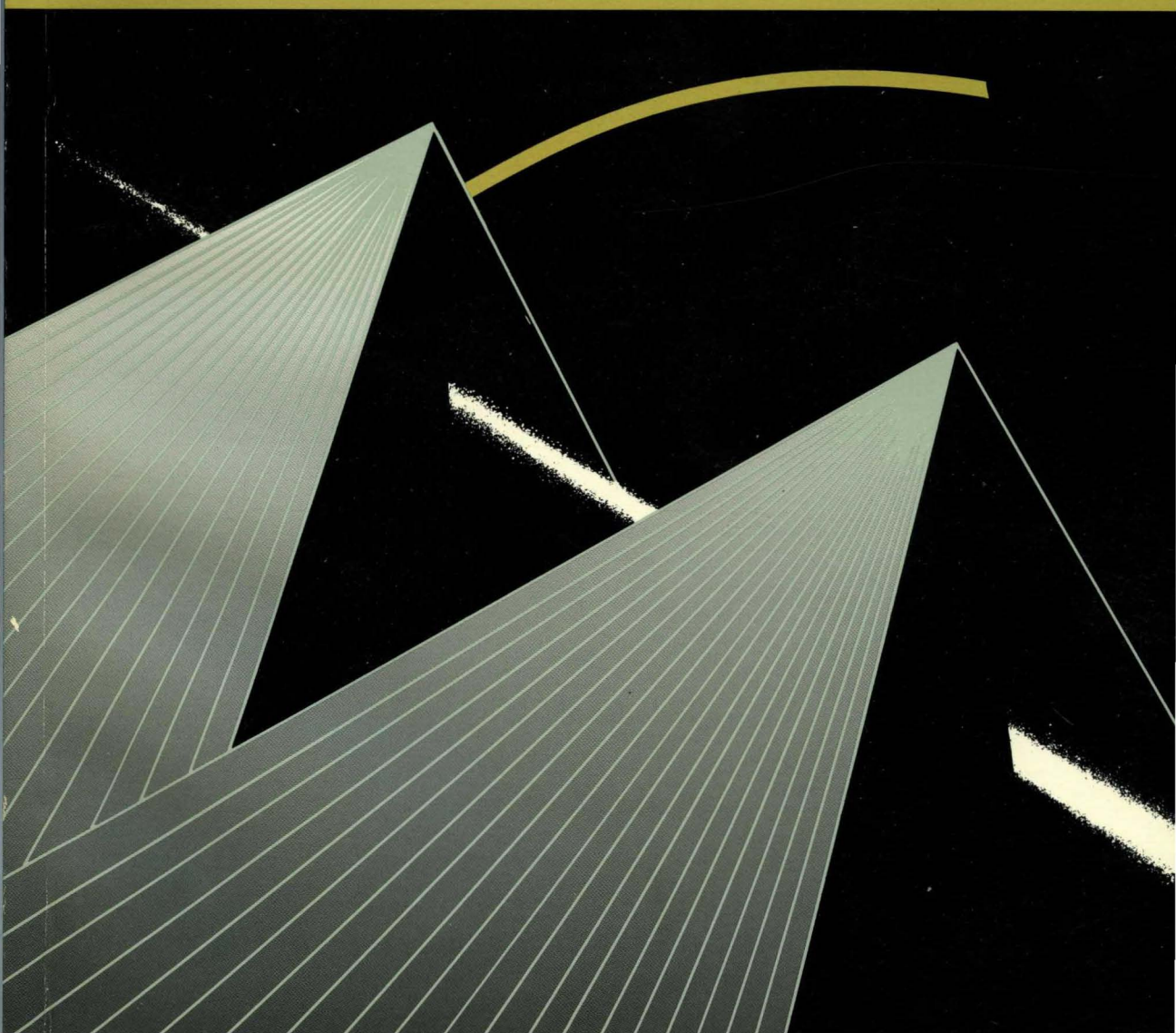
intel

i486™ MICROPROCESSOR

i486™



i486™ MICROPROCESSOR

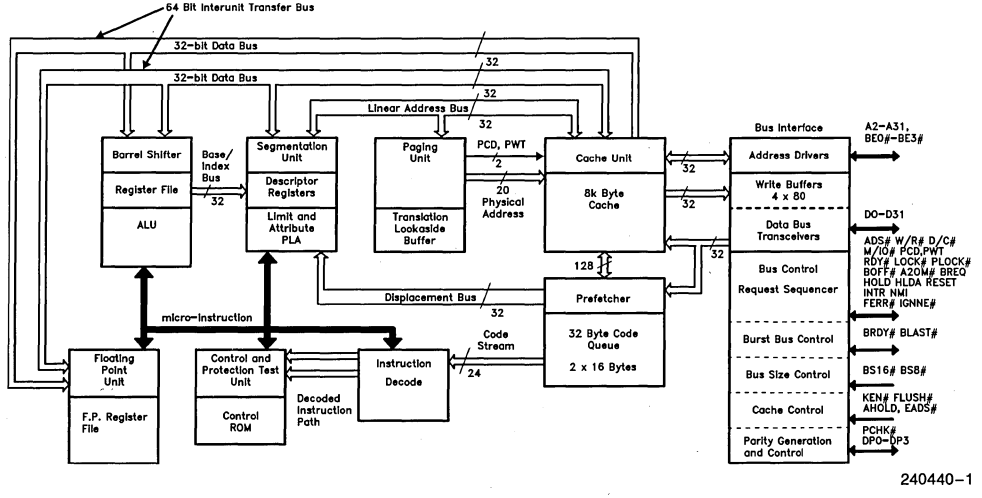
- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows**
  - **— UNIX*** System V/386**
  - **— iRMX®, iRMK™ Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built-In Self Test**
  - **— Hardware Debugging Support**
  - **— Intel Software Support**
  - **— Extensive Third Party Software Support**
- **High Performance Design**
  - **— Frequent Instructions Execute in One Clock**
  - **— 25 MHz and 33 MHz Clock Frequencies**
  - **— 106 Mbyte/Sec Burst Bus**
  - **— CHMOS IV Process Technology**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
- **Multiprocessor Support**
  - **— Multiprocessor Instructions**
  - **— Cache Consistency Protocols**
  - **— Support for Second Level Cache**

The i486™ CPU offers the highest performance for DOS, OS/2, Windows and UNIX System V/386 applications. It is 100% binary compatible with the 386™ CPU. One million transistors integrate cache memory, floating point hardware and memory management on-chip while retaining binary compatibility with previous members of the 86 architectural family. Frequently used instructions execute in one cycle resulting in RISC performance levels. An 8 Kbyte unified code and data cache combined with a 106 Mbyte/Sec burst bus at 33.3 MHz ensure high system throughput even with inexpensive DRAMs.

New features enhance multiprocessing systems. New instructions speed manipulation of memory based semaphores. On-chip hardware ensures cache consistency and provides hooks for multilevel caches.

The built in self test extensively tests on-chip logic, cache memory and the on-chip paging translation cache. Debug features include breakpoint traps on code execution and data accesses.

i486™ Microprocessor Pipelined 32-Bit Microarchitecture



240440-1

iRMX, iRMK, 386, 387, 486, i486 are trademarks of Intel Corporation.
*MS-DOS® is a registered trademark of Microsoft Corporation.
**OS/2™ is a trademark of Microsoft Corporation.
***UNIX™ is a trademark of AT&T.

April 1989

Order Number: 240440-001

CONTENTS

PAGE

CONTENTS (Continued)

CONTENTS (Continued) PAGE

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

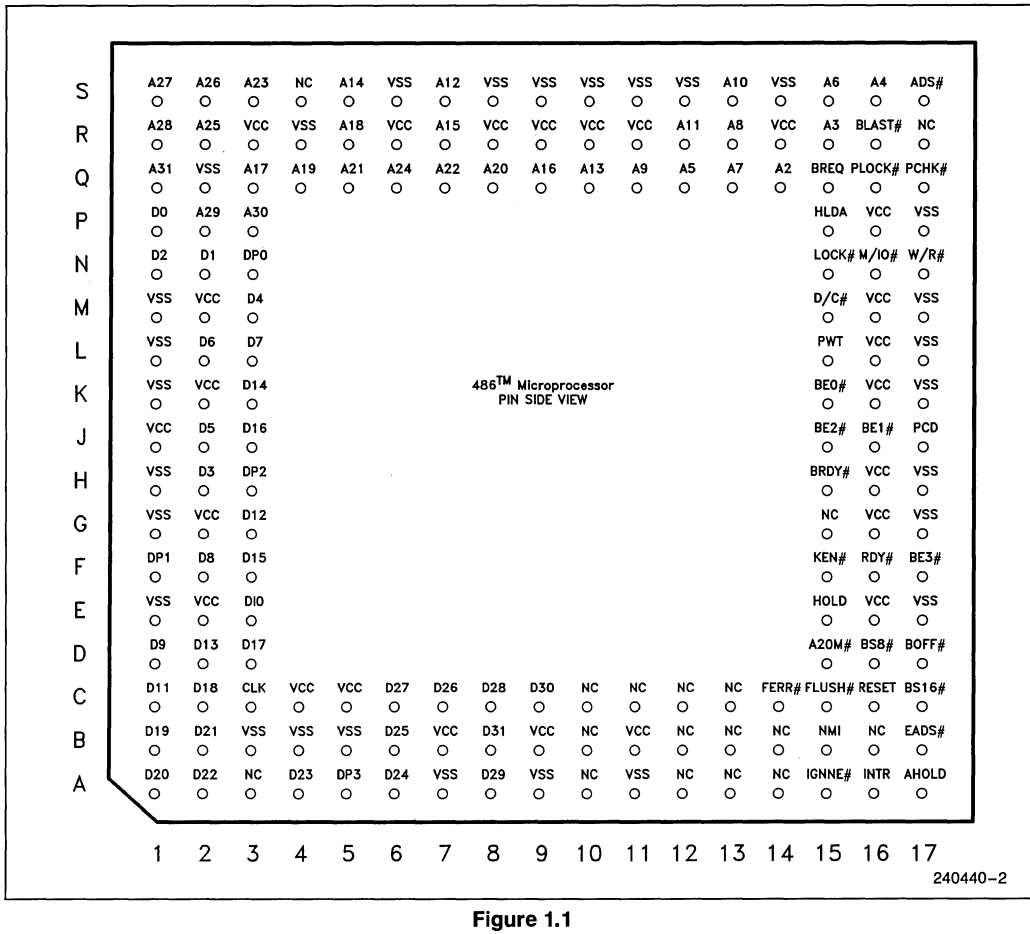

S A27 A26 A23 NC A14 VSS A12 VSS VSS VSS VSS VSS A10 VSS A6 A4 ADS#
R A28 A25 VCC VSS A18 VCC A15 VCC VCC VCC VCC A11 A8 VCC A3 BLAST# NC
Q A31 VSS A17 A19 A21 A24 A22 A20 A16 A13 A9 A5 A7 A2 BREQ PLOCK# PCHK#
P D0 A29 A30 HLDA VCC VSS
N D2 D1 DP0 LOCK# M/IO# W/R#
M VSS VCC D4 D/C# VCC VSS
L VSS D6 D7 PWT VCC VSS
K VSS VCC D14 BE0# VCC VSS
J VCC D5 D16 BE2# BE1# PCD
H VSS D3 DP2 BRDY# VCC VSS
G VSS VCC D12 NC VCC VSS
F DP1 D8 D15 KEN# RDY# BE3#
E VSS VCC DI0 HOLD VCC VSS
D D9 D13 D17 A20M# BS8# BOFF#
C D11 D18 CLK VCC VCC D27 D26 D28 D30 NC NC NC NC FERR# FLUSH# RESET BS16#
B D19 D21 VSS VSS VSS D25 VCC D31 VCC NC VCC NC NC NC NMI NC EADS#
A D20 D22 NC D23 DP3 D24 VSS D29 VSS NC VSS NC NC NC IGNNE# INTR AHOLD
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
486™ Microprocessor
PIN SIDE VIEW
240440-2

Figure 1.1

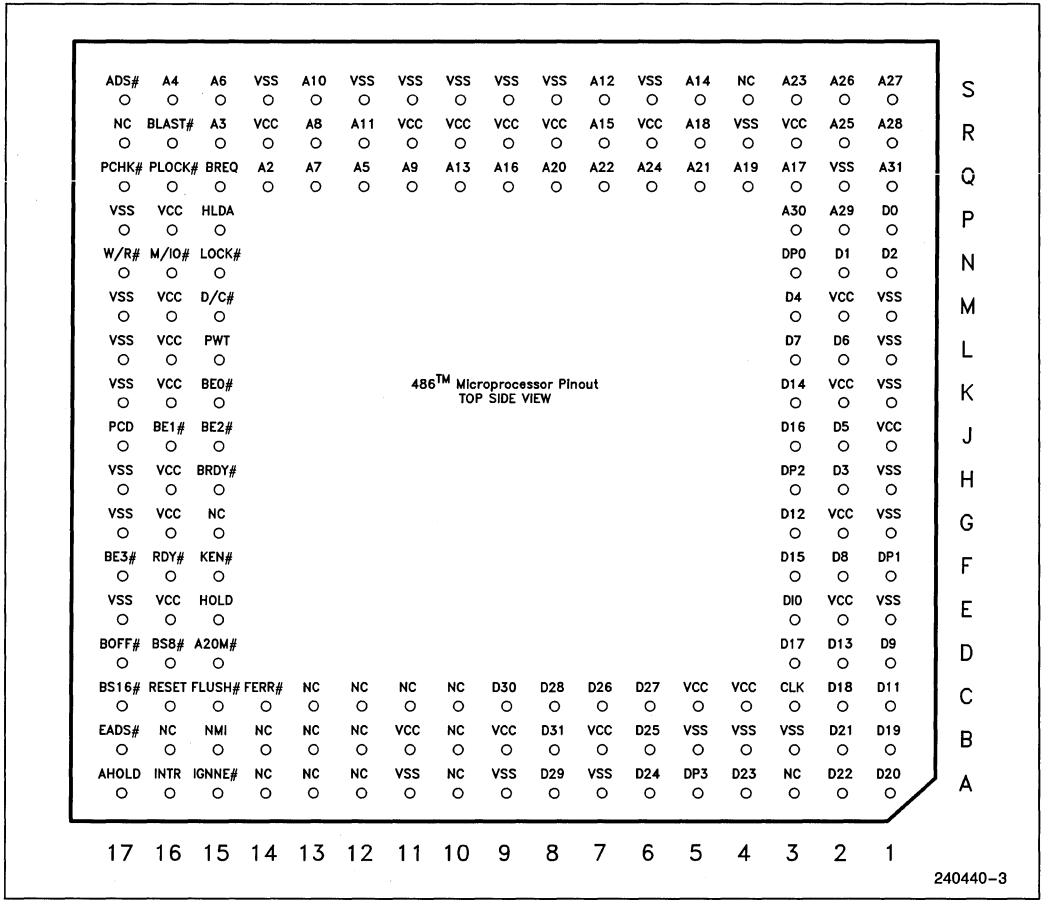

ADS# A4 A6 VSS A10 VSS VSS VSS VSS VSS A12 VSS A14 NC A23 A26 A27 S
NC BLAST# A3 VCC A8 A11 VCC VCC VCC VCC A15 VCC A18 VSS VCC A25 A28 R
PCHK# PLOCK# BREQ A2 A7 A5 A9 A13 A16 A20 A22 A24 A21 A19 A17 VSS A31 Q
VSS VCC HLDA A30 A29 D0 P
W/R# M/IO# LOCK# DP0 D1 D2 N
VSS VCC D/C# D4 VCC VSS M
VSS VCC PWT D7 D6 VSS L
VSS VCC BE0# D14 VCC VSS K
PCD BE1# BE2# D16 D5 VCC J
VSS VCC BRDY# DP2 D3 VSS H
VSS VCC NC D12 VCC VSS G
BE3# RDY# KEN# D15 D8 DP1 F
VSS VCC HOLD DI0 VCC VSS E
BOFF# BS8# A20M# D17 D13 D9 D
BS16# RESET FLUSH# FERR# NC NC NC NC D30 D28 D26 D27 VCC VCC CLK D18 D11 C
EADS# NC NMI NC NC NC VCC NC VCC D31 VCC D25 VSS VSS VSS D21 D19 B
AHOLD INTR IGNNE# NC NC NC VSS NC VSS D29 VSS D24 DP3 D23 NC D22 D20 A
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
486™ Microprocessor Pinout
TOP SIDE VIEW
240440-3

Figure 1.2

Pin Cross Reference by Pin Name

Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name	Location
A2	Q14	BS16#	C17	HLDA	P15	V_{SS}	B4
A3	R15	CLK	C3	HOLD	E15	V_{SS}	B5
A4	S16	D0	P1	IGNNE#	A15	V_{SS}	E1
A5	Q12	D1	N2	INTR	A16	V_{SS}	E17
A6	S15	D2	N1	KEN#	F15	V_{SS}	G1
A7	Q13	D3	H2	LOCK#	N15	V_{SS}	G17
A8	R13	D4	M3	M/IO#	N16	V_{SS}	H1
A9	Q11	D5	J2	NMI	B15	V_{SS}	H17
A10	S13	D6	L2	PCD	J17	V_{SS}	K1
A11	R12	D7	L3	PCHK#	Q17	V_{SS}	K17
A12	S7	D8	F2	PWT	L15	V_{SS}	L1
A13	Q10	D9	D1	PLOCK#	Q16	V_{SS}	M1
A14	S5	D10	E3	RDY#	F16	V_{SS}	M17
A15	R7	D11	C1	RESET	C16	V_{SS}	P17
A16	Q9	D12	G3	V_{CC}	B7	V_{SS}	Q2
A17	Q3	D13	D2	V_{CC}	B9	V_{SS}	R4
A18	R5	D14	K3	V_{CC}	B11	V_{SS}	S6
A19	Q4	D15	F3	V_{CC}	C4	V_{SS}	S8
A20	Q8	D16	J3	V_{CC}	C5	V_{SS}	S9
A21	Q5	D17	D3	V_{CC}	E2	V_{SS}	S10
A22	Q7	D18	C2	V_{CC}	E16	V_{SS}	S11
A23	S3	D19	B1	V_{CC}	G2	V_{SS}	S12
A24	Q6	D20	A1	V_{CC}	G16	V_{SS}	S14
A25	R2	D21	B2	V_{CC}	H16	W/R#	N17
A26	S2	D22	A2	V_{CC}	J1		
A27	S1	D23	A4	V_{CC}	K2	NC	A3
A28	R1	D24	A6	V_{CC}	K16	NC	A10
A29	P2	D25	B6	V_{CC}	L16	NC	A12
A30	P3	D26	C7	V_{CC}	M2	NC	A13
A31	Q1	D27	C6	V_{CC}	M16	NC	A14
A20M#	D15	D28	C8	V_{CC}	P16	NC	B10
ADS#	S17	D29	A8	V_{CC}	R3	NC	B12
AHOLD	A17	D30	C9	V_{CC}	R6	NC	B13
BE0#	K15	D31	B8	V_{CC}	R8	NC	B14
BE1#	J16	D/C#	M15	V_{CC}	R9	NC	B16
BE2#	J15	DP0	N3	V_{CC}	R10	NC	C10
BE3#	F17	DP1	F1	V_{CC}	R11	NC	C11
BLAST#	R16	DP2	H3	V_{CC}	R14	NC	C12
BOFF#	D17	DP3	A5	V_{SS}	A7	NC	C13
BRDY#	H15	EADS#	B17	V_{SS}	A9	NC	G15
BREQ	Q15	FERR#	C14	V_{SS}	A11	NC	R17
BS8#	D16	FLUSH#	C15	V_{SS}	B3	NC	S4

NOTE:
Pins identified as NC should remain completely unconnected.

Pin Cross Reference By Location

Location	Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name
A1	D20	C9	D30	J15	BE2#	Q10	A13
A2	D22	C10	NC	J16	BE1#	Q11	A9
A3	NC	C11	NC	J17	PCD	Q12	A5
A4	D23	C12	NC	K1	V_{SS}	Q13	A7
A5	DP3	C13	NC	K2	V_{CC}	Q14	A2
A6	D24	C14	FERR#	K3	D14	Q15	BREQ
A7	V_{SS}	C15	FLUSH#	K15	BE0#	Q16	PLOCK#
A8	D29	C16	RESET	K16	V_{CC}	Q17	PCHK#
A9	V_{SS}	C17	BS16#	K17	V_{SS}	R1	A28
A10	NC	D1	D9	L1	V_{SS}	R2	A25
A11	V_{SS}	D2	D13	L2	D6	R3	V_{CC}
A12	NC	D3	D17	L3	D7	R4	V_{SS}
A13	NC	D15	A20M#	L15	PWT	R5	A18
A14	NC	D16	BS8#	L16	V_{CC}	R6	V_{CC}
A15	IGNNE#	D17	BOFF#	L17	V_{SS}	R7	A15
A16	INTR	E1	V_{SS}	M1	V_{SS}	R8	V_{CC}
A17	AHOLD	E2	V_{CC}	M2	V_{CC}	R9	V_{CC}
B1	D19	E3	D10	M3	D4	R10	V_{CC}
B2	D21	E15	HOLD	M15	D/C#	R11	V_{CC}
B3	V_{SS}	E16	V_{CC}	M16	V_{CC}	R12	A11
B4	V_{SS}	E17	V_{SS}	M17	V_{SS}	R13	A8
B5	V_{SS}	F1	DP1	N1	D2	R14	V_{CC}
B6	D25	F2	D8	N2	D1	R15	A3
B7	V_{CC}	F3	D15	N3	DP0	R16	BLAST#
B8	D31	F15	KEN#	N15	LOCK#	R17	NC
B9	V_{CC}	F16	RDY#	N16	M/IO#	S1	A27
B10	NC	F17	BE3#	N17	W/R#	S2	A26
B11	V_{CC}	G1	V_{SS}	P1	D0	S3	A23
B12	NC	G2	V_{CC}	P2	A29	S4	NC
B13	NC	G3	D12	P3	A30	S5	A14
B14	NC	G15	NC	P15	HLDA	S6	V_{SS}
B15	NMI	G16	V_{CC}	P16	V_{CC}	S7	A12
B16	NC	G17	V_{SS}	P17	V_{SS}	S8	V_{SS}
B17	EADS#	H1	V_{SS}	Q1	A31	S9	V_{SS}
C1	D11	H2	D3	Q2	V_{SS}	S10	V_{SS}
C2	D18	H3	DP2	Q3	A17	S11	V_{SS}
C3	CLK	H15	BRDY#	Q4	A19	S12	V_{SS}
C4	V_{CC}	H16	V_{CC}	Q5	A21	S13	A10
C5	V_{CC}	H17	V_{SS}	Q6	A24	S14	V_{SS}
C6	D27	J1	V_{CC}	Q7	A22	S15	A6
C7	D26	J2	D5	Q8	A20	S16	A4
C8	D28	J3	D16	Q9	A16	S17	ADS#

NOTE:
All pins identified as NC should remain completely unconnected.

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the microprocessor. A31–A2 together with the byte enables, BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are active HIGH and are not driven during bus or address hold.
BE3# BE2# BE1# BE0#	O O O O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the 486 microprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are active HIGH and are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION (Continued)		
LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The 486 microprocessor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The 486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the 486 microprocessor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
INTERRUPTS		
RESET	I	The *reset* input forces the 486 microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *internal cycle pending* signal indicates that the 486 microprocessor has internally generated a bus request. BREQ is generated whether or not the 486 microprocessor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the 486 microprocessor bus. In response to HOLD going active the 486 microprocessor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The 486 microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the 486 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.
BOFF#	I	The *backoff* input forces the 486 microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the 486 microprocessor's address bus for a cache invalidation cycle. The 486 microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
CACHE INVALIDATION (Continued)		
EADS	I	This signal indicates that a *valid external address* has been driven onto the 486 microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the 486 microprocessor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before ready during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache enable bit (CE) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the 387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# is active LOW, and is not floated during bus hold.
IGNNE#	I	When the *ignore numeric error* pin is asserted the 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions. When IGNNE# is deasserted the 486 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the 486 microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the 486 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.
ADDRESS MASK		
A20M#	I	When the *address bit 20 mask* pin is asserted, the 486 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

Table 1.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#–BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2–A3	HIGH	Bus, Address Hold

Table 1.2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous

Table 1.3. Input/Output Pins

Name	Active Level	When Floated
D0–D31	HIGH	Bus Hold
DP0–DP3	HIGH	Bus Hold
A4–A31	HIGH	Bus, Address Hold

2.0 ARCHITECTURAL OVERVIEW

The 486 microprocessor is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The 486 microprocessor contains all the features of the 386™ microprocessor with enhancements to increase performance. The instruction set includes the complete 386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the 386 microprocessor MMU. The 486 microprocessor brings the 387™ math coprocessor on-chip. All software written for the 386 microprocessor, 387 math coprocessor and previous members of the 86/87 architectural family will run on the 486 microprocessor without any modifications.

Several enhancements have been added to the 486 microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features combined, lead to performance greater than twice that of a 386 microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, the 486 microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on a 486 microprocessor can have a maximum of 16,381 segments each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 486 microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the 486 microprocessor operates as a very fast 8086. Real Mode is required primarily to setup the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the 486 microprocessor has features to facilitate high performance hardware designs. The 1X clock eases high frequency board level designs. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The 486 microprocessor register set includes all the registers contained in the 386 microprocessor and the 387 math coprocessor. The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers

Floating Point Registers
- Data Registers
- Tag Word
- Status Word
- Instruction and Data Pointers
- Control Word

Debug and Test Registers

The base architecture and floating point registers are accessible by the applications program. The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the 486 microprocessor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

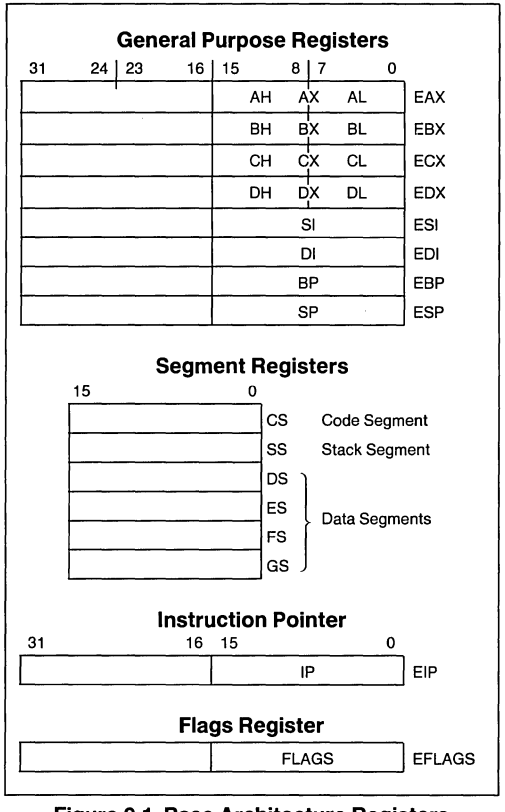


Figure 2.1. Base Architecture Registers

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the 486 microprocessor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

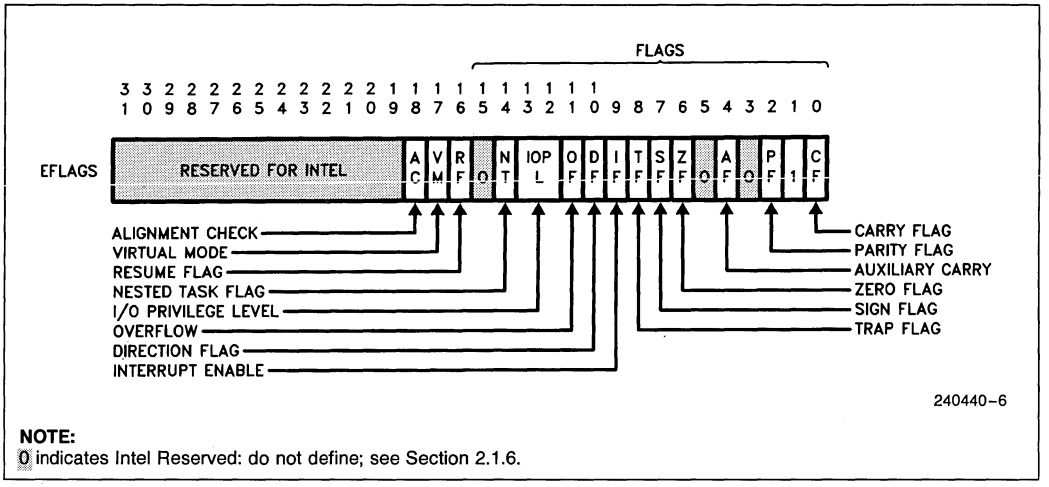
Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the 486 microprocessor. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.



NOTE:
0 indicates Intel Reserved: do not define; see Section 2.1.6.

Figure 2.2. Flags Register

EFLAGS bits 1, 3, 5, 15 and 19–31 are "undefined". When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the 486 microprocessor contains a new bit not available in the 386 microprocessor. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A mis-aligned address is a word access to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the 486 microprocessor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit "Pseudo-Descriptor"	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:
Several instructions on the 486 microprocessor generate misaligned references, even if their memory address is aligned. For example, on the 486 microprocessor the LGDT/LIDT (load global/interrupt descriptor table) and SGDT/SIDT (store global/interrupt descriptor table) instructions read/write two bytes, and then read/write four bytes from a "pseudo-descriptor" at the given address. The 486 microprocessor will generate misaligned references unless the address is on a dword boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for ½ of the register save/restore cycles. The 486 microprocessor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 486 Microprocessor is in Protected Mode, the 486 Microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 486 Microprocessor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:
In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

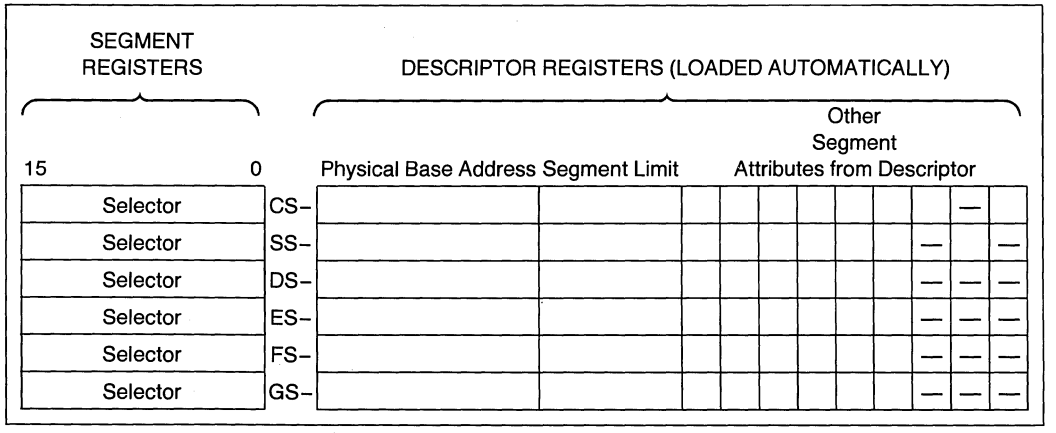


Figure 2.3. i486™ Microprocessor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point unit (FPU) and the segmentation and paging mechanisms. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (GDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the 486 microprocessor's CR0 are not contained in the 386 microprocessor's CR0. The new bits are CE, WT, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

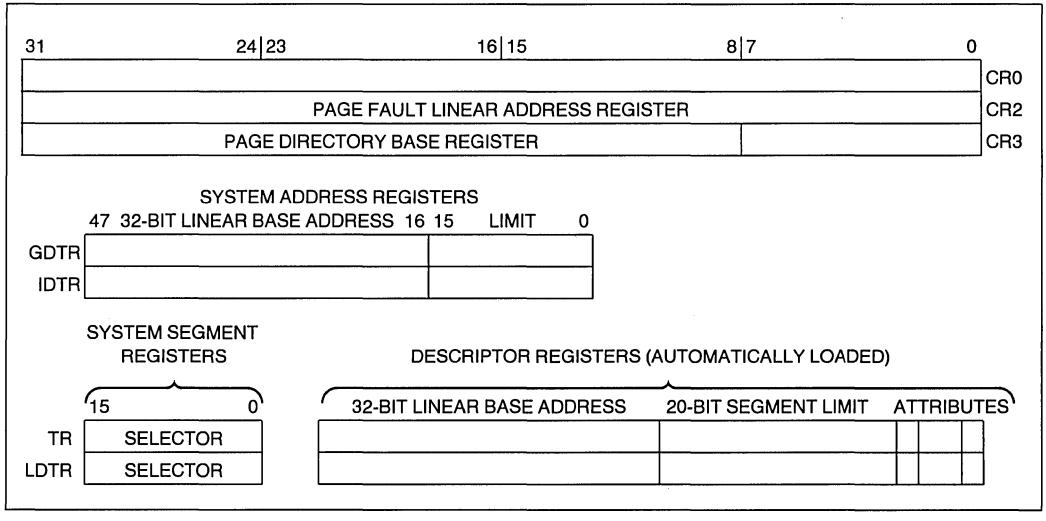


Figure 2.4. System Level Registers

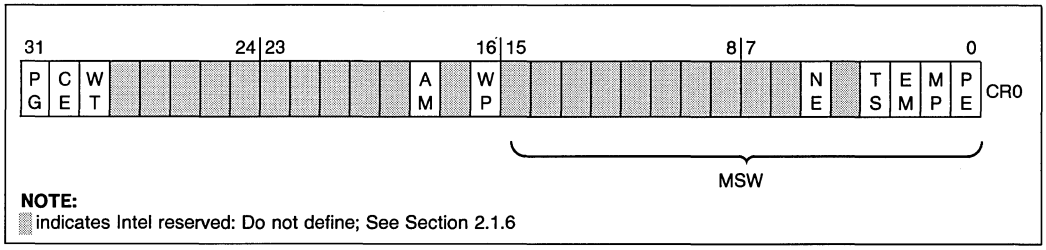


Figure 2.5. Control Register 0

486 Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CE, WT (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CE	WT	Operating Mode
0	0	Cache fills disabled, write-through and invalidates disabled.
0	1	Cache fills disabled, write-through and invalidates enabled.
1	0	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
1	1	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. On-Chip Floating Point Unit Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the 486 microprocessor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New 486 microprocessor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG = 1) or disabled (PG = 0). See Table 2.2.

CE (Cache Enable, bit 30)

The CE bit is used to enable the on-chip cache. When CE = 0, the cache will not be filled on cache misses. When CE = 1, cache fills may be performed on misses. See Table 2.3.

The state of the CE bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CE = 1 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CE is set to zero after RESET for compatibility with the 386 microprocessor.

WT (Writes Transparent, bit 29)

The WT bit enables on-chip cache write-throughs and write-invalidate cycles (WT = 1). When WT = 1, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when WT = 1. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When WT = 0, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With WT = 0 the only write cycles that reach the external bus are cache misses. Write hits with WT = 0 will never update main memory. Invalidate cycles are ignored when WT = 0.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM = 0 disables the AC bit. AM = 1 enables the AC bit. AM = 0 is the 386 microprocessor compatible mode.

386 microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM=0 will prevent AC faults from occurring before the 486 microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The 386 microprocessor allows a read-only page to be written from privilege levels 0–2. The 486 microprocessor is compatible with the 386 microprocessor when WP=0. WP=1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

The NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE=1) or through an external interrupt (NE=0). NE=0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and 387 math coprocessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE=0, the 486 microprocessor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the 486 microprocessor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:

The freeze does not take place if the next instruction is one of the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE=1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

TS (Task Switched, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 will cause a device not available (DNA) fault (trap vector 7). If TS=1 and MP=1 (monitor coprocessor in CR0) a WAIT instruction will cause a DNA fault. See Table 2.4.

EM (Emulate Coprocessor, bit 2)

The EM bit determines whether floating point instructions are trapped (EM=1) or executed. If EM=1, all floating point instructions will cause fault 7.

NOTE:

WAIT instructions are not affected by the state of EM. See Table 2.4.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. If MP=1 and TS=1, WAIT instructions cause fault 7. Refer to Table 2.4. The TS bit is set to 1 on task switches by the 486 microprocessor. Floating point instructions are not affected by the state of the MP bit. It is recommended that the MP bit be set to one for the normal operation of the 486 microprocessor.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism. If PE=1 protection is enabled. When PE=0 the 486 microprocessor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

All new CR0 bits added to the 386 and 486 microprocessors, except for ET and NE, are upwards compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

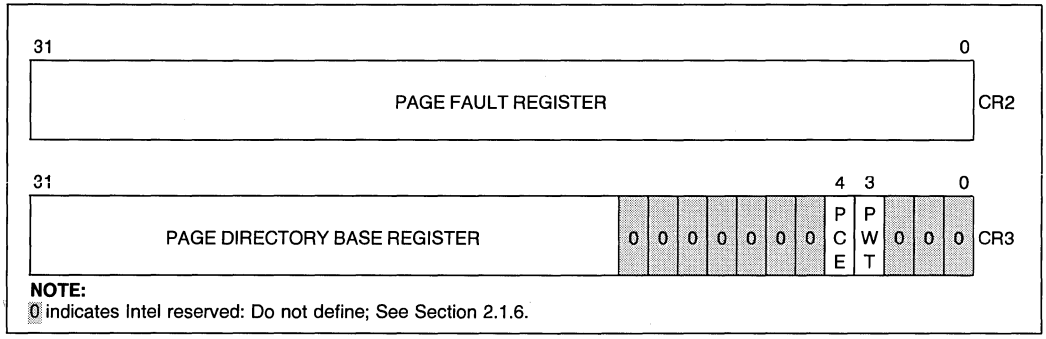


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The 486 microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the 486 microprocessor CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 under two conditions: when paging is disabled (PG = 0 in CR0) or when the PDE is being updated.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB). If the value in CR3 does not change during the task switch, the page table entries in the TLB are not flushed.

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, 386 and 486 microprocessor protection model. These tables or segments are:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:
A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

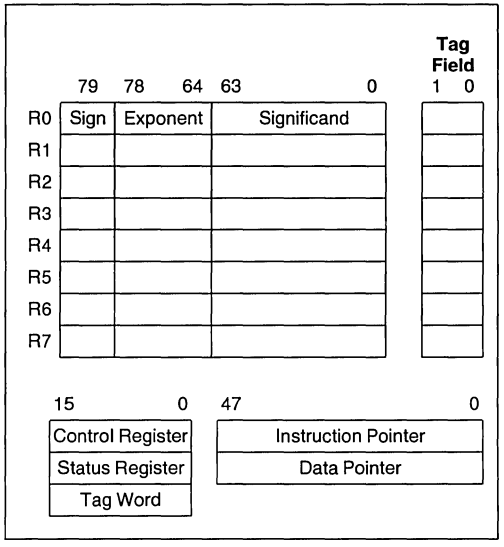


Figure 2.7. Floating Point Registers

The operation of the 486 microprocessors on-chip floating point unit is exactly the same as the 387 math coprocessor. Software written for the 387 math coprocessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the 486 microprocessor's FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either 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. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one 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 one. Like other 486 microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

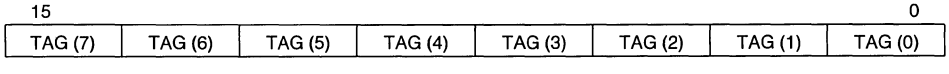
Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPUs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

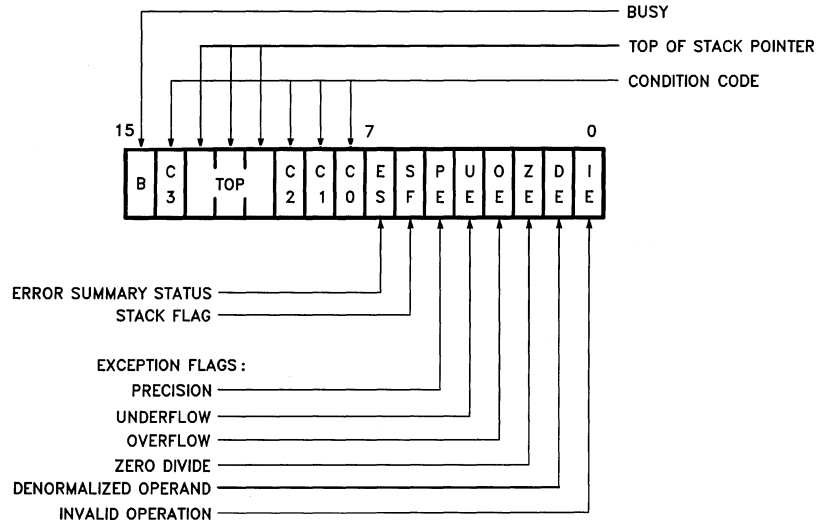
2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.



NOTE:
The index i of tag(i) is **not** top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 2.8. FPU Tag Word



ES is set if any unmasked exception bit is set; cleared otherwise.
See Table 2.5 for interpretation of condition code.
TOP values:
000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack
•
•
•
111 = Register 7 is Top of Stack
For definitions of exceptions, refer to the Section entitled "Exception Handling".

Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.5 through 2.8.

Table 2.5. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2	Q0	Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.7)		Zero or O/U#	Operand is not comparable (Table 2.7)
FXAM	Operand class (see Table 2.8)		Sign or O/U#	Operand class (Table 2.8)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.6. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
0	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.7. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.8. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Table 2.9 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.9 lists the exception conditions, and their causes in order of precedence. Table 2.9 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the 486 microprocessor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the 486 microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the 486 microprocessor the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the 486 microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

Table 2.9. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form (0 * ∞, 0/0, (+∞) + (−∞), etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the 486 microprocessor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 486 microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 486 microprocessor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:
The operand size attribute is the D bit in a segment descriptor.

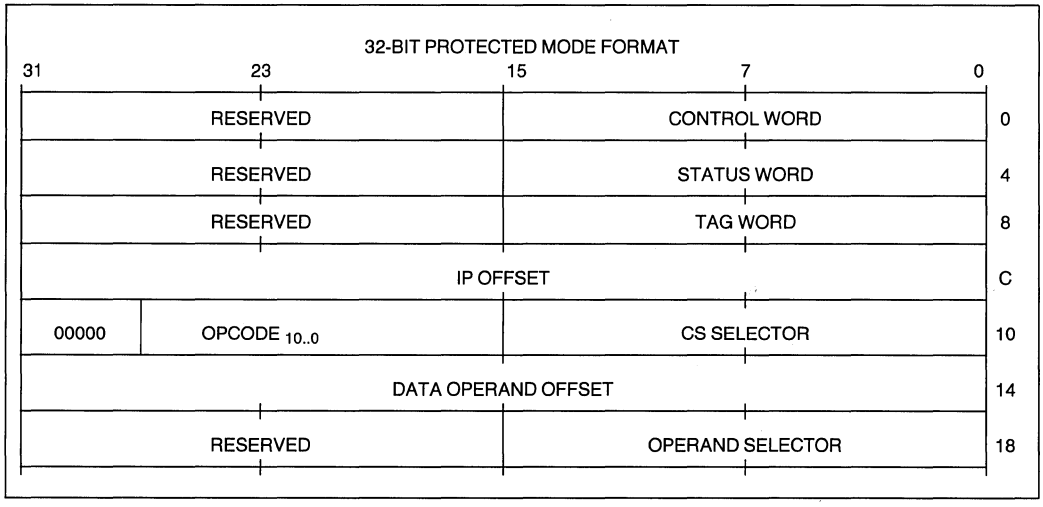


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

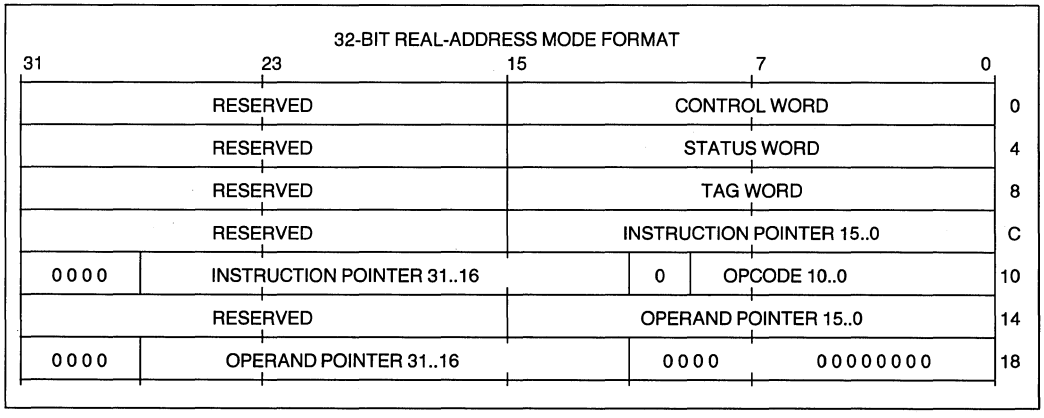


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

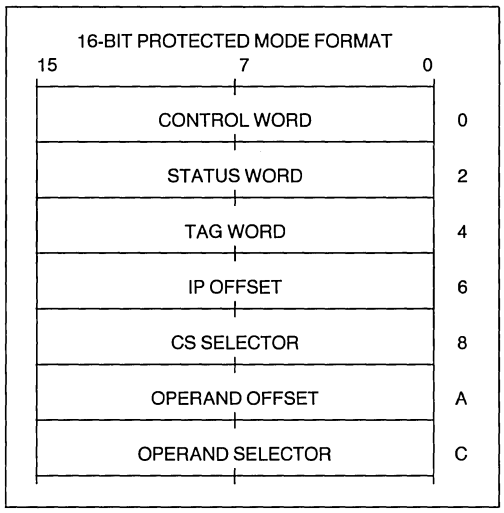


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

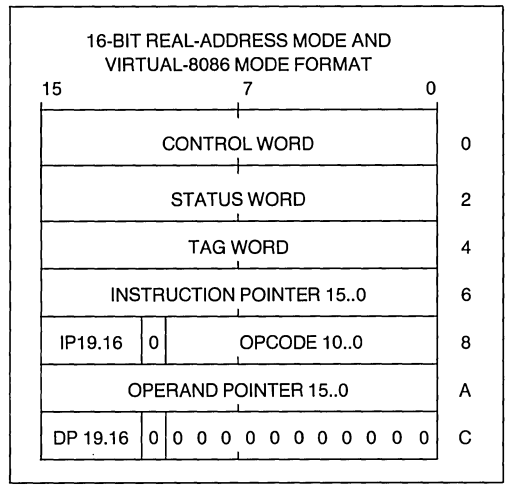


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.

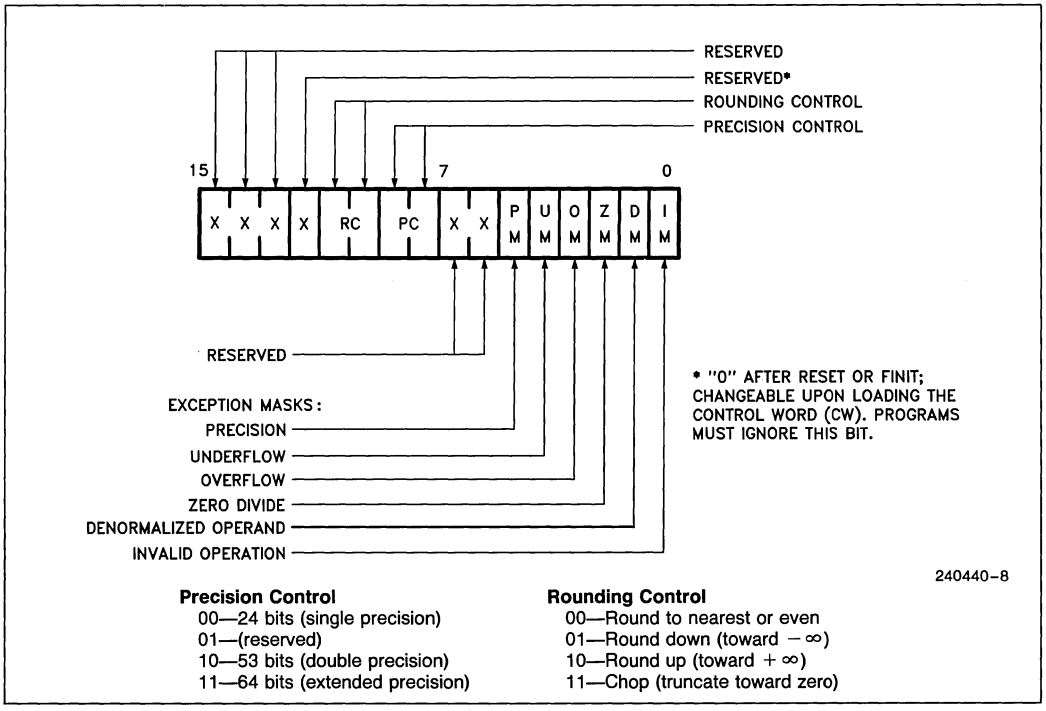


Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers

LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7

Test Registers

CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The 486 microprocessor contains five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.10 summarizes these differences. See Section 4, Protected Mode Architecture for further details.

Table 2.10. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain 486 Microprocessor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**
5) **However, registers which have been previously stored may be reloaded without masking.**

Depending upon the values of undefined register bits will make your software dependent upon the unspecified 486 Microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 486 Microprocessor-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 486 MICROPROCESSOR REGISTER BITS.

2.2 Instruction Set

The 486 microprocessor instruction set can be divided into 11 categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control
Floating Point
Floating Point Control

The 486 microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All 486 microprocessor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 486 microprocessor has a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Memory to Memory
Immediate to Register
Register to Memory
Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 486 or 386 microprocessors (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the 486 Microprocessor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 486 Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 486 Microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The 486 Microprocessor has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the 486 Microprocessor has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

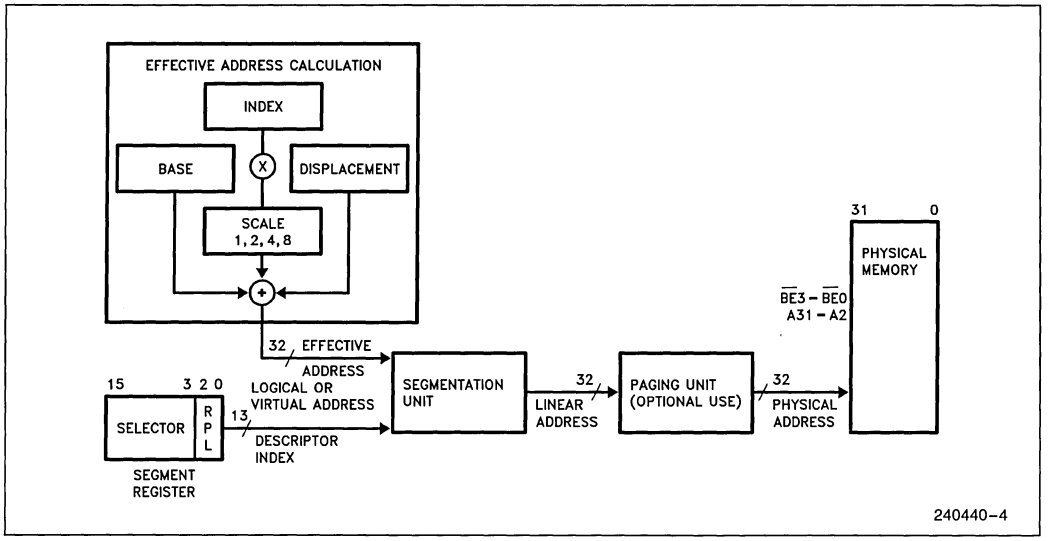


Figure 2.16. Address Translation

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 486 Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.11 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provides the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.11. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The 486 Microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 486 Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

Table 2.11. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The 486 Microprocessor provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index

mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operands offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

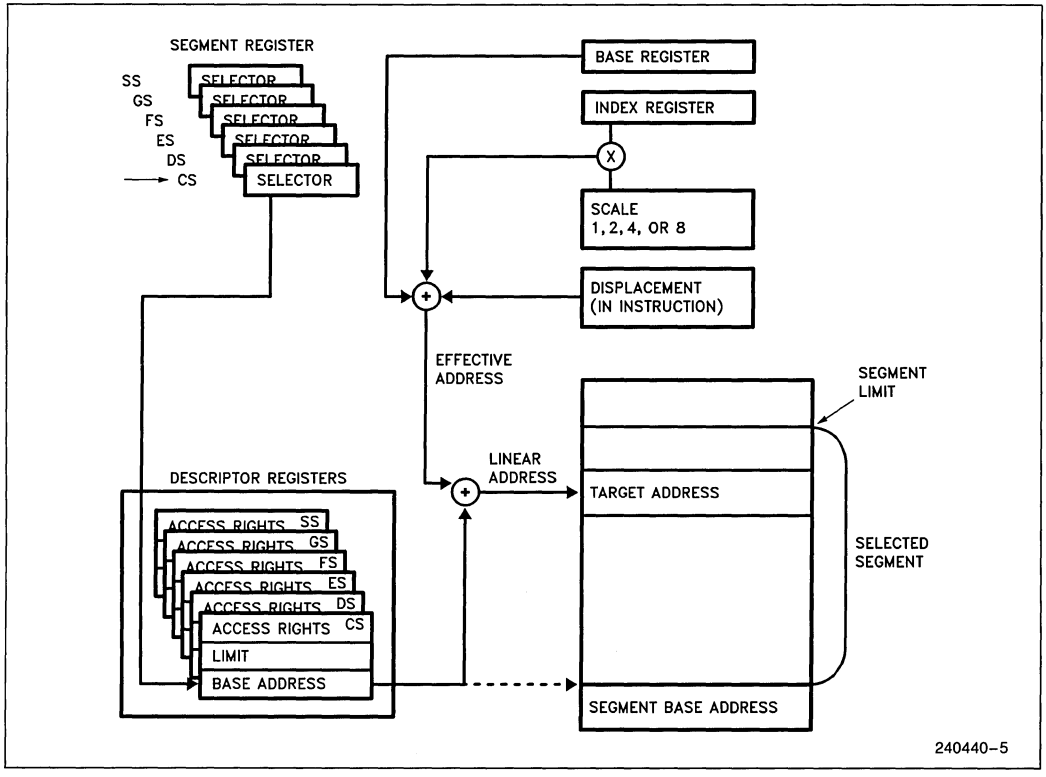


Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP+00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP+80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the 486 Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the 486 Microprocessor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 486 Microprocessor addressing modes.

When executing 32-bit code, the 486 Microprocessor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.12 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The 486 microprocessor can support a wide-variety of data types. In the following descriptions, the on-chip floating point unit (FPU) consists of the floating point registers. The central processing unit (CPU) consists of the base architecture registers.

2.6.1.1 Unsigned Data Types

The FPU does not support unsigned data types. Refer to Table 2.13.

Byte: Unsigned 8-bit quantity

Word: Unsigned 16-bit quantity

Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.12. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.13.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

The FPU only supports 16-, 32- and 64-bit integers. The CPU only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the 486 microprocessor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.13.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The 486 microprocessor supports packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types. Refer to Table 2.13.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.14.

String data types are only supported by the CPU.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

Bit String: A set of contiguous bits. In the 486 microprocessor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

The 486 microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. The CPU can only operate on ASCII data. Refer to Table 2.14.

Table 2.13. i486™ Microprocessor Data Types

Data Format	Supported by Base Registers	Supported by FPU	Range	Precision
Byte	X		0–255	8 bits
Word	X		0K–64K	16 bits
Dword	X		$0\mu-4\mu$	32 bits
8-Bit Integer	X		10^2	8 bits
16-Bit Integer	X	X	10^4	16 bits
32-Bit Integer	X	X	10^9	32 bits
64-Bit Integer		X	10^{19}	64 bits
8-Bit Unpacked BCD	X		0–9	1 Digit
8-Bit Packed BCD	X		0–9	2 Digits
80-Bit Packed BCD		X		18 Digits
Single Precision Real		X	$10^{\pm 38}$	24 Bits
Double Precision Real		X	$10^{\pm 308}$	53 Bits
Extended Precision Real		X	$10^{\pm 4932}$	64 Bits

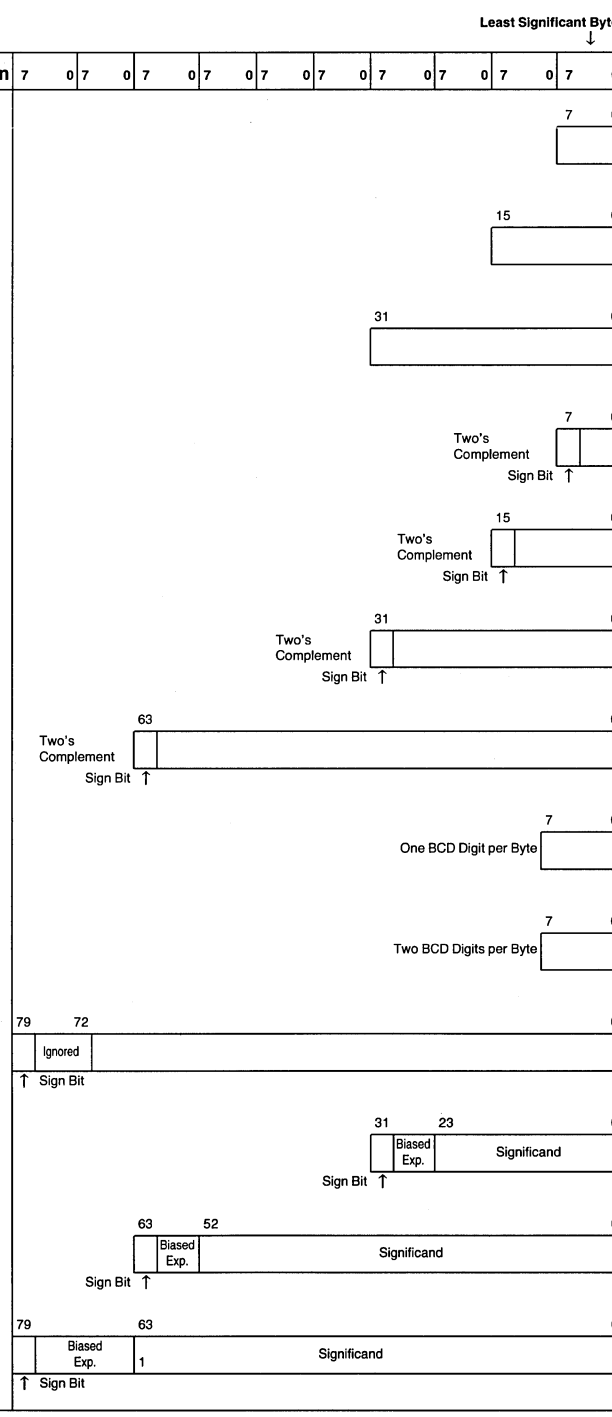
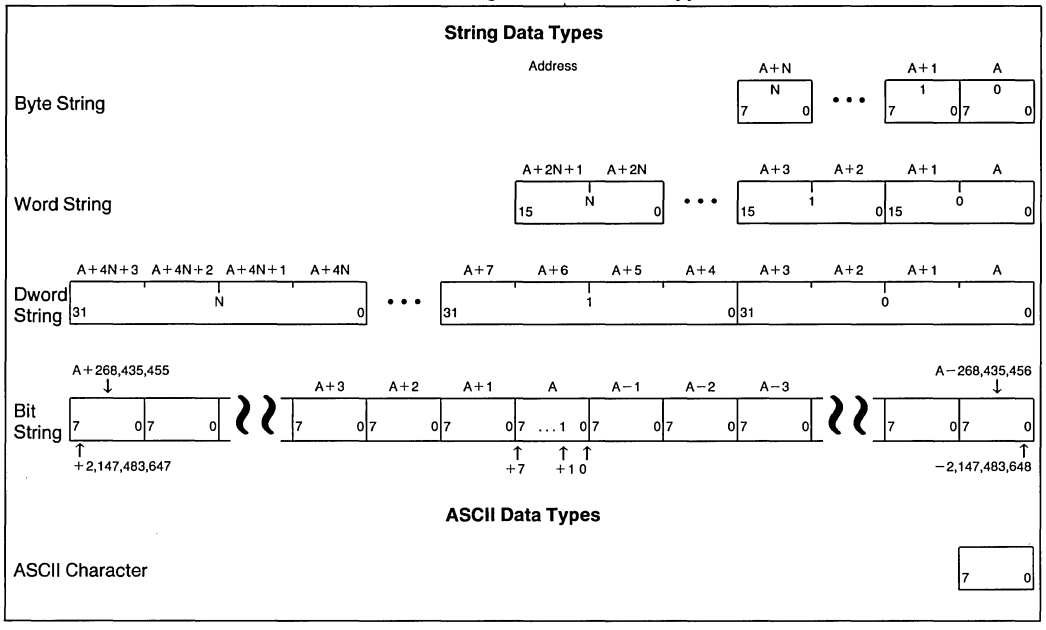


Table 2.14. String and ASCII Data Types



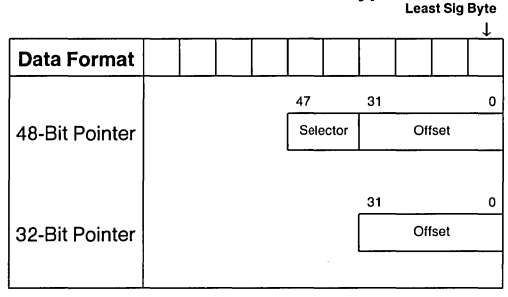
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The 486 microprocessor supports two types of pointers. Refer to Table 2.15.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.15. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The 486 microprocessor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The 486 microprocessor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

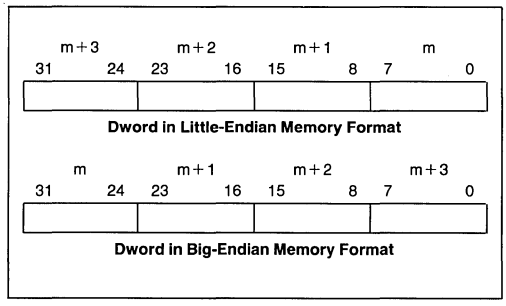


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 486 Microprocessor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.16 summarizes the possible interrupts for the 486 Microprocessor and shows where the return address points.

The 486 Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 486 Microprocessor which identifies the

appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 486 Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the 486 Microprocessor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.16. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18–32			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 486 Microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the 486 Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.0.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 486 Microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 486 Microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. i486™ Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the 486 Microprocessor if, in the course of executing an instruction, the 486 Microprocessor detects a problematic condition. The 486 Microprocessor then immediately invokes the appropriate exception service routine. The state of the 486 Microprocessor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the 486 Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.17b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.17b. Sequence of Exception Checking

Consider the case of the 486 Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
The order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The 486 Microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.17b), the 486 Microprocessor invokes the appropriate exception service routine. The 486 Microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 2.17c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.17c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:
These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

When a Double Fault occurs, the 486 Microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the 486 microprocessor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.18 shows these interrupts and their causes.

Table 2.18. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the 486™ processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The 486™ processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU, FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 486 Microprocessor. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the 486 Microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 486 Microprocessor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

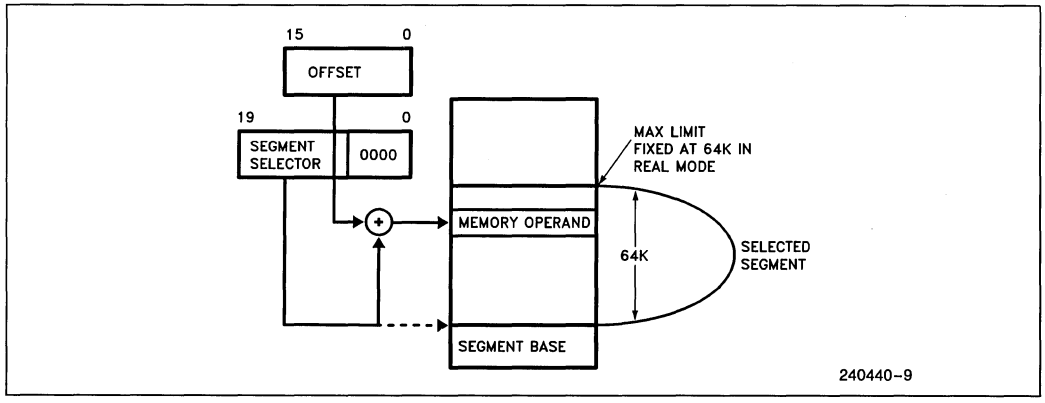


Figure 3.1. Real Address Mode Addressing

The LOCK prefix on the 486 Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 486 Microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The 486 Microprocessor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the 486 Microprocessor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the 486 Microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the 486 Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, the high address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 2.10)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits this implies that Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The 486 Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.16 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 486 Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

Table 3.1

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). Otherwise shutdown can only be exited via the RESET input.

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the 486 Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the 486 Microprocessor to run all of the existing 8086, 80286 and 386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the 486 Microprocessor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 486 Microprocessor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 486 Microprocessor addressing mechanism with paging enabled.

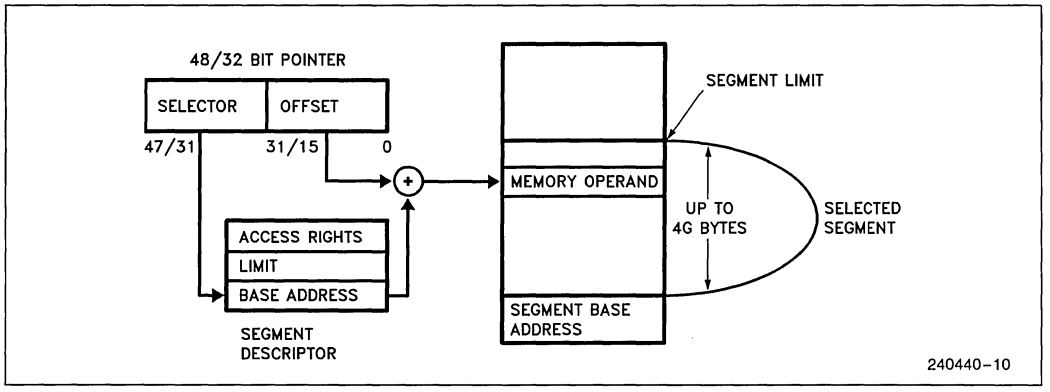


Figure 4.1. Protected Mode Addressing

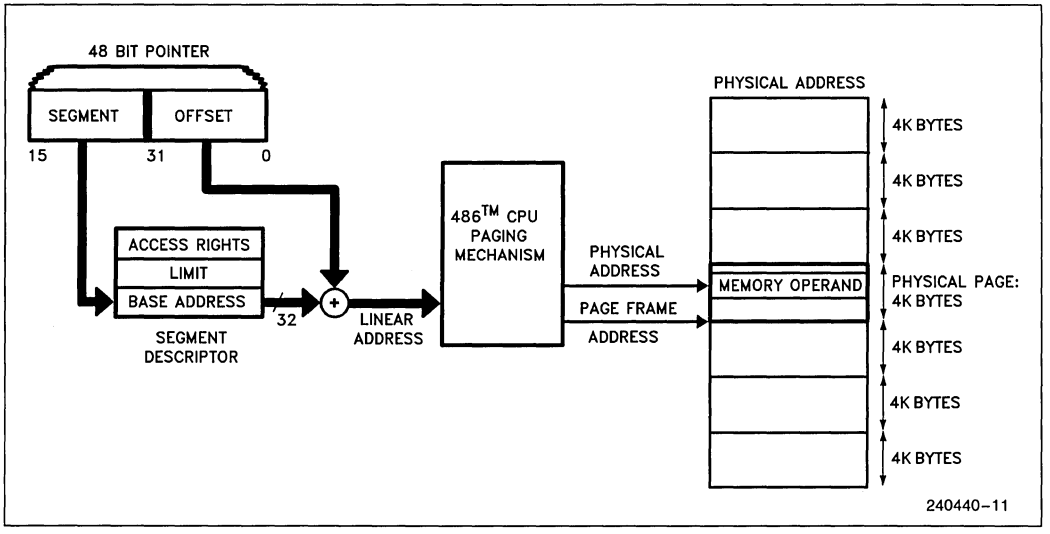


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in an 486 Microprocessor system. There are three types of tables on the 486 Microprocessor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

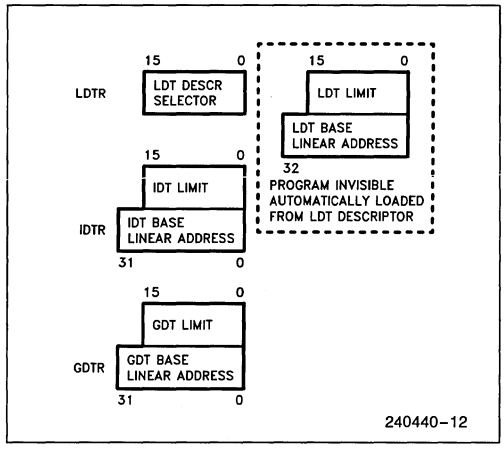


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every 486 Microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This pro-

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for 486 Microprocessor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts).**

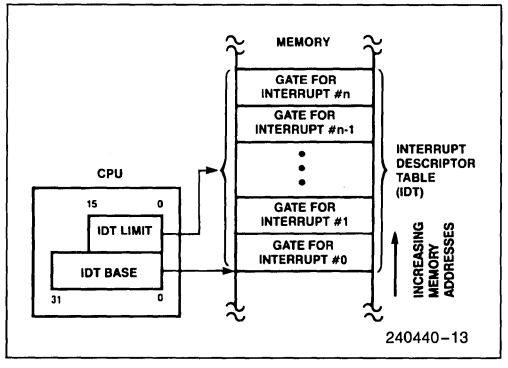


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

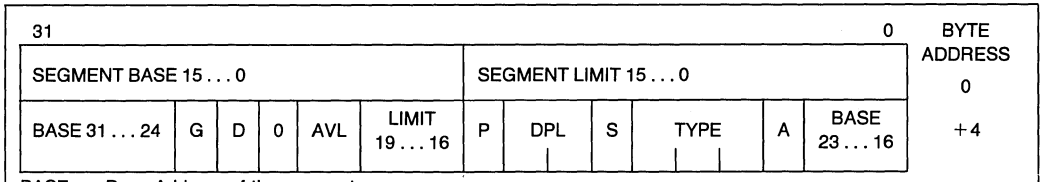
4.3.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the 486 Microprocessor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The 486 Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 i486™ CPU Code, Data Descriptors (S=1)

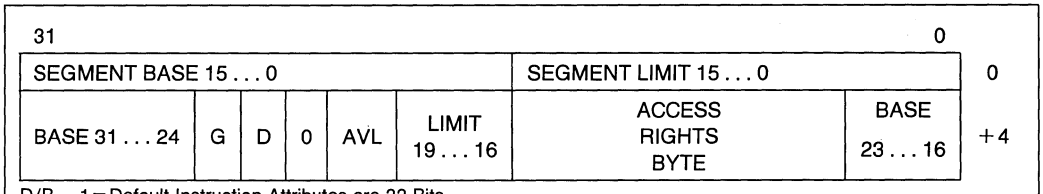
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1 = Present 0 = Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G = 1 and segment limit 19...0 = FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11...000 = 000H).

Figure 4.5. Segment Descriptors



D/B	1 = Default Instruction Attributes are 32-Bits 0 = Default Instruction Attributes are 16-Bits
AVL	Available field for user or OS
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
0	Bit must be zero (0) for compatibility with future processors

Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function	
	7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits, 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 Code or Data (includes stacks) segment descriptor. S = 0 System Segment Descriptor or Gate Descriptor.	
Type Field Definition	3	Executable (E)	E = 0 Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.	
	3	Executable (E)	E = 1 Descriptor type is code segment:	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 Code segment may not be read. R = 1 Code segment may be read.	
	0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 486 Microprocessor segments can be one megabyte long with byte granularity (G = 0) or four gigabytes with page granularity (G = 1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A 486 Microprocessor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E = 1, S = 1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R = 0, and execute/read if R = 1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D = 1 then 32-bit operands and 32-bit addressing modes are assumed. If D = 0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the 486 Microprocessor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C = 1, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection.**)

Segments identified as data segments (E = 0, S = 1) are used for two types of 486 Microprocessor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W=0. The stack segment must have W=1.

The **B** bit controls the size of the stack pointer register. If B=1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B=0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. 486 Microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S=0, TYPE=2)

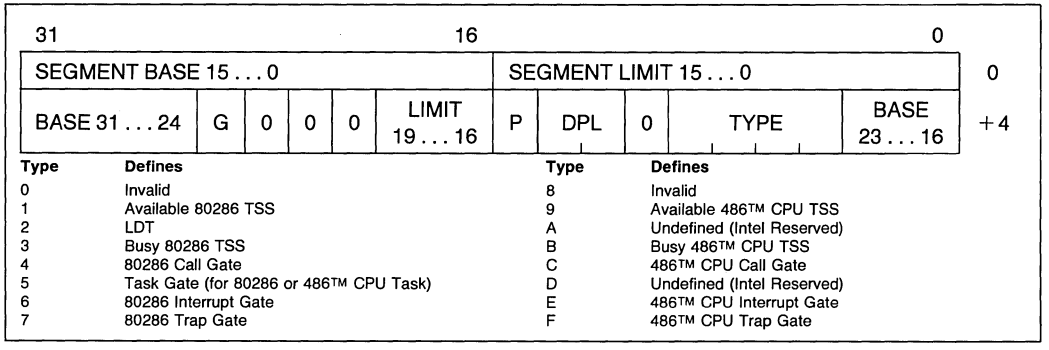
LDT descriptors (S=0, TYPE=2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S=0, TYPE=1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or a 486 Microprocessor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S=0, TYPE=4-7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of 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 processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available 486™ CPU TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy 486™ CPU TSS
4	80286 Call Gate	C	486™ CPU Call Gate
5	Task Gate (for 80286 or 486™ CPU Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	486™ CPU Interrupt Gate
7	80286 Trap Gate	F	486™ CPU Trap Gate

Figure 4.7. System Segments Descriptors

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

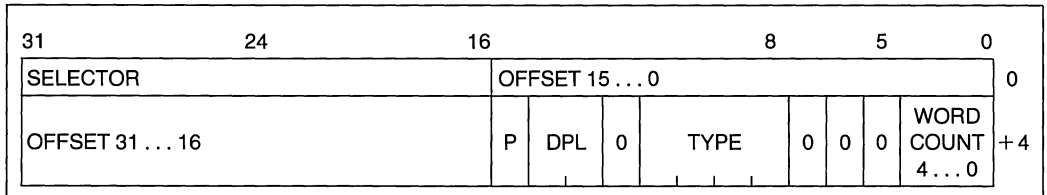
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

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 (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences Between i486™ Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and 486 Microprocessor, the 486 Microprocessor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and 486 Microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 486 Microprocessor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 486 Microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or 486™ CPU task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	486™ CPU call gate
	E	486™ CPU interrupt gate
	F	486™ CPU trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for 486™ CPU gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit 486™ CPU	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats

By supporting 80286 system segments the 486 Microprocessor is able to execute 80286 application programs on a 486 Microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are 486 Microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and 486 Microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 486 Microprocessor call gates. The B bit controls the size of PUSHes when using a call gate; if B=0 PUSHes are 16 bits, if B=1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

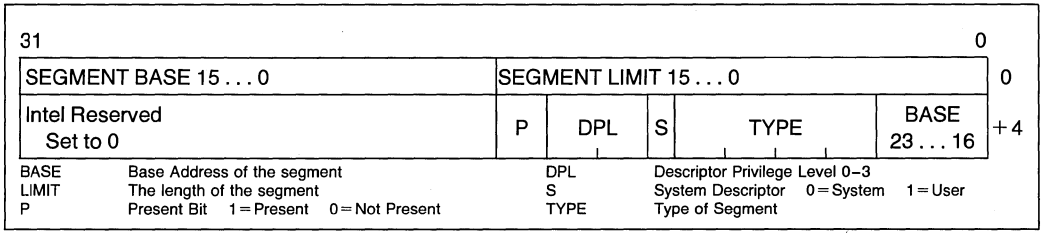


Figure 4.9. 80286 Code and Data Segment Descriptors

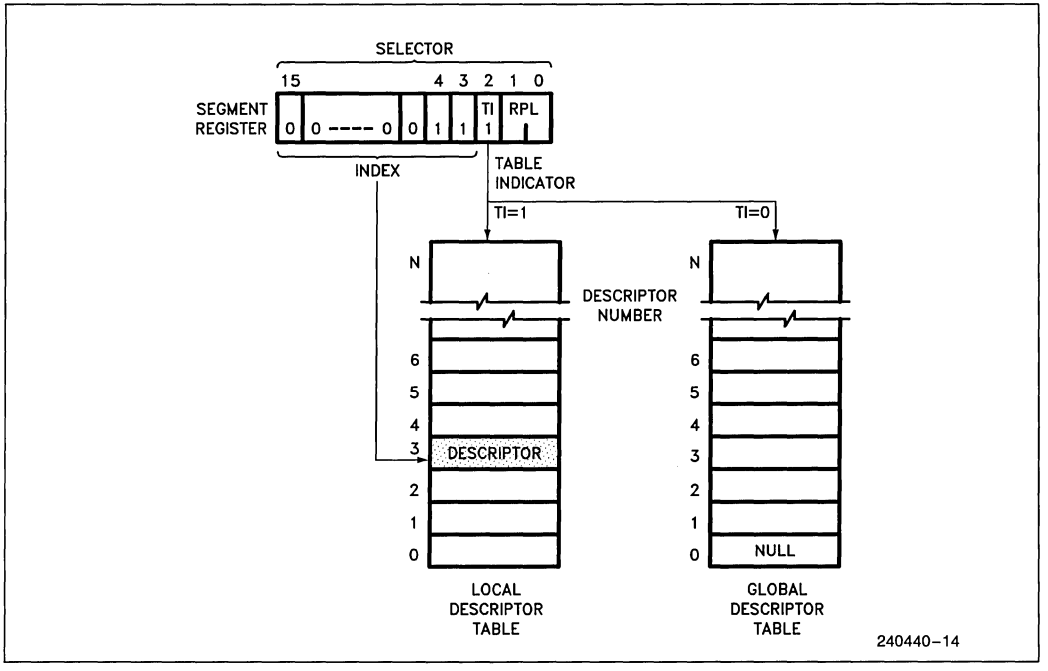
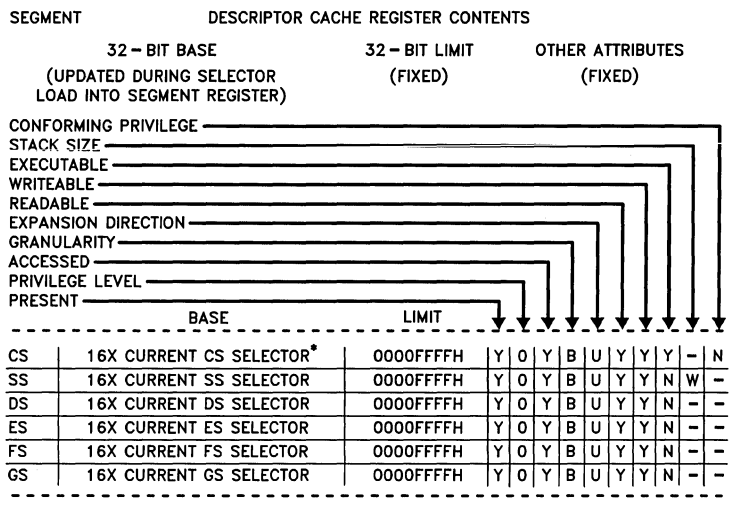


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the 486 Microprocessor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

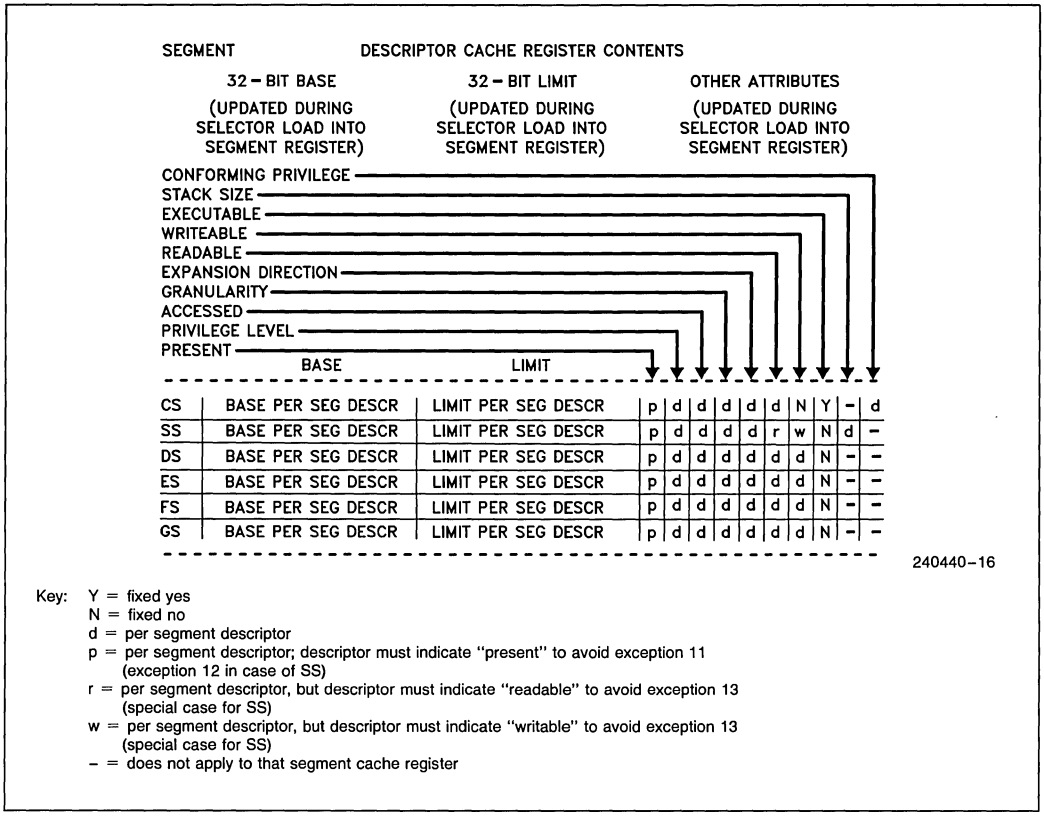


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4.11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

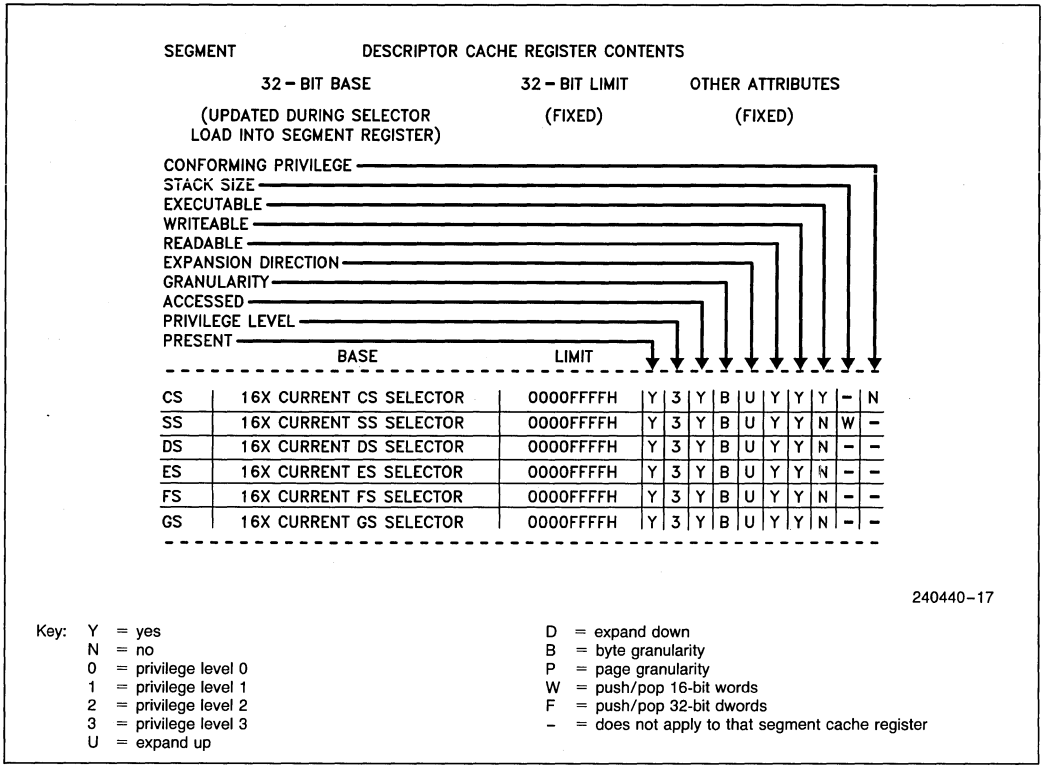
When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.



Key: Y = yes
N = no
0 = privilege level 0
1 = privilege level 1
2 = privilege level 2
3 = privilege level 3
U = expand up
D = expand down
B = byte granularity
P = page granularity
W = push/pop 16-bit words
F = push/pop 32-bit dwords
– = does not apply to that segment cache register

Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

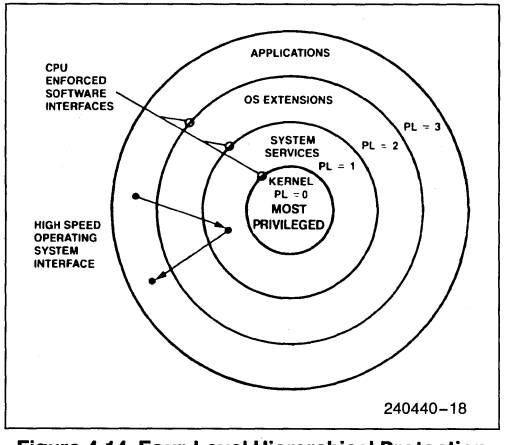


Figure 4.14. Four-Level Hierarchical Protection

The 486 Microprocessor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 486 Microprocessor provides the protection as part of its integrated Memory Management Unit. The 486 Microprocessor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the 486 Microprocessor paging

mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The 486 Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the 486 Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with a 486 Microprocessor TSS, the I/O Permission Bitmap (part of a 486 Microprocessor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the 486 Microprocessor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4.2. 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.
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.

4.4.3.4 Privilege Validation

The 486 Microprocessor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the 486 Microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. 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 as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 486 Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Table 4.3. 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
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged 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's word count field) 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.

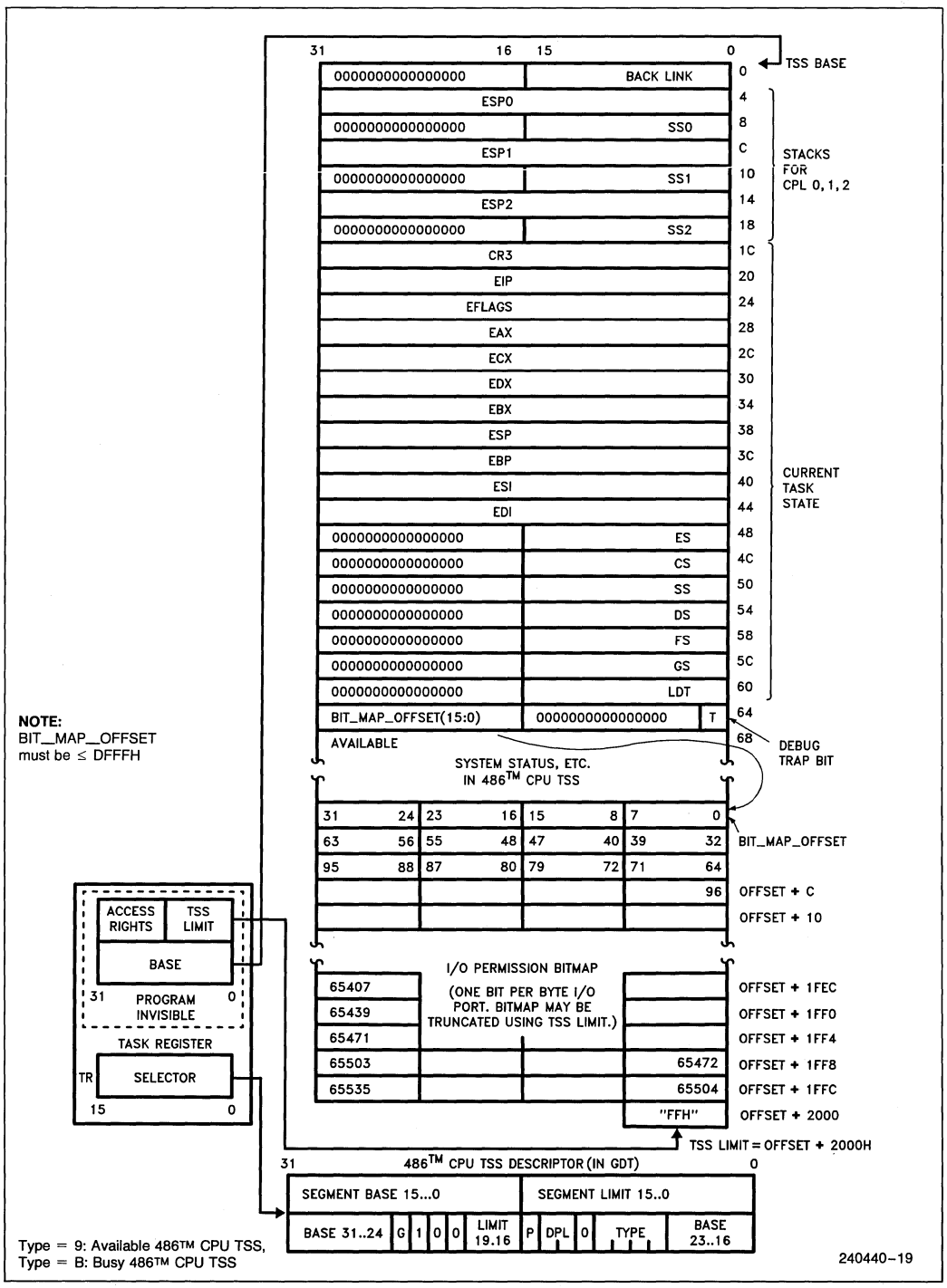


Figure 4.15a. i486™ Microprocessor TSS and TSS Registers

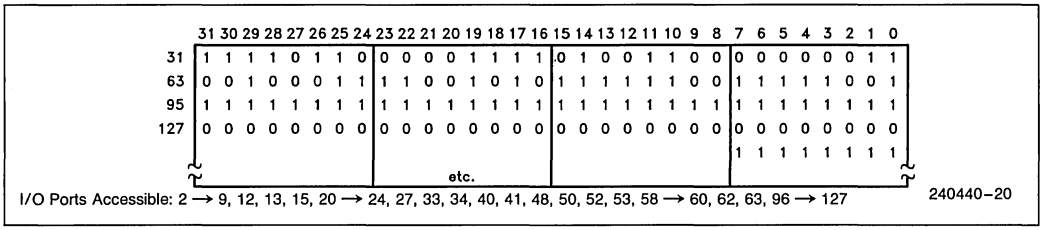


Figure 4.15b. Sample I/O Permission Bit Map

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 486 Microprocessor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 486 Microprocessor directly supports this operation by providing a task switch instruction in hardware. The 486 Microprocessor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 microseconds. Like transfer of control via 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 a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire 486 Microprocessor execution state while a task gate descriptor contains a TSS selector. The 486 Microprocessor supports both 80286 and 486 Microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of a 486 Microprocessor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 486 Microprocessor called the Task State Segment 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. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

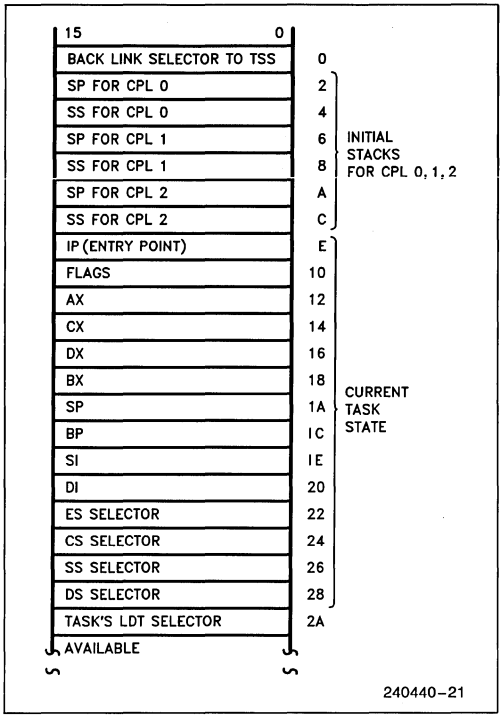


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the 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. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The 486 Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS 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 an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the 486 Microprocessor switches tasks, it sets the TS bit. The 486 Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the 486 Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the 486 Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode 486 Microprocessor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 486 Microprocessor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

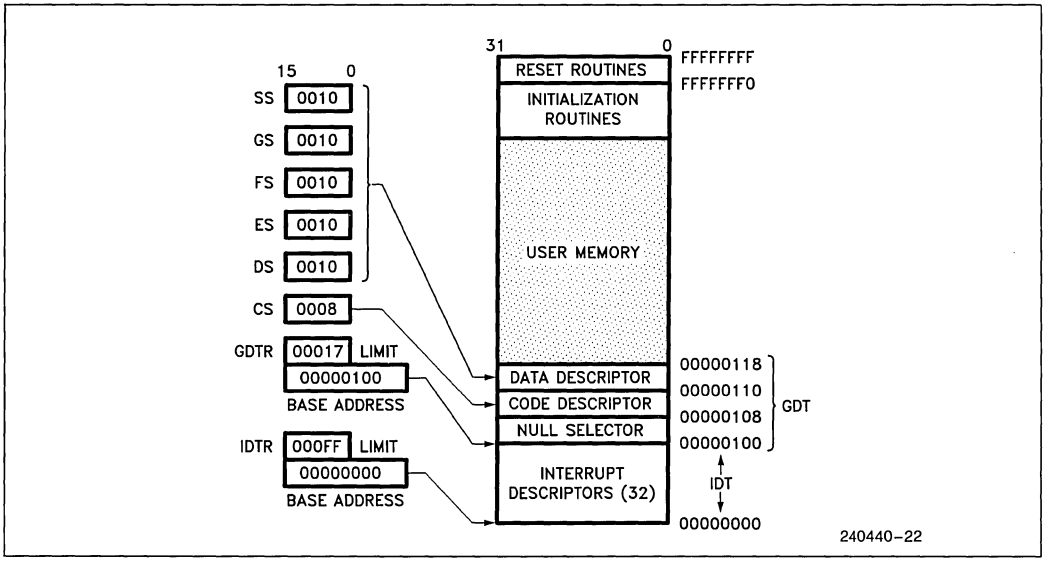


Figure 4.17. Simple Protected System

	2	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)
DATA DESCRIPTOR		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)								
	1	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)
CODE DESCRIPTOR		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)								
		NULL DESCRIPTOR														
	0															
		31			24		16	15			8					0

Figure 4.18. GDT Descriptors for Simple System

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 486 Microprocessor system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The 486 Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 486 Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 486 Microprocessor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 **Translation Lookaside Buffer**).

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

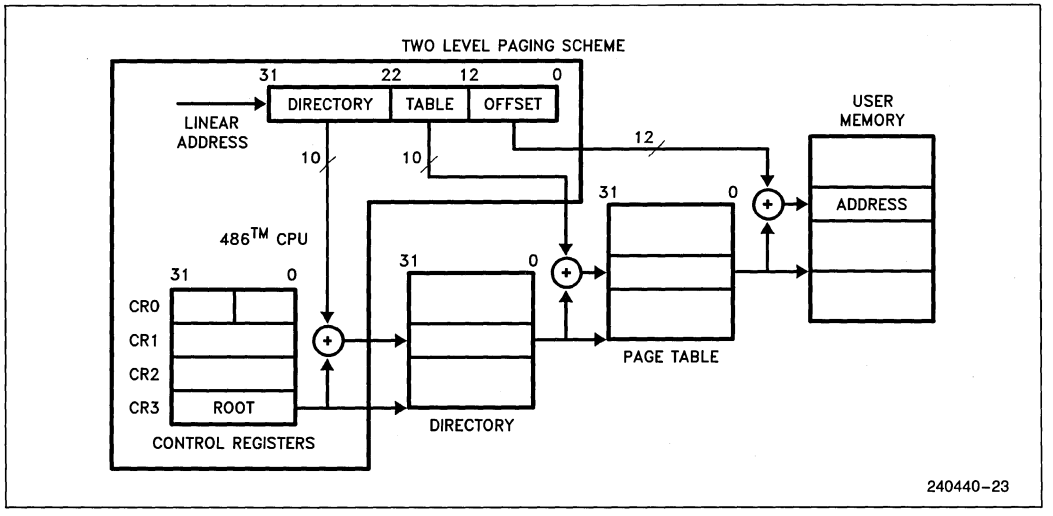


Figure 4.19. Paging Mechanism

31 12	11 10 9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED	0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation if P = 0 the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 486 Microprocessor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 486 Microprocessor, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The 486 microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The 386 microprocessor does not contain the WP bit. The WP bit has been added to the 486 microprocessor to protect read-only pages from supervisor write accesses. The 386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the 386 microprocessor compatible mode. When WP=0 the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP=1 supervisor access to a read-only page (R/W=0) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP=0, the supervisor can write to pages regardless of the state of the R/W bit. When WP=1 and R/W=0 the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S=0), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the 486 microprocessor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CE (cache enable bit) and WT (writes transparent) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CE and WT. (See Section 5.0, **On-Chip Cache**).

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. If paging is not enabled (PG = 0 in CR0), or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and Halt cycles), the PWT and PCD bits are taken from bits 3 and 4 of CR3. These bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The 486 Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 486 Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the 486 Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the 486 microprocessor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.

1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.

2. Read the correct entry in the Page Table and place the entry in the TLB.

2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the 486 on-chip cache just like normal data.

4.5.6 PAGING OPERATION

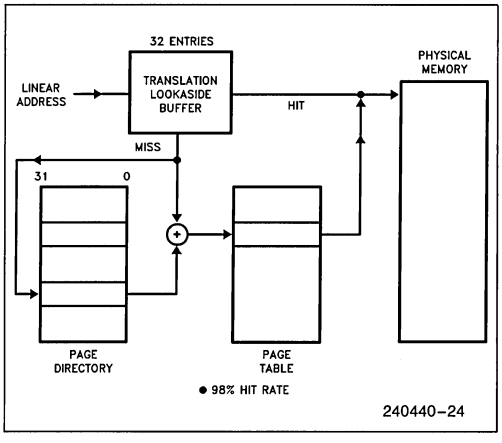


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 486 Microprocessor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the 486 Microprocessor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the 486 Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14 page fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

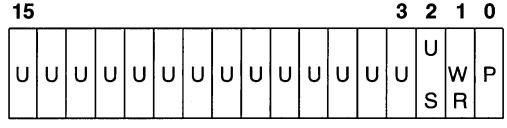


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The 486 Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The 486 Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 486 Microprocessor protection mechanism. In particular, the 486 Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and a 486 Microprocessor operating system and both 80286 and 486 Microprocessor applications. Thus, in a multi-user 486 Microprocessor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between 486 Microprocessor Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 486 Microprocessor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 486 Microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 486 Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the 486 Microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;        STI;
OUT;       CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

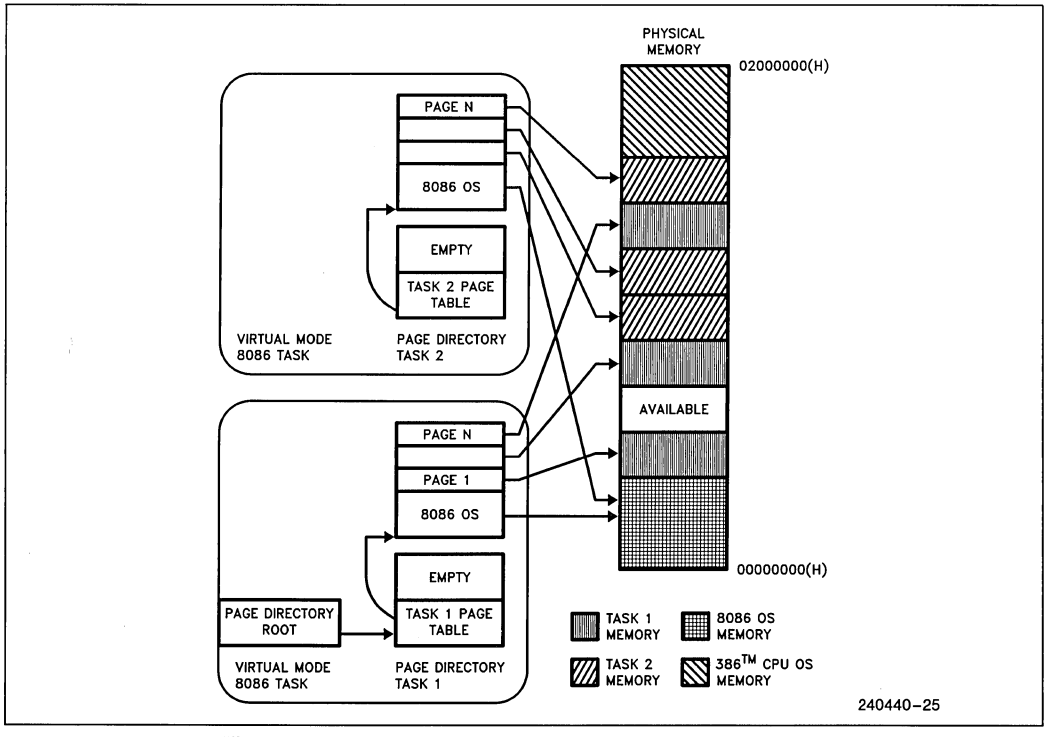


Figure 4.24. Virtual 8086 Environment Memory Management

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **486 Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the 486 Microprocessor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 486 Microprocessor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 486 Microprocessor operating system. The 486 Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 486 Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 486 Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 486 Microprocessor operating system. The 486 Microprocessor operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the 486 Microprocessor operating system could intercept an 8086 operating system's call to "Open a File".

A 486 Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL=0), or Task Switch (at any CPL) to a 486 Microprocessor task whose 486 Microprocessor TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a 486 Microprocessor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM=1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to 486 Microprocessor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 486 Microprocessor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL>0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new 486 Microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 486 Microprocessor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS.

The segment registers in the TSS will contain 8086 segment base values rather than selectors.

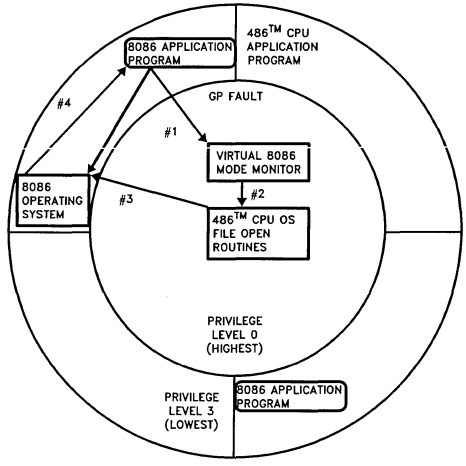
A task switch into a task described by a 486 Microprocessor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 486 Microprocessor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 486 Microprocessor Trap Gate (Type 14), or 486 Microprocessor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL=0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 486 Microprocessor gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 486 Microprocessor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.



8086 Application makes "Open File Call" → causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls 486™ CPU OS (Arrow #2)
486™ CPU OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected 486 Microprocessor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended 486 Microprocessors IRET instruction (operand size=32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If

VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.

Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5.0 ON-CHIP CACHE

To meet its performance goals the 486 microprocessor contains an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the x86 architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

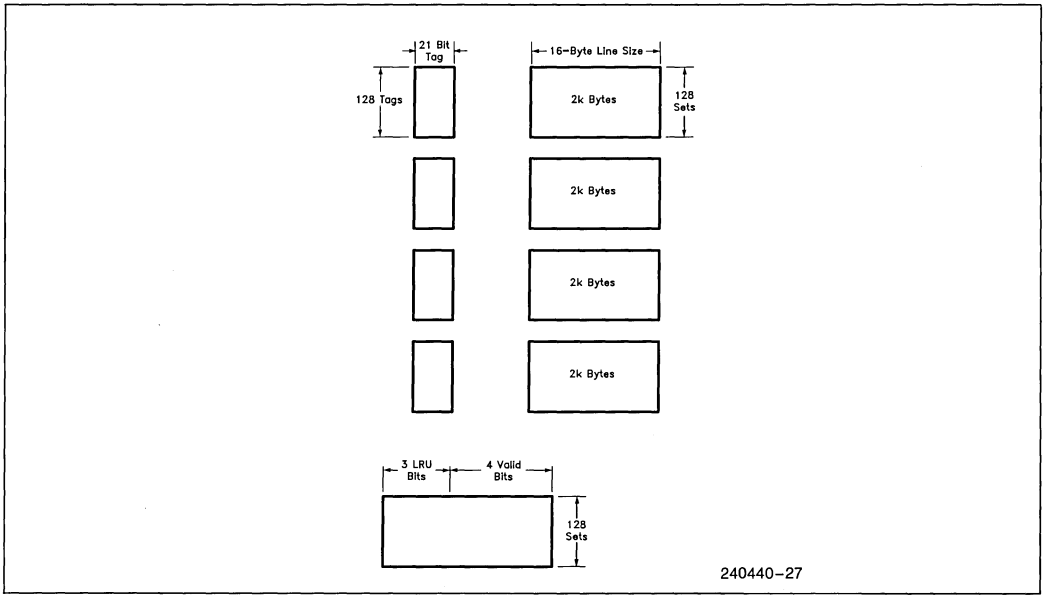


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CE and WT bits in CR0. CE enables and disables the cache. WT controls memory write-through and invalidates.

The CE and WT bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

The **CE** and **WT** bits define four operating modes of the on-chip code and data cache, as given in the following table:

Table 5.1. Cache Operating Modes

CE	WT	Operating Mode
0	0	Cache fills disabled, write-through and invalidates disabled
0	1	Cache fills disabled, write-through and invalidates enabled
1	0	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
1	1	Cache fills enabled, write-through and invalidates enabled

CE = 0, WT = 0

The cache is completely disabled by setting CE = 0 and WT = 0 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CE = 0 and WT = 0. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CE = 0, WT = 1

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CE = 1, WT = 0

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CE = 1, WT = 1

This is the normal operating mode.

Completely disabling the cache is a two step process. First CE and WT must be set to 0 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the 486 microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the 486 microprocessor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the 486 microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the 486 microprocessor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The 486 microprocessor contains a mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the 486 microprocessor's cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the 486 microprocessor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

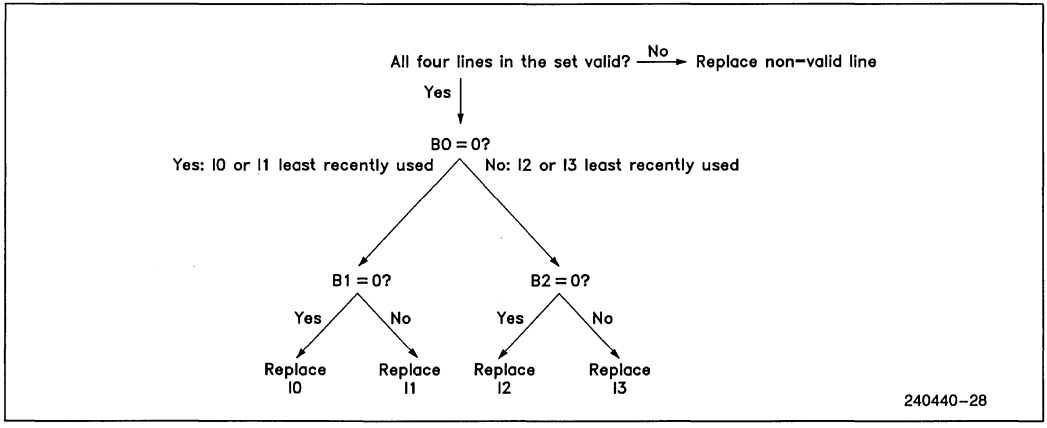


Figure 5.2. On-Chip Cache Replacement Strategy

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the 486 microprocessor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 allows the possibility of write-back. The state of PWT is ignored internally by the 486 microprocessor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

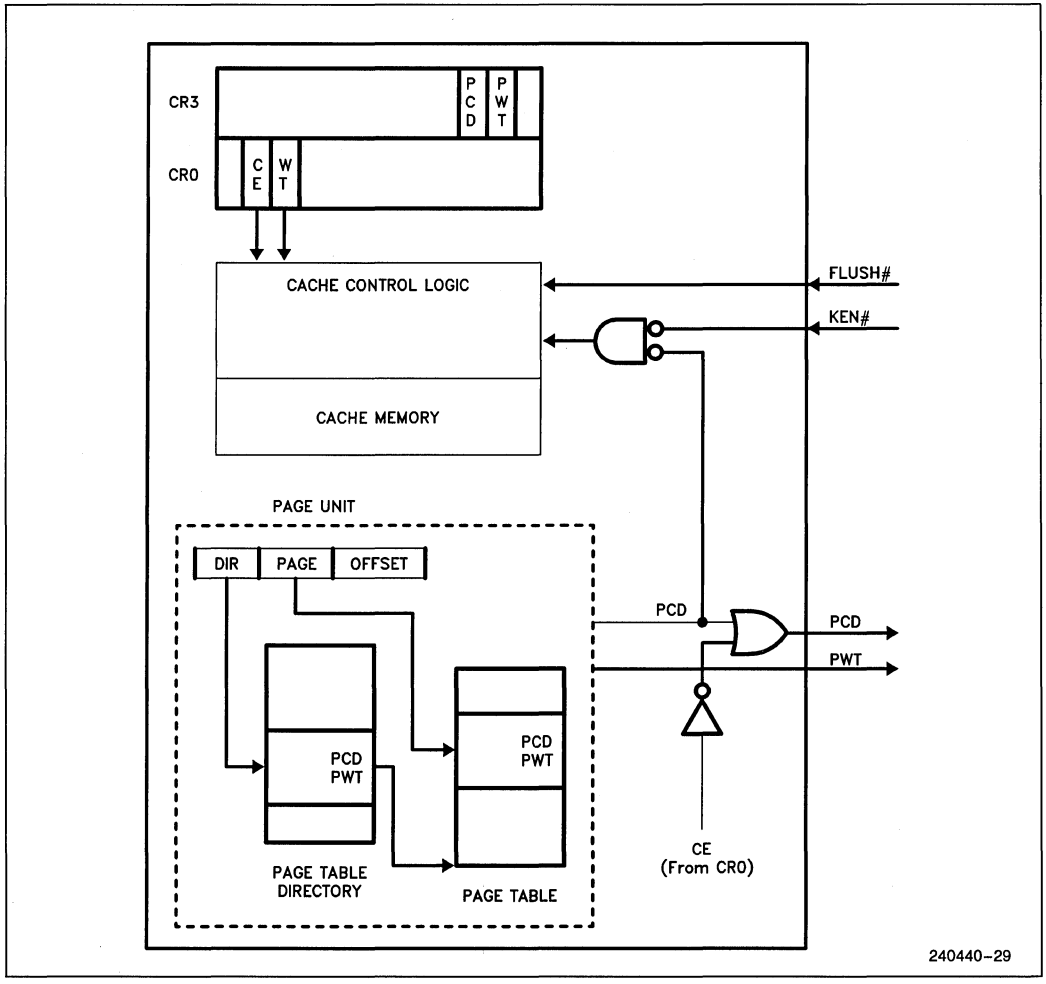


Figure 5.3. Page Cacheability

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the 486 microprocessor if the area is cacheable. The 486 microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

The PCD bit is masked with the CE (cache enable) bit in control register 0 to determine the state of the PCD pin. If CE=0 the 486 microprocessor forces the PCD pin HIGH. If CE=1 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. If paging is not enabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles), the PWT and PCD bits are taken from CR3. These bits are initialized to 0 on reset, but can be set to any value by level 0 software.

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the 486 microprocessor are signalled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the 486 microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the 486 microprocessor's on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the 486 microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The 486 microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the 386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The 486 microprocessor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The 486 microprocessor bus has been designed to be similar to the 386 microprocessor bus whenever possible. Several new features have been added to the 486 microprocessor bus resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a BS8# bus sizing mechanism and parity support.

The 486 microprocessor is driven by a 1X clock as opposed to a 2X clock in the 386 microprocessor. A 25 MHz 486 microprocessor uses a 25 MHz clock in contrast to a 25 MHz 386 microprocessor which requires a 50 MHz clock. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the 386 microprocessor, the 486 microprocessor has separate parallel busses for data and addresses. The bidirectonal data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses external memory in the same manner as the 386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

The 486 microprocessor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

The 486 microprocessor has a bus hold feature similar to that of the 386 microprocessor. During bus hold, the 486 microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The 486 microprocessor has an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

Ahead is a brief description of the 486 microprocessor input and output signals arranged by functional

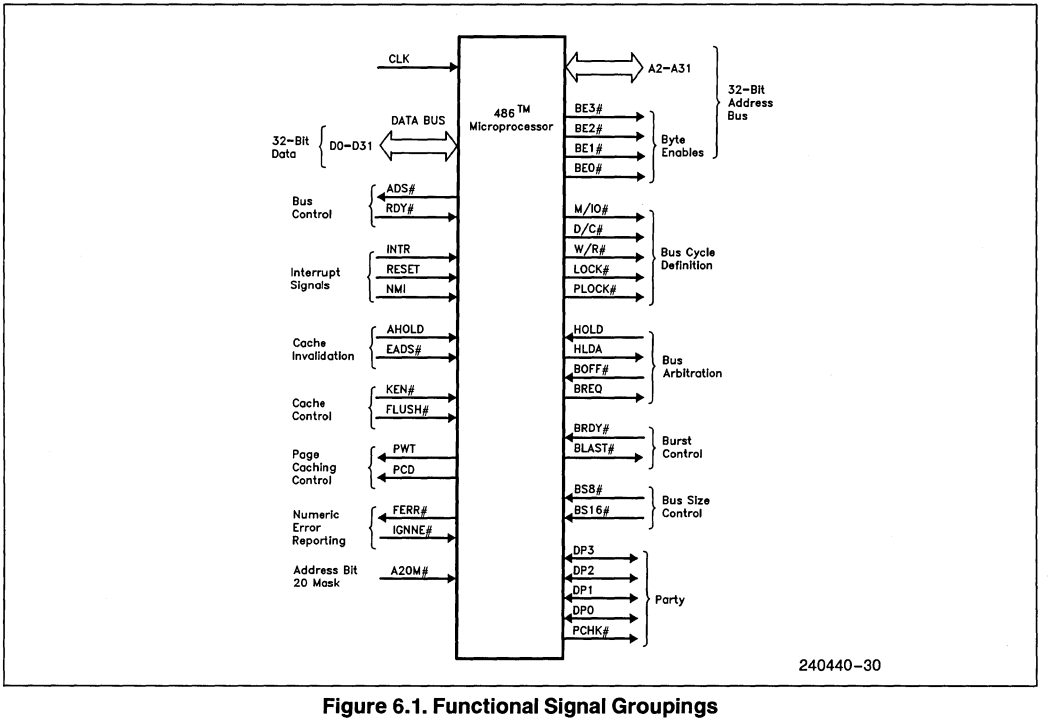


Figure 6.1. Functional Signal Groupings

groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the 486 microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

The 486 microprocessor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 Address Bus (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The 486 microprocessor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the 486 microprocessor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

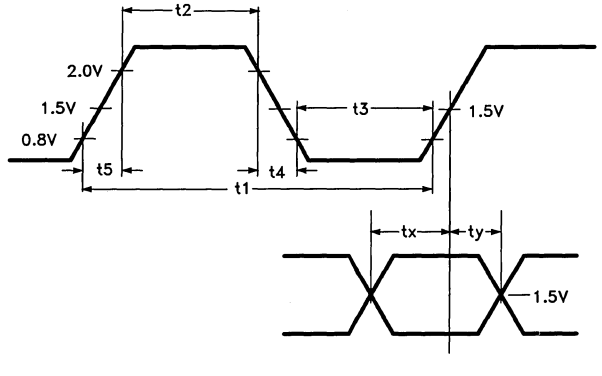
The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

BE3# applies to D24–D31
BE2# applies to D16–D23
BE1# applies to D8–D15
BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the 486 microprocessor. D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.



tx = input setup times
ty = input hold times, output float, valid and hold times

Figure 6.2. CLK waveform

D31–D0 are active HIGH. For reads, D31–D0 must meet the setup and hold times, t_{22} and t_{23}. D31–D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0–DP3)

DP0–DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back to the 486 microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0–DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the 486 microprocessor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the 486 microprocessor. The 486 microprocessor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0–DP3 should be connected to V_{CC} through a pullup resistor.

6.2.5 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the 486 microprocessor and 386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the 486 microprocessor from location 101 in the 386 microprocessor. Location 101 is now reserved and will never be generated by the 486 microprocessor.

Table 6.1. AD5# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the 486 microprocessor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the 486 microprocessor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The 486 microprocessor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to transfer. The 486 microprocessor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes. The 486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.

Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the 486 microprocessor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the 486 microprocessor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the 486 microprocessor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the 486 microprocessor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the 486 microprocessor to begin execution at a known state. V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms before the 486 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper 486 microprocessor operation. The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

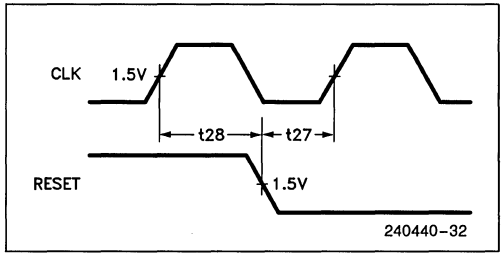


Figure 6.3 Reset Waveform

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The 486 microprocessor drives the BREQ pin active whenever a bus request has been generated internally. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the 486 microprocessor bus. The 486 microprocessor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The 486 microprocessor will maintain its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for a bus hold cycle.

Unlike the 386 microprocessor, the 486 microprocessor will recognize HOLD during reset. Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the 486 microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the 486 microprocessor will resume driving the bus. The 486 microprocessor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the 486 microprocessor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the 486 completes the current bus cycle before floating its bus in response to HOLD. Second the 486 does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the 486 microprocessor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the 486 microprocessor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the 486 microprocessors address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the 486 microprocessor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the 486 microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the 486 microprocessor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The 486 microprocessor will not initiate another bus cycle during address hold. Since the 486 microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pulldown resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the 486 microprocessor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the 486 address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the 486 microprocessor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte

enables were asserted. The 486 microprocessor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tristate test mode of the 486 microprocessor will be invoked on assertion of RESET.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD corresponds to bits 3 and 4 of the page table entry respectively. When paging is disabled, or for cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3.

PCD is masked by the CE (cache enable) bit in control register 0 (CR0). When CE=0 (cache line fills disabled) the 486 microprocessor forces PCD HIGH. When CE=1, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The 486 will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the 486 microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The 486 microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the 387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in 486 microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

Ignore Numeric Error Input (IGNNE#)

The 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted. When deasserted, the 486 microprocessor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The 486 CPU samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the 486 microprocessor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the 486 microprocessor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the 486 microprocessor emulates the 1 Mbyte address wraparound that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For testability, this pin also determines whether or not the external cache test features of the 486 microprocessor will be exercised upon assertion of RESET.

6.3 Write Buffers

The 486 microprocessor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the 486 microprocessor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. The problem with reordering more than one write is illustrated with the following example. A write to external memory location M is pending in the write buffers. This write was a cache hit to location C in the on-chip cache. A read is reordered ahead of the write to location M. The data from this read replaces the data in location C in the on-chip cache. Before the pending write can update location M the processor generates a read to location M. This read is a cache miss since the previous read replaced the information in on-chip location C. If the 486 microprocessor reordered this second read ahead of the pending write to location M, the processor would read stale data.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the 486 microprocessor will update all memory locations before reading status from an I/O device.

The 486 microprocessor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the 486 microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. Repeated OUT instructions will be buffered.

I/O device recovery time must be handled slightly differently by the 486 microprocessor than with the 386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the 386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the 386 microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the 486 microprocessor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the 486 microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the 486 microprocessor will always access external memory, it will never look for the location in the on-chip cache. All data pending in the 486 microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The 486 microprocessor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the microprocessor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The 486 microprocessor provides two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The 486 microprocessor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer. There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the 486 microprocessor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the 486 microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a two-clock synchronizer like that used on the INTR line. Other than the two-clock synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least two clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The 486 microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted on the falling edge of RESET. Refer to Section 8.0 for information on 486 microprocessor testability.

The 486 registers have the values shown in Table 6.2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The EDX register always reflects the chip and revision ID after RESET. The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0004 + Revision ID	0004 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	00000000h	00000000h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

The 486 microprocessor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the 486 microprocessor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

RESET forces the 486 microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

6.5.1 PIN STATE DURING RESET

The 486 microprocessor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the 486 microprocessor bus is in the state shown in Table 6.3 if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized.

Table 6.3. Pin State during Reset

Pin Name	Pin State during Reset, Provided HOLD, AHOLD, and BOFF# are Inactive
A31–A2	Undefined
BE3#–BE0#	Undefined
PWT, PCD	Undefined
M/IO#, D/C#, W/R#	Undefined
D31–D0	High Impedance
DP0–DP3	High Impedance
PCHK#	High
LOCK#	High
PLOCK#	Undefined
ADS#	High
BREQ	Low
HLDA	Low
BLAST#	Undefined
FERR#	Undefined

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The 486 microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0—least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

486™ CPU Address Signals								
A31 ..... A2					BE3#	BE2#	BE1#	BE0#
	Physical Base Address							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

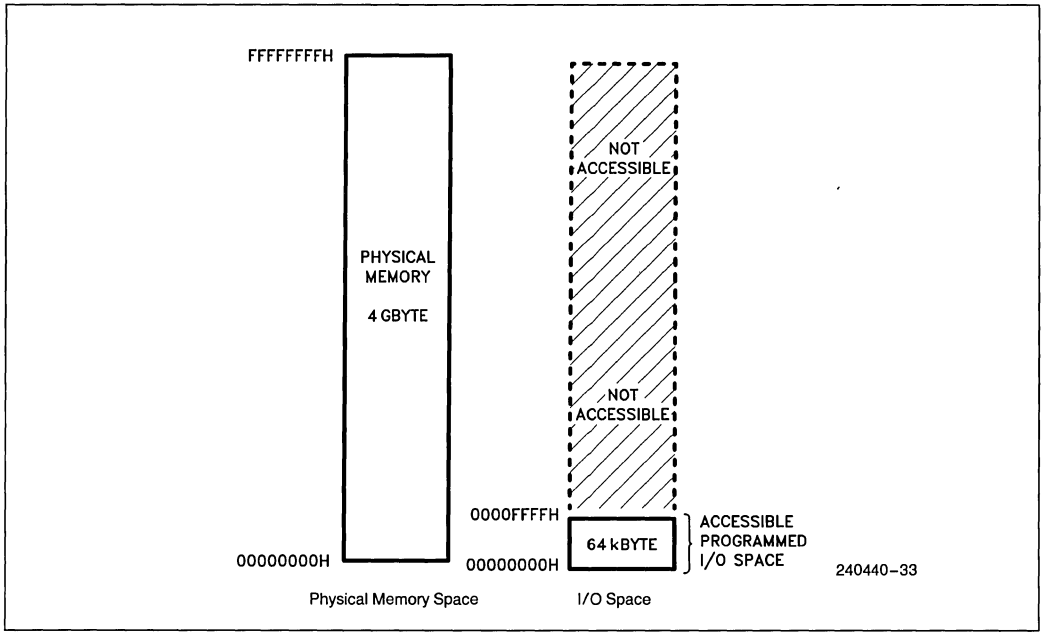


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The 486 microprocessor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The 486 microprocessor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

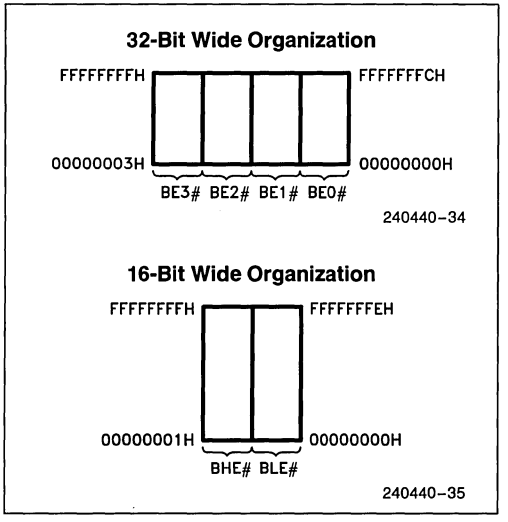


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing direct processor connection to 32-, 16- or 8-bit buses for memory or I/O. A processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the 486 microprocessor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The 486 microprocessor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the 486 microprocessor is significantly different than that of the 386 microprocessor. Unlike the 386 microprocessor, the 486 microprocessor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the 486 microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The 486 microprocessor expects the two low order bytes on D0–D15. The 386 microprocessor expects both the high and low order bytes on D0–D15. The 386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the 486 microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the 486 microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus may be driven but they will not contain valid data. Unlike the 386 microprocessor, the 486 microprocessor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	W BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0#–BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

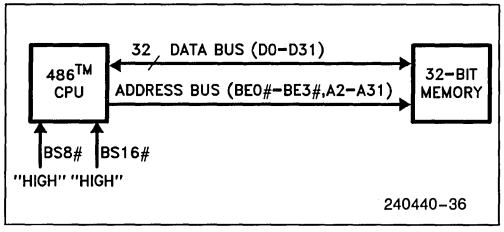


Figure 7.3. i486™ Microprocessor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the 486 microprocessor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#–BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

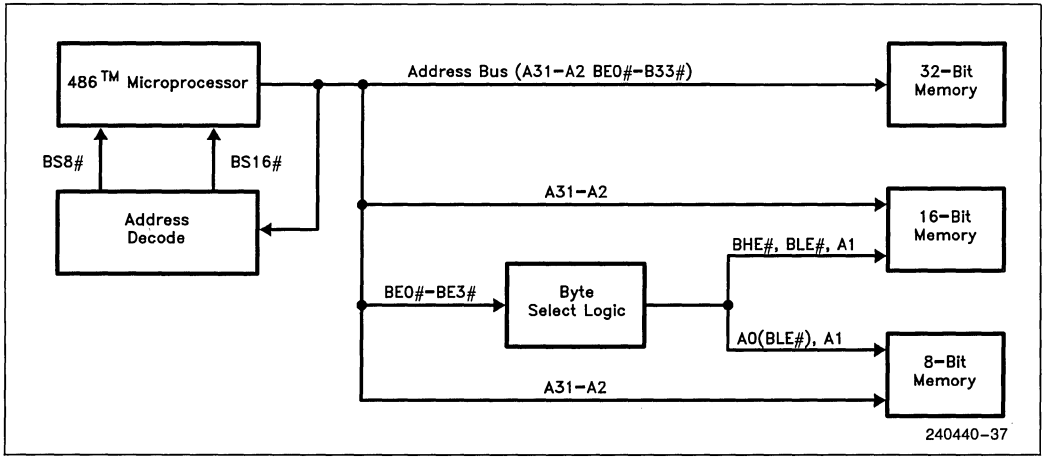


Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

i486™ CPU Signals				8, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

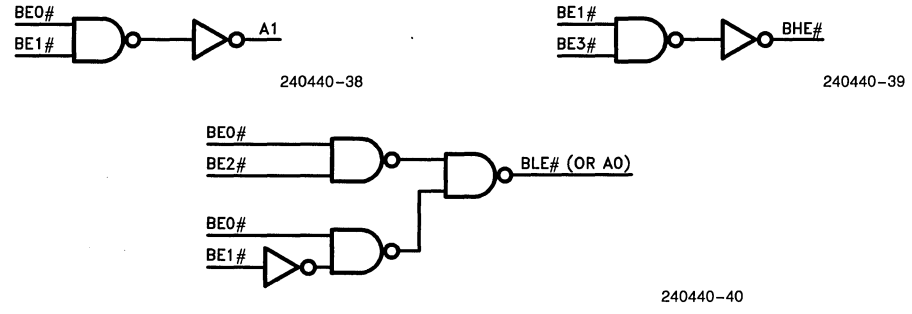


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

Combinations of BE0#–BE3# which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 7.6 shows a 486 microprocessor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the 486 microprocessor on the correct data pins (see Table 7.4).

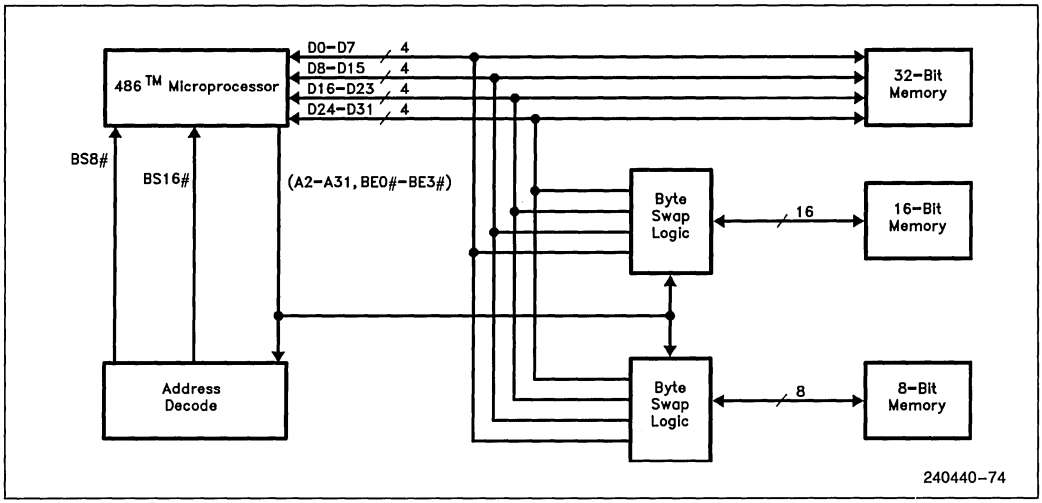


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The 486 microprocessor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The 486 microprocessor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the 486 microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the 486 microprocessor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

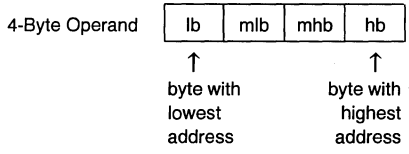
Table 7.6. Generating A0, A1 and BHE# from the i486™ Microprocessor Byte Enables

BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 7.7. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Data Bus ▒ = BS16# Asserted	b	w	lb hb	w	hb lb	lw hw	hb lb mw	hw lw	mw hb lb
Transfer Cycles over 8-Bit Data Bus ▒ = BS8# Asserted	b	lb hb	lb hb	lb hb	hb lb	lb mlb mhb hb	hb lb mlb mhb	mhb hb lb mlb	mlb mhb hb lb

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the 486 microprocessor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

7.2 Bus Functional Description

The 486 microprocessor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked. To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in Section 7.2.3. The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the 486 microprocessor during a data cycle. A bus cycle contains one or more data cycles.

Refer to Section 7.2.13 for a description of the bus states shown in the timing diagrams.

7.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

7.2.1.1 No Wait States

The fastest non-burst bus cycle that the 486 microprocessor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first 2 refers to reads and the

second to writes. For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The 486 microprocessor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The 486 microprocessor samples RDY# at the rising edge of the third clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the 486 microprocessor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The 486 microprocessor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The 486 microprocessor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The 486 microprocessor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to a 486 microprocessor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the 486 microprocessor or by the external memory system. An internal request for a 64-bit floating point load or a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete. The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The 486 microprocessor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the 486 microprocessor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The 486 microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the 486 microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the 486 microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the 486 microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The 486 microprocessor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal 486 microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the 486 microprocessor internal cache lines.

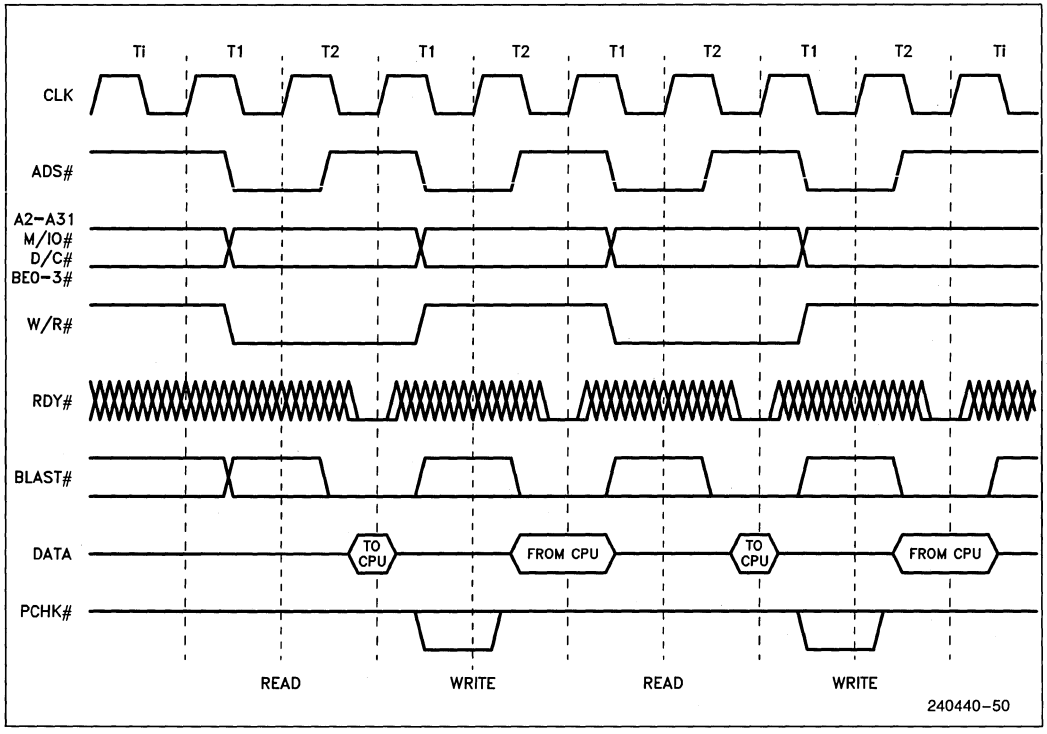


Figure 7.7. Basic 2-2 Bus Cycle

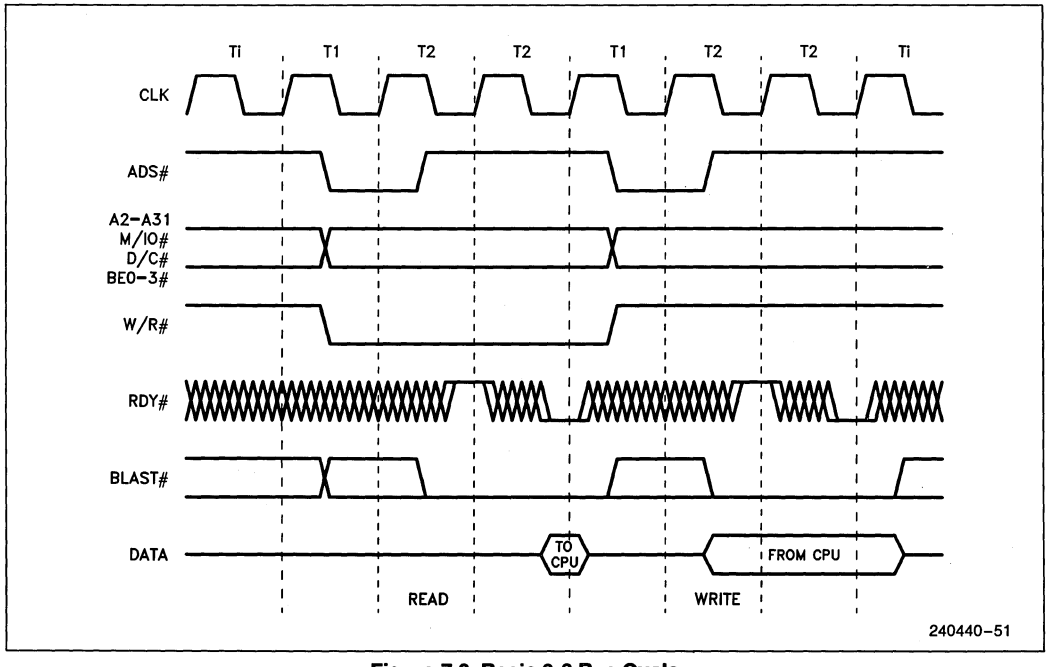


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the 486 microprocessor can be converted into a burst cycle. The 486 microprocessor will only burst the number of bytes needed to complete a transfer. For example, eight bytes will be bursted in for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The 486 microprocessor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the microprocessor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the 486 microprocessor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must tell the microprocessor whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The 486 microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the 486 microprocessor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The 486 microprocessor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the 486 microprocessor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The 486 microprocessor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

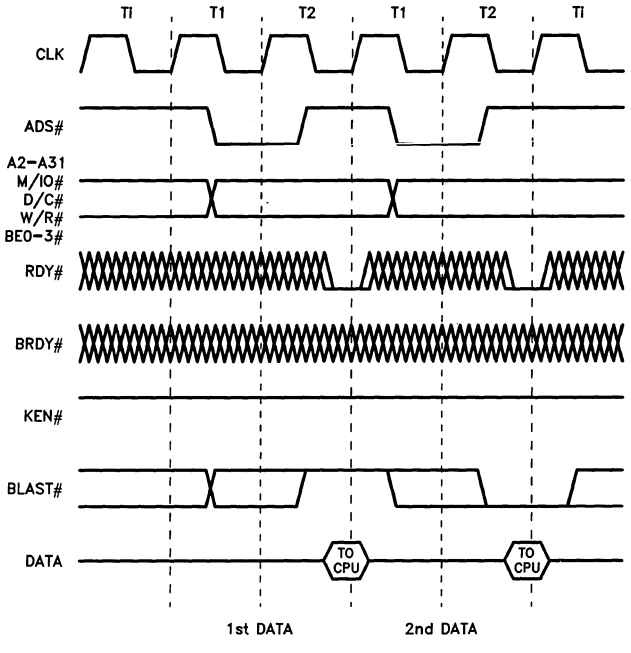


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

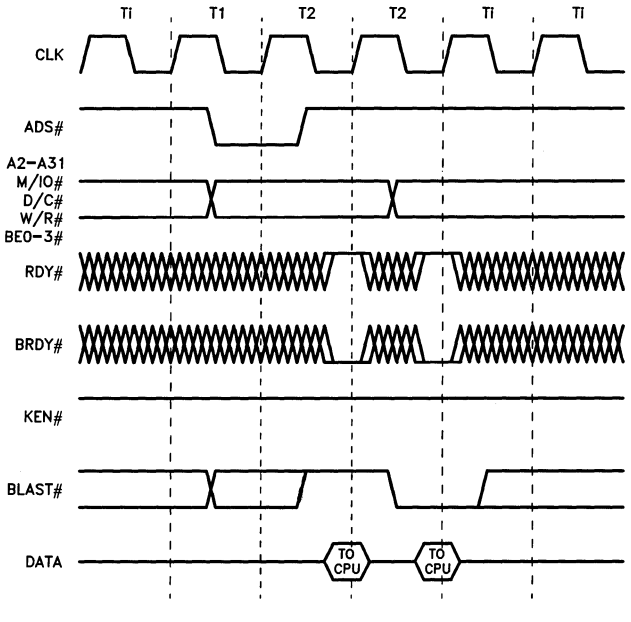


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the 486 microprocessor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The 486 microprocessor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD=0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD=0.
4. The cache enable (CE) bit in control register 0 (CR0) must be set.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The 486 microprocessor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The 486 microprocessor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the 486 microprocessor samples KEN# active at the end of the first clock. The 486 microprocessor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the 486 microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

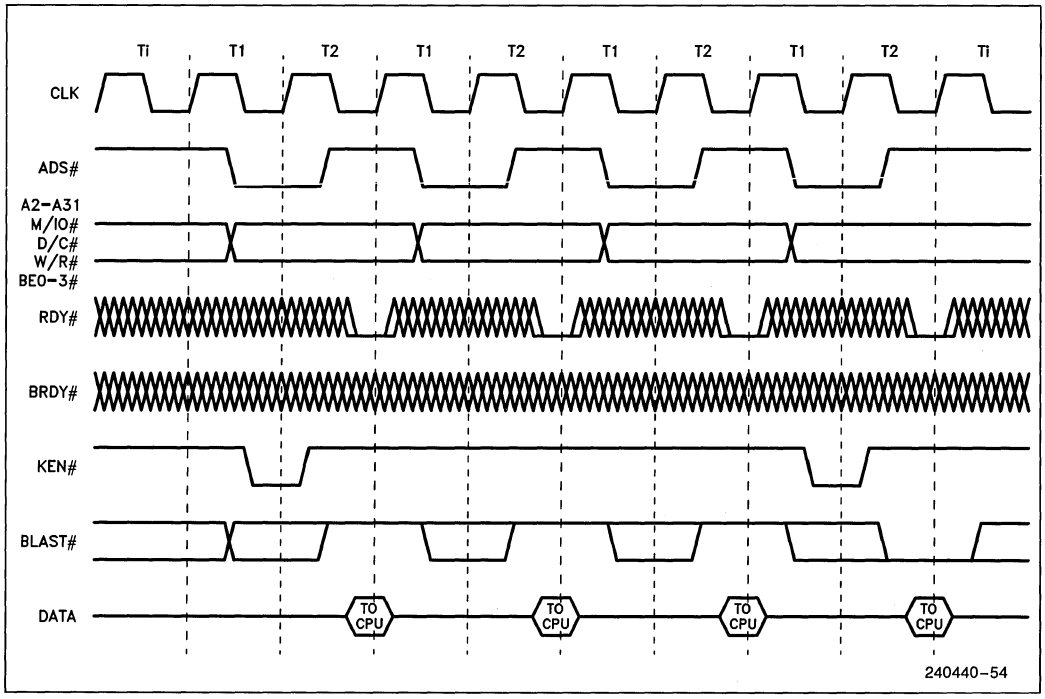


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the 486 microprocessor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

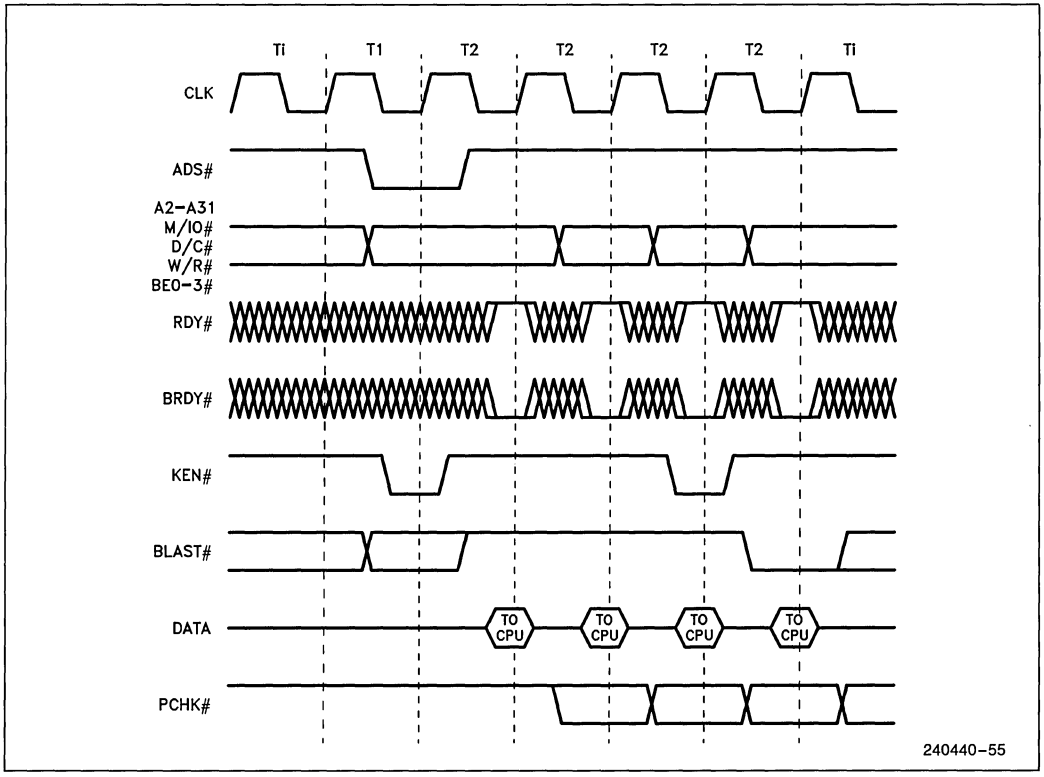


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. In the first clock KEN# is driven active converting the cycle into a cache fill and in the second clock BLAST# is driven inactive in response. In the second clock, KEN# is driven inactive converting the cache fill back to a normal cycle and BLAST# responds by going active in the next clock. Finally in the third clock KEN# goes active again converting the cycle to a cache fill and BLAST# go inactive in the next clock. RDY# is returned active in the fourth clock starting the cache fill.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

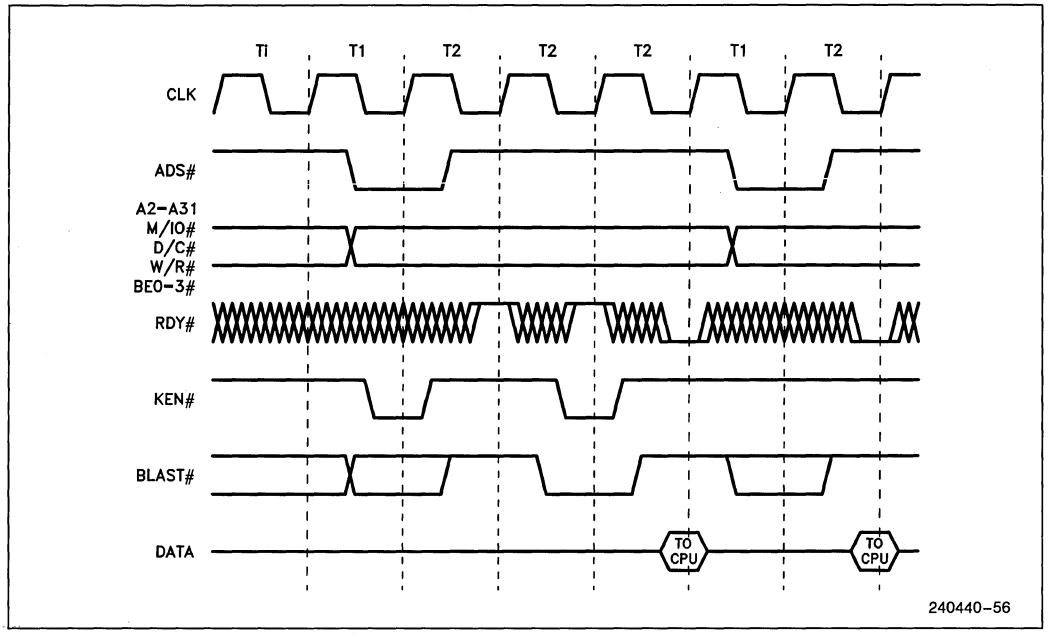


Figure 7.13. Effect of Changing KEN#

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The 486 microprocessor will only strobe data into the chip when either RDY# or BRDY# are active. Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

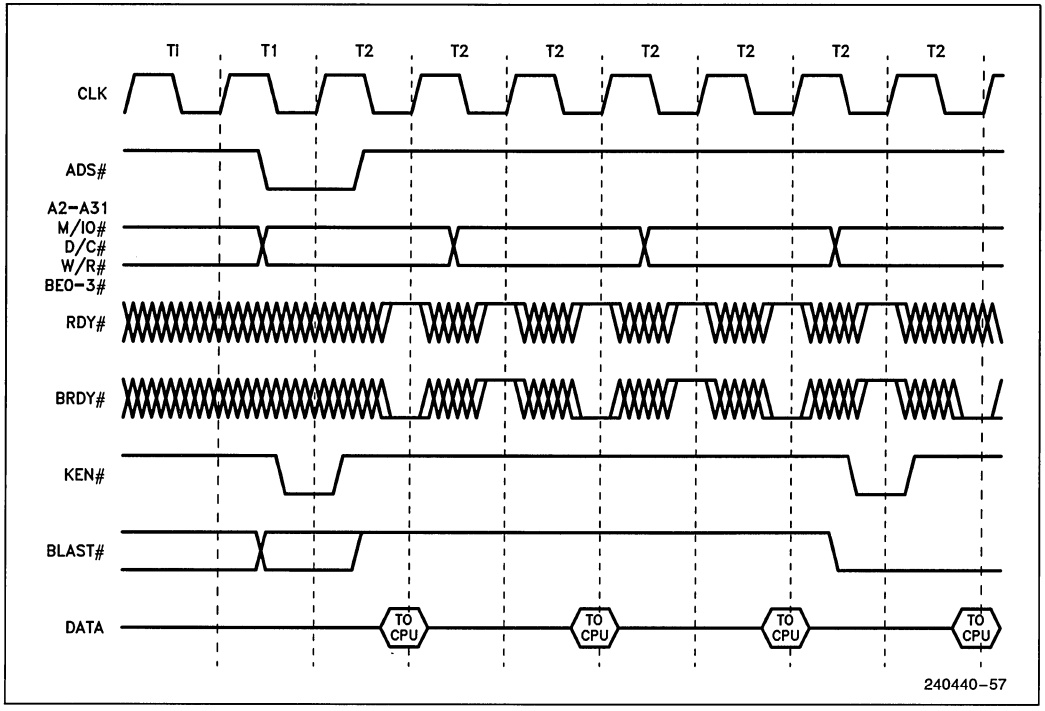


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the 486 microprocessor is shown in Table 7.7. Non-burst cache line fills also follow the order in Table 7.7.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.7. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

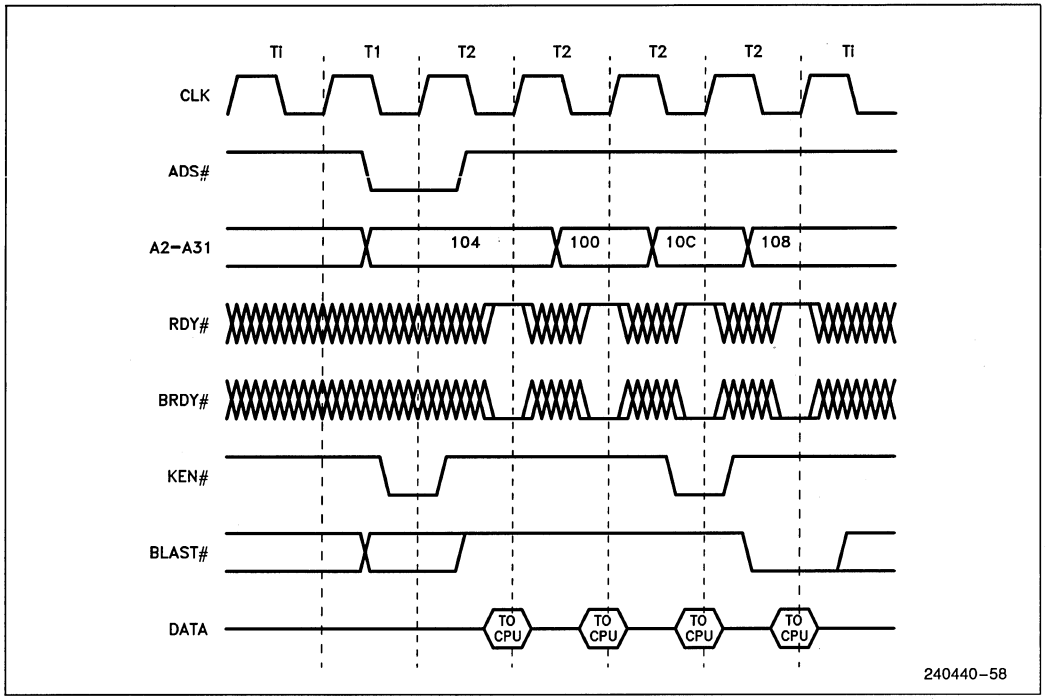


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the 486 microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the 486 microprocessor allows a burst cycle to be interrupted at any time. The 486 microprocessor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The 486 microprocessor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# is driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

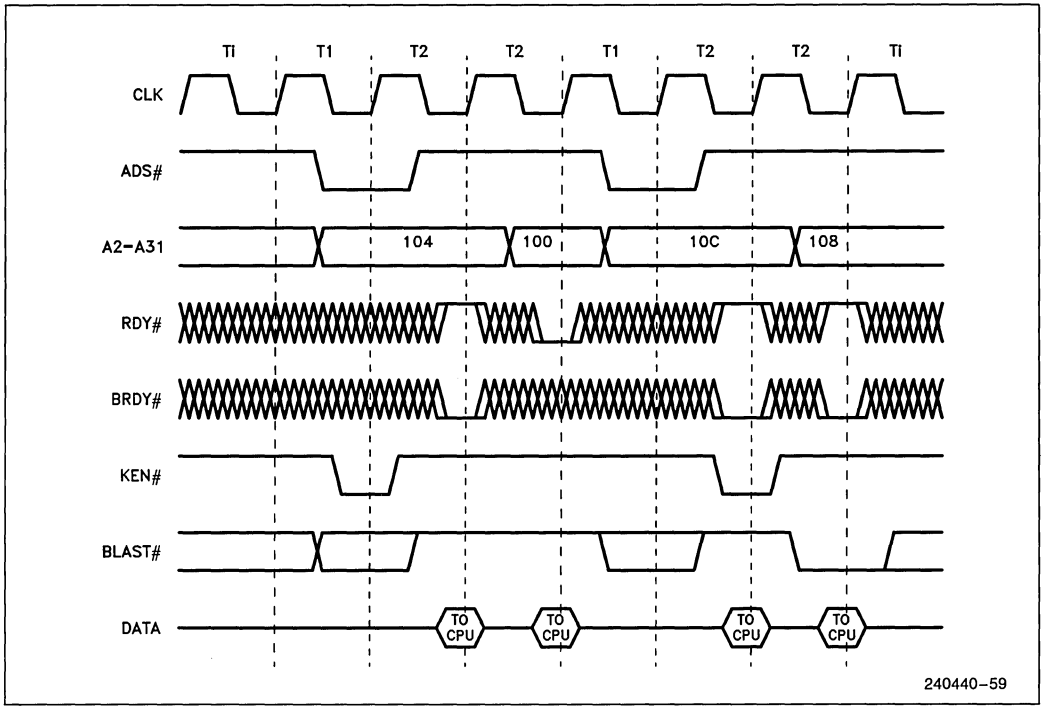


Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the 486 microprocessor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the 486 microprocessor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the 486 microprocessor.

An example of the order in which the 486 microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The 486 microprocessor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The 486 microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the 486 microprocessor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

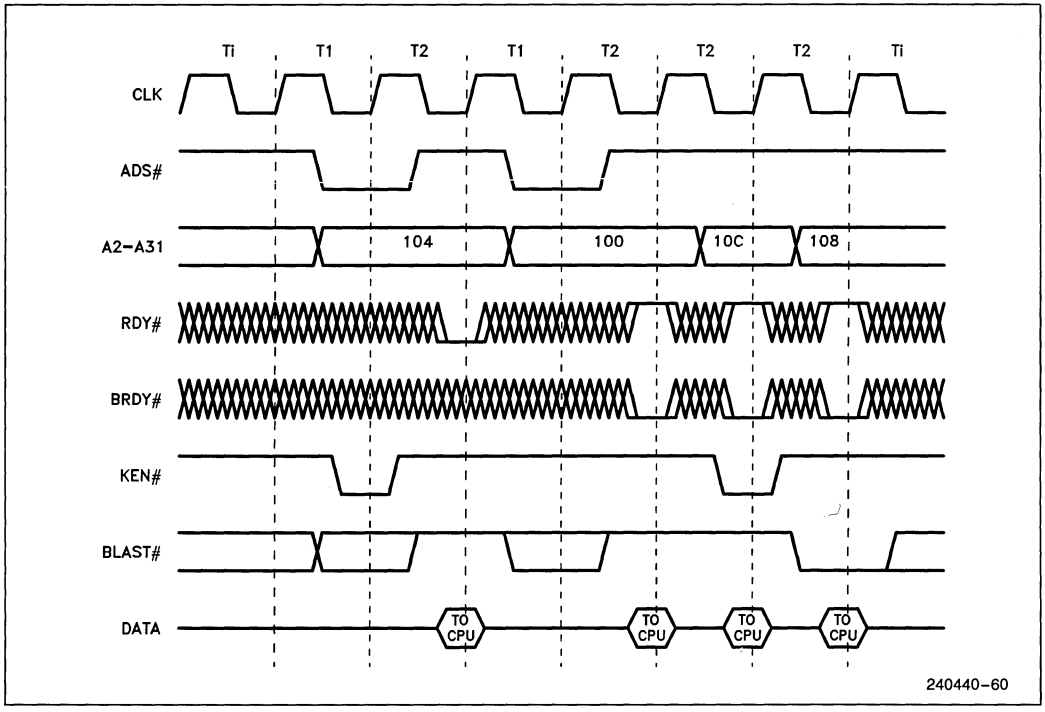


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The 486 microprocessor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the 486 microprocessor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active. Driving the BS16# and BS8# active can force the 486 microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the 486 microprocessor to run two extra cycles to complete a transfer. The 486 microprocessor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The 486 microprocessor issues two extra cycles to complete the transfer.

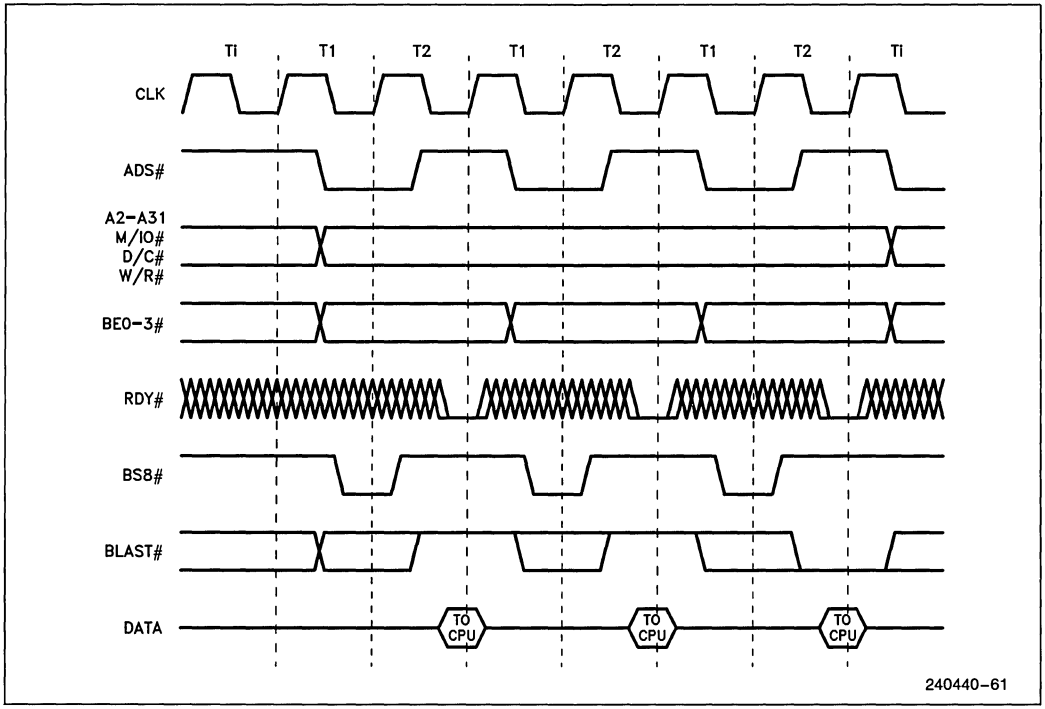


Figure 7.18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The 486 microprocessor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the 486 microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

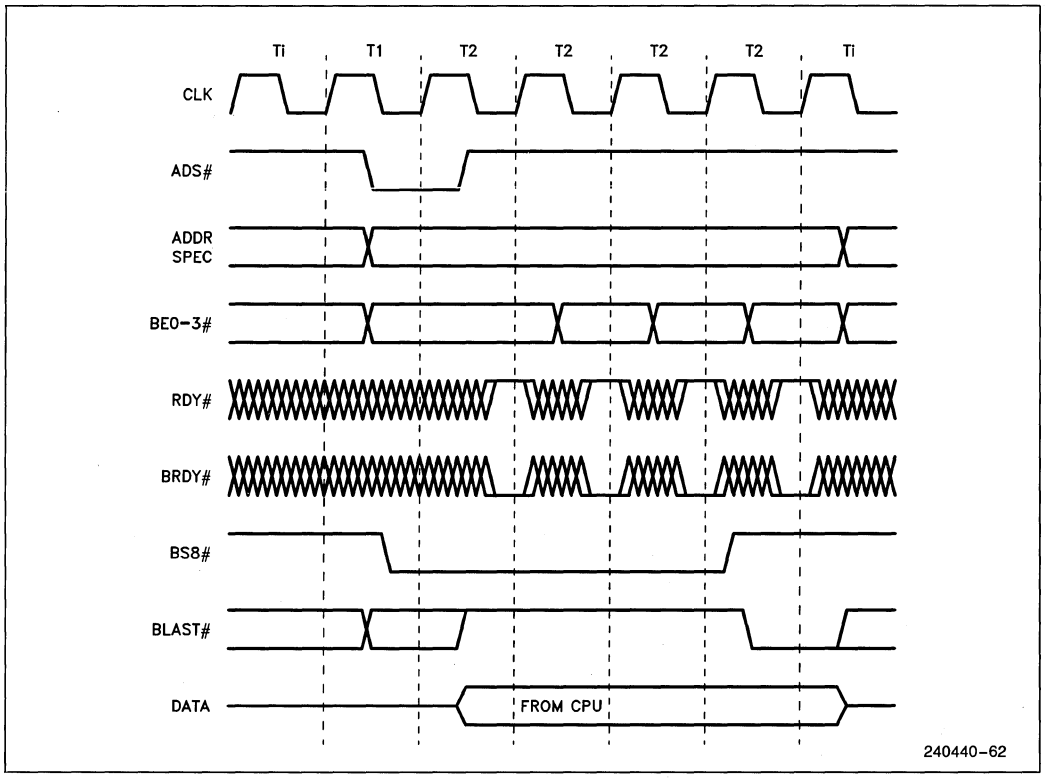


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle.

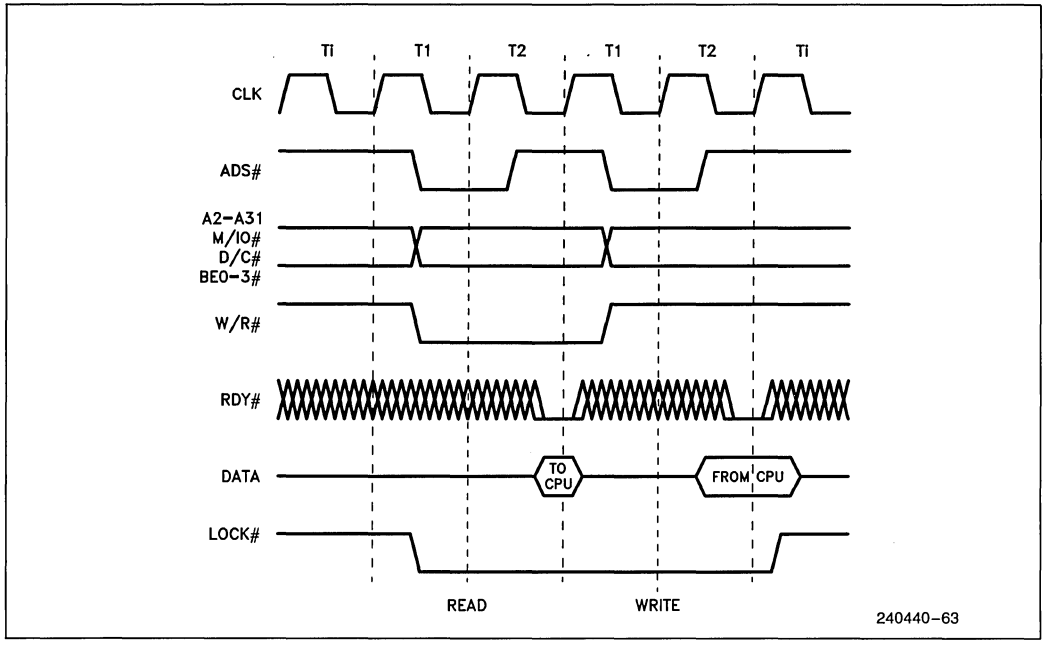


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the 486 microprocessor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the 486 microprocessor generates LOCK#.

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle. Examples include 64-bit floating point read and writes, 64-bit descriptor loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the 486 microprocessor cannot burst write more than 32 bits. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

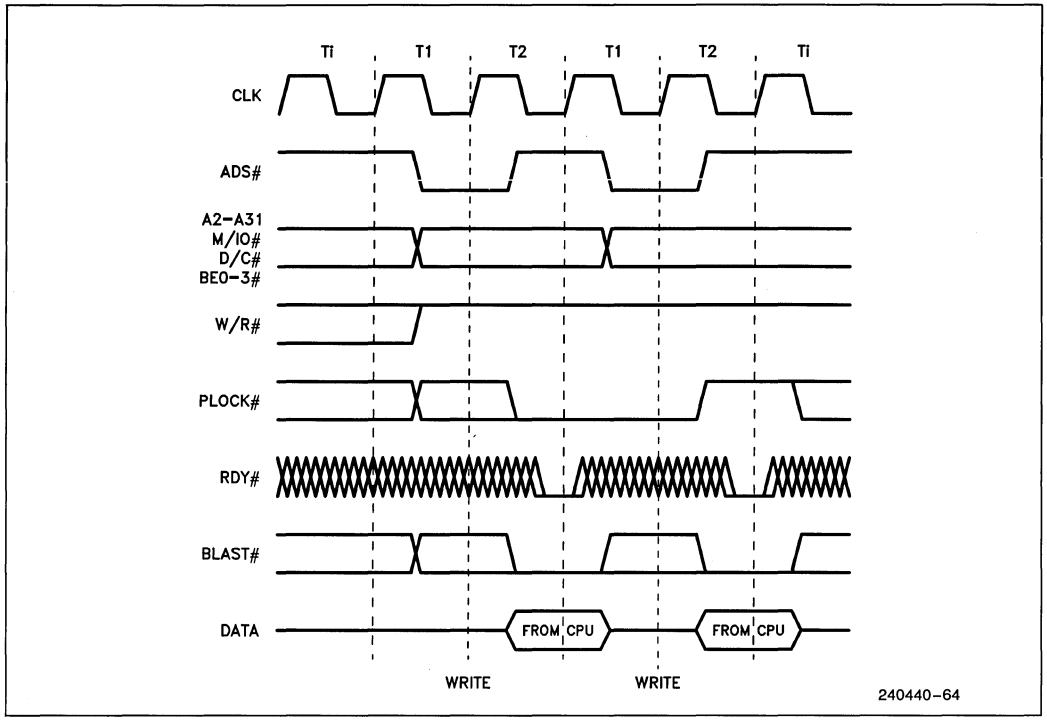


Figure 7.21. Pseudo Lock Timing

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the 486 microprocessor's internal cache contents consistent with external memory. The 486 microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a Section of external memory contained in its internal cache, the processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the 486 microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the 486 microprocessor's address bus. The microprocessor reads the address over its address lines. If the microprocessor finds this address in its internal cache, the cache entry is invalidated. Note that the 486 microprocessor's address bus is input/output unlike the 386 microprocessor's bus, which is output only.

The 486 microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the 486 microprocessor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the 486 microprocessor is driving the address bus, it is possible that the invalidation address will come from the 486 microprocessor itself.

Running an invalidate cycle prevents the 486 microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the rest of the 486 microprocessor.

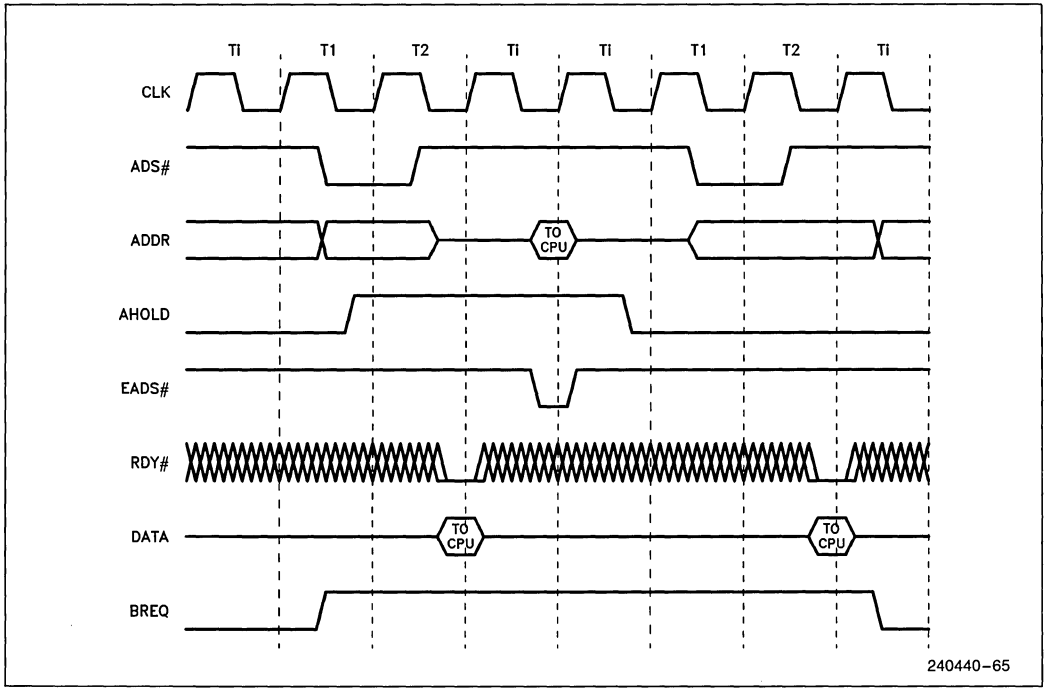


Figure 7.22. Fast Internal Cache Invalidation Cycle

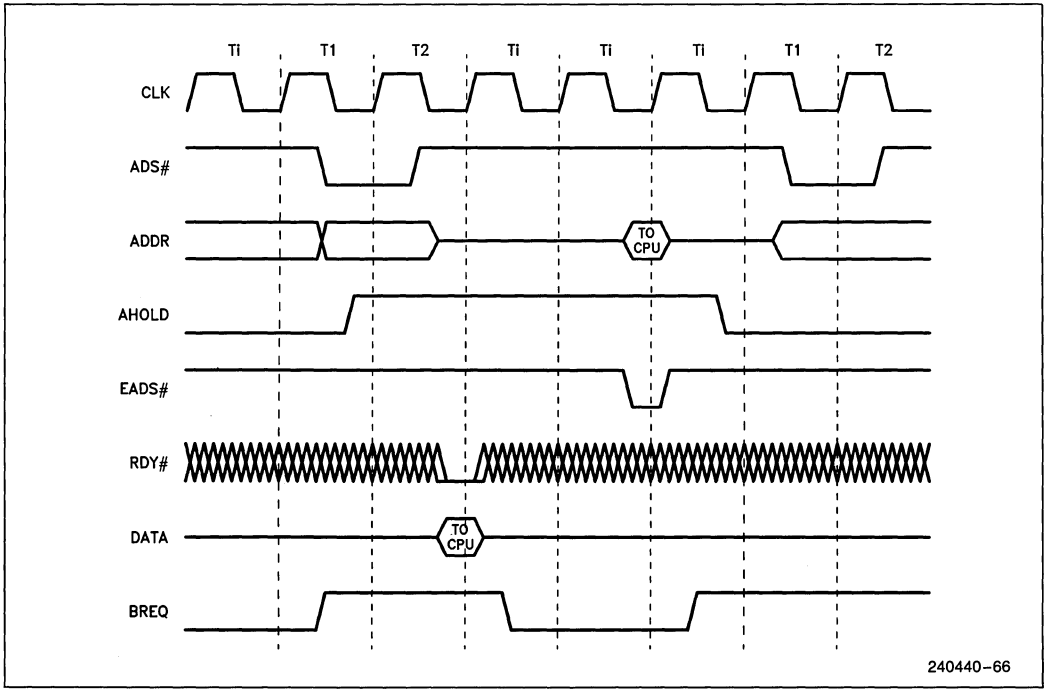


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The 486 microprocessor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the 486 microprocessor's cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the 486 microprocessor is retrieving data from the second level cache. The 486 microprocessor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the 486 microprocessor's cache.

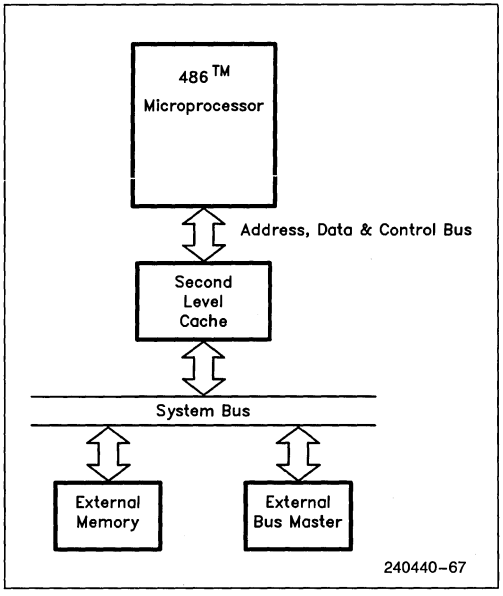
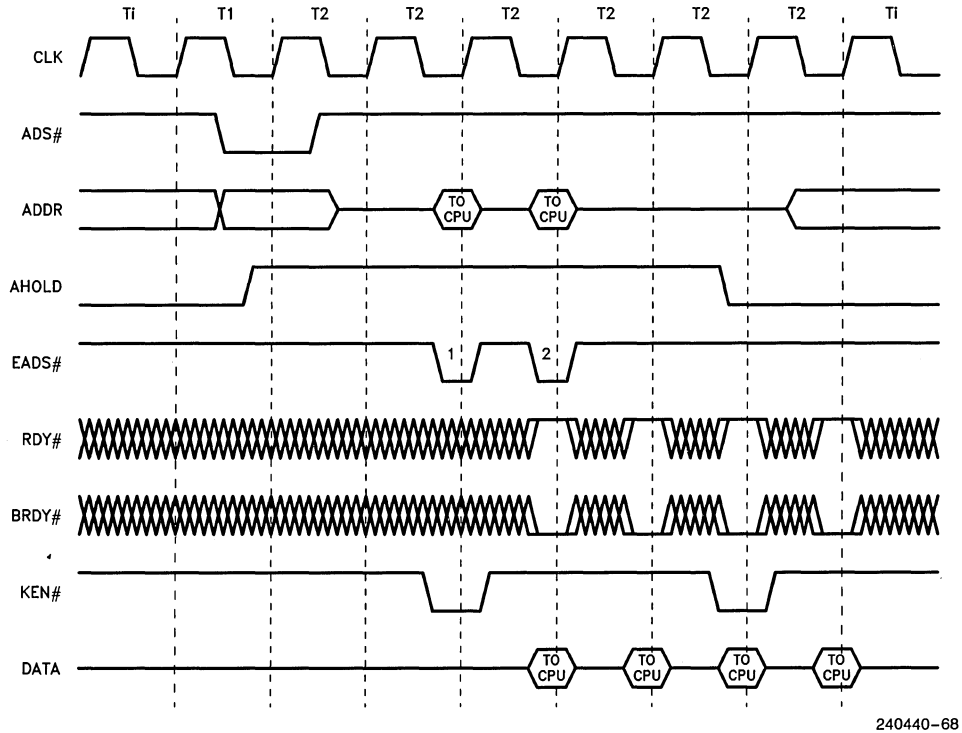


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the 486 microprocessor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the 486 microprocessor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the 486 microprocessor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the 486 microprocessors input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.

7.2.9 BUS HOLD

The 486 microprocessor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the 486 microprocessor's bus. The processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26. Unlike the 386 microprocessor, the 486 microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

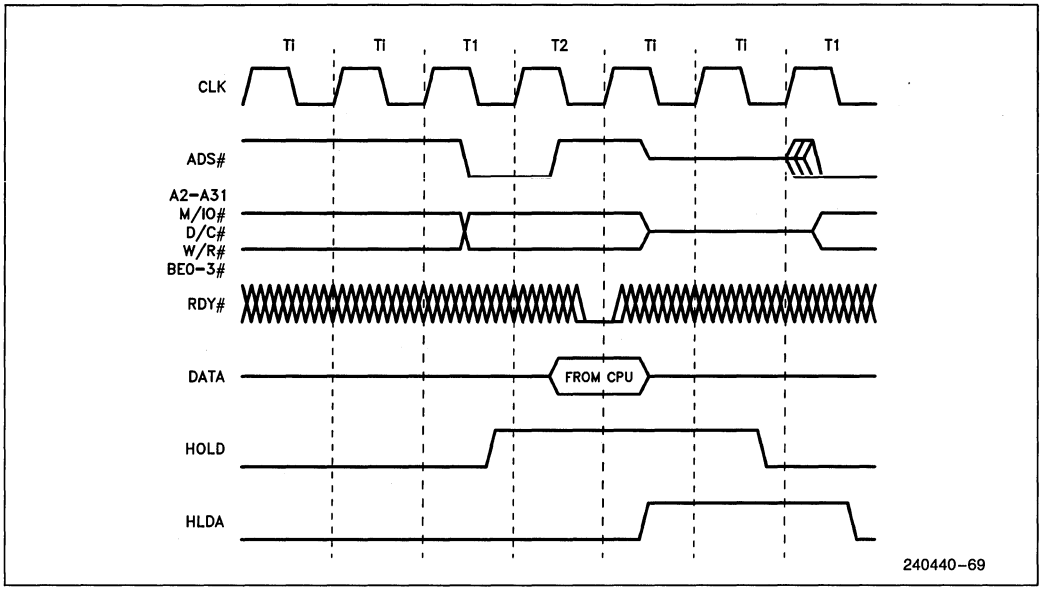


Figure 7.26. HOLD/HLDA Cycles

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The 486 microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

An example interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The 486 microprocessor has 256 possible interrupt vectors.

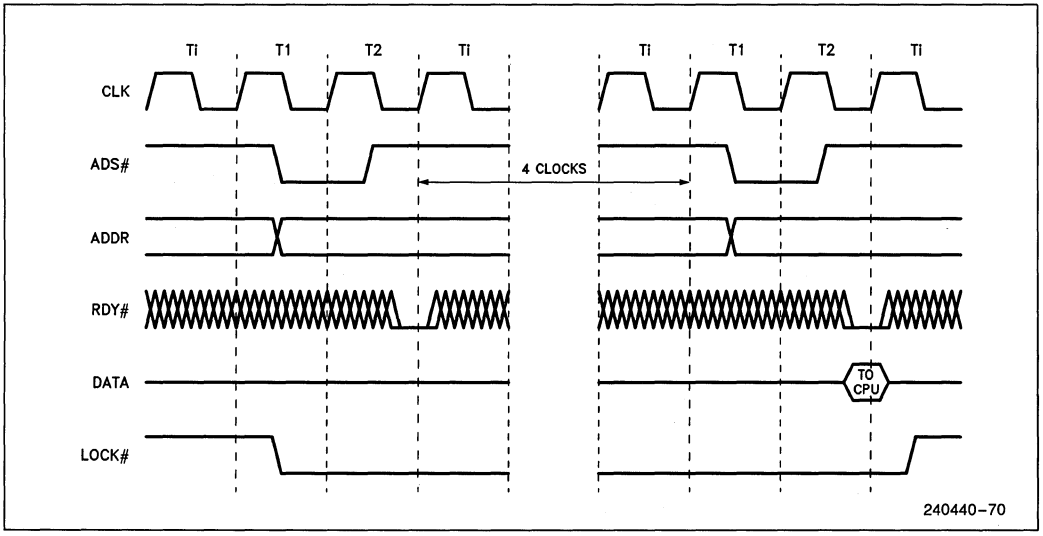


Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The 486 microprocessor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The 486 microprocessor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.8 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the 486 microprocessor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the 486 microprocessor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.8. Special Bus Cycle Encoding

BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the 486 microprocessor to complete its current bus request. In this situation the 486 microprocessor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The 486 microprocessor samples the BOFF# pin every clock. The 486 microprocessor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and

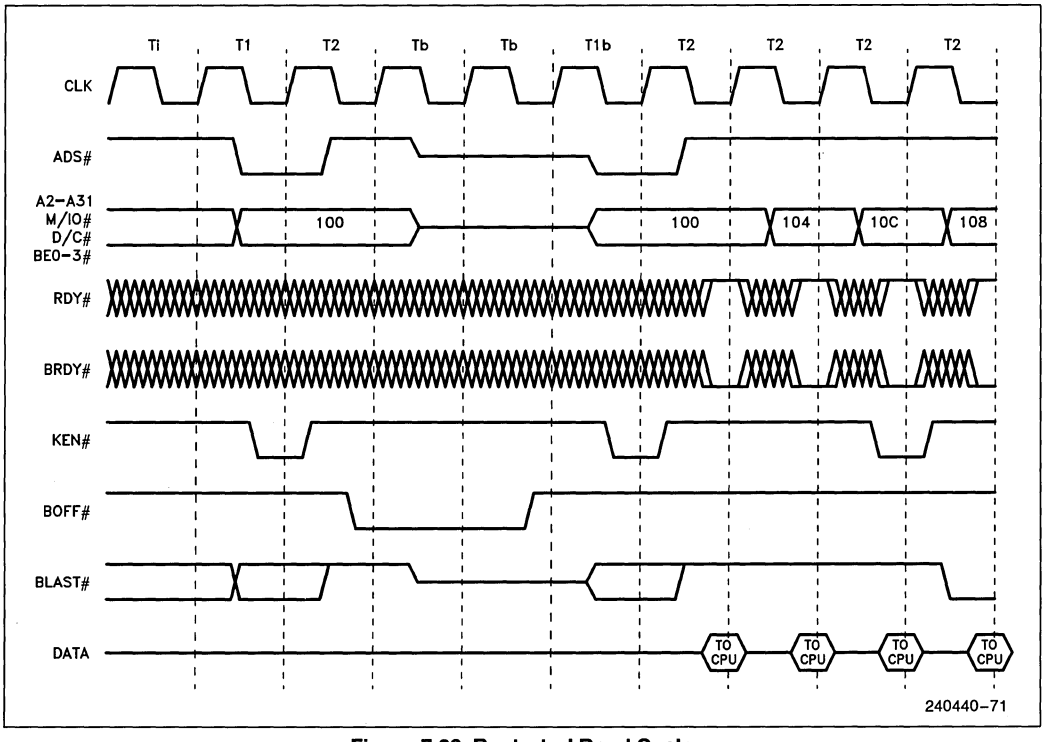


Figure 7.28. Restarted Read Cycle

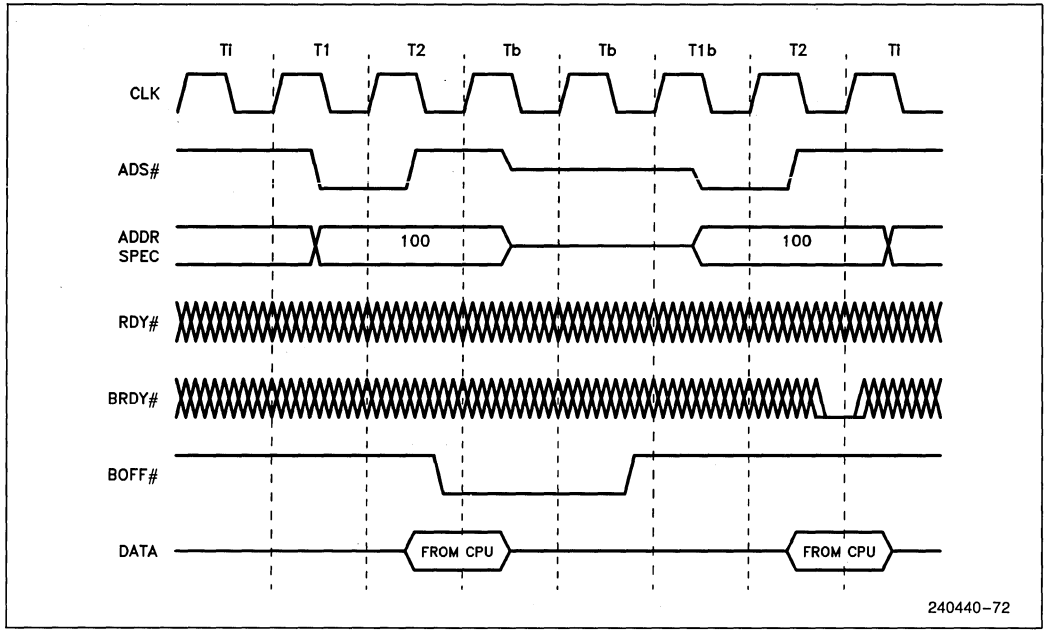


Figure 7.29. Restarted Write Cycle

any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the 486 microprocessor bus is in its high impedance state. If backoff is requested after the 486 microprocessor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the 486 microprocessor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the 486 microprocessor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the 486 microprocessor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the 486 microprocessor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. The 486 microprocessor has five bus states described in Table 7.9. A description of the signals used in the diagram is given in Table 7.10. Table 7.11 shows when the 486 microprocessor will float its address, data and status pins. The description of the bus states in this Section applies only to user bus cycles not ICE cycles.

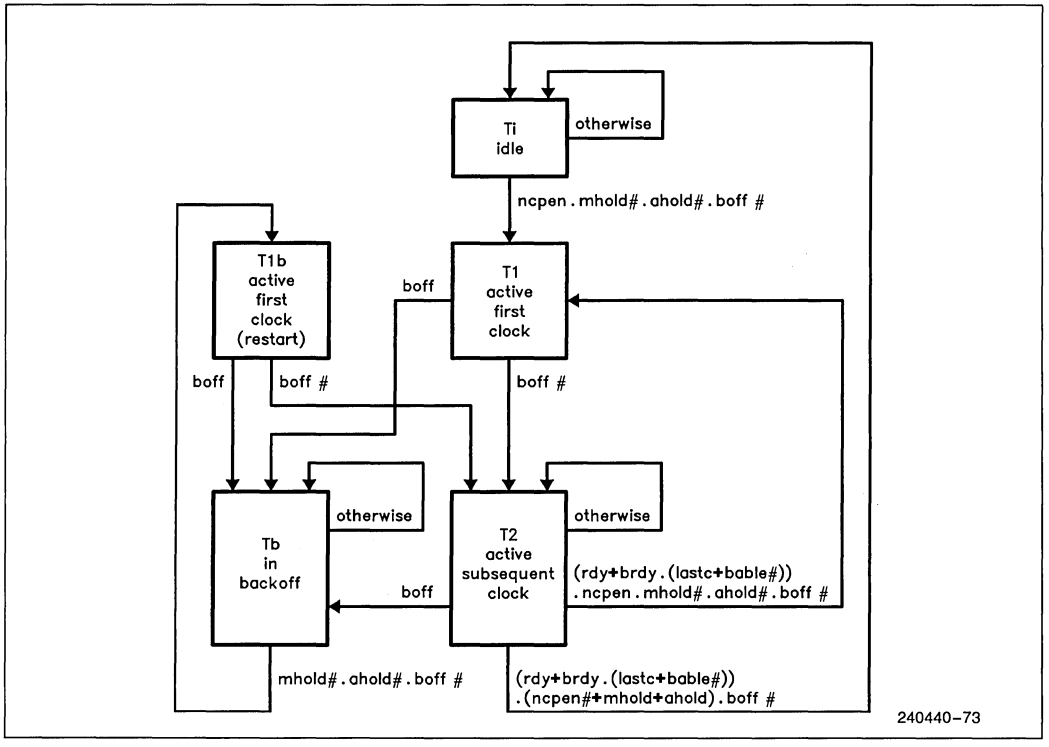


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status may be driven to undefined values, or the bus may be in high impedance state.
T1	First clock of a bus cycle. Valid address and status are driven out and ADS# is asserted.
T2	Second and subsequent clocks of a bus cycle. Data is driven out (if the cycle is a write), or data is expected (if the cycle is a read), and **RDY#** and **BRDY#** pins are sampled.
T1b	First clock of a restarted bus cycle. Valid address and status are driven out and ADS# is asserted. This state cannot be distinguished from T1 externally.
Tb	Second and subsequent clocks of aborted bus cycle.

Table 7.10. Signals used in Bus State Diagram

Signal	Means
ncpen	The 486 CPU has generated a new bus cycle internally.
mhold	The external **HOLD** pin is asserted (HIGH), and the hold qualifications are true. These qualifications include lock, unfinished cycle, last transfer in cycle, and ICE cycle.
ahold	The external **AHOLD** pin is asserted (HIGH).
boff	The external **BOFF#** pin is asserted (LOW).
rdy	The external **RDY#** pin is asserted (LOW).
brdy	The external **BRDY#** pin is asserted (LOW).
lastc	The current cycle is the last in a transfer.
bable	The current cycle is burstable.

Table 7.11. Bus Float Description

486 Action	When Condition is True
Float Address Bus in Next Clock	mhold.rdy.T2.boff# + mhold.brdy.(lastc + bable#).T2.boff# + mhold.Ti + mhold.Tb + boff + ahold
Float Bus in Next Clock	mhold.rdy.T2.boff# + mhold.brdy.(lastc + bable#).T2.boff# + mhold.Ti + mhold.Tb + boff
Assert HLDA in Next Clock	mhold.rdy.T2.boff# + mhold.brdy.(lastc + bable#).T2.boff# + mhold.Ti + mhold.Tb

7.2.14 FLOATING POINT ERROR HANDLING

The 486 microprocessor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The 486 microprocessor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The 486 microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the 387 math coprocessor.

IGNNE# is an input to the 486 microprocessor. When the NE bit in CR0 is cleared, and IGNNE# is asserted, the 486 microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the 486 microprocessor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV,

FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the 486 clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the 486 microprocessor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

8.0 TESTABILITY

Testing in the 486 microprocessor can be divided into two categories: Built-in Self Test (BIST) and external testing. The BIST tests the non-random logic, control ROM (CROM), translation lookaside buffer (TLB) and on-chip cache memory. External tests can be run on the TLB and the on-chip cache. The 486 microprocessor also has a test mode in which all outputs are tristated.

8.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) pin HIGH in the clock prior to RESET going from HIGH to LOW as shown in Figure 8.1. The BIST takes approximately 2**20 clocks, or approximately 42 milliseconds with a 25 MHz 486 microprocessor. No bus cycles will be run by the 486 microprocessor until the BIST is concluded.

The results of BIST is stored in the EAX register. The 486 microprocessor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the microprocessor.

The microprocessor performs reset and begins normal operation at the completion of the BIST. The DX register will contain a component identifier at the conclusion of reset, with or without the BIST. The upper byte of DX will contain 04 and the lower byte will contain a stepping identifier (see Figure 8.2). Refer to Section 6.5 for the register states upon reset.

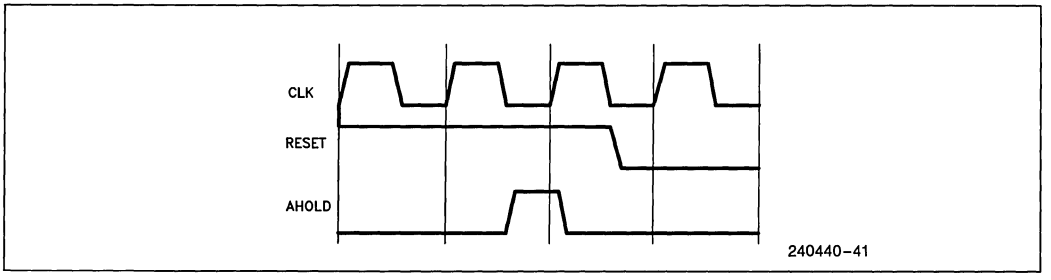


Figure 8.1. BIST Initiation

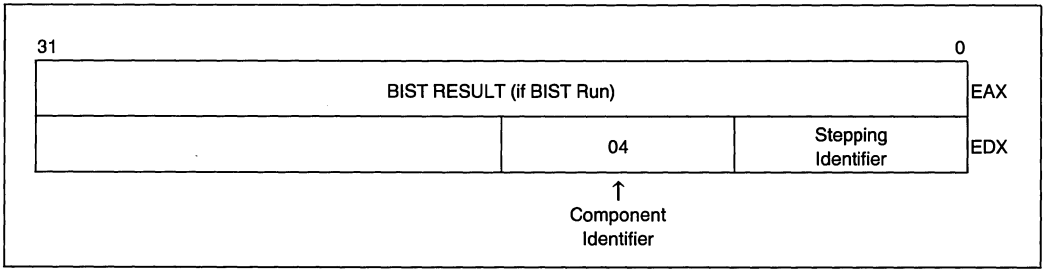


Figure 8.2. BIST Results

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache. The following sub-Sections describe the 486 microprocessor cache organization, its relation to the testability hardware and the algorithms used for accessing the cache.

8.2.1 CACHE ORGANIZATION

The cache is logically organized into three blocks shown in Figure 8.3. The cache holds 8 Kbytes of data and is 4-way set associative. Each line in the cache is 16 bytes wide. The cache contains 128 sets.

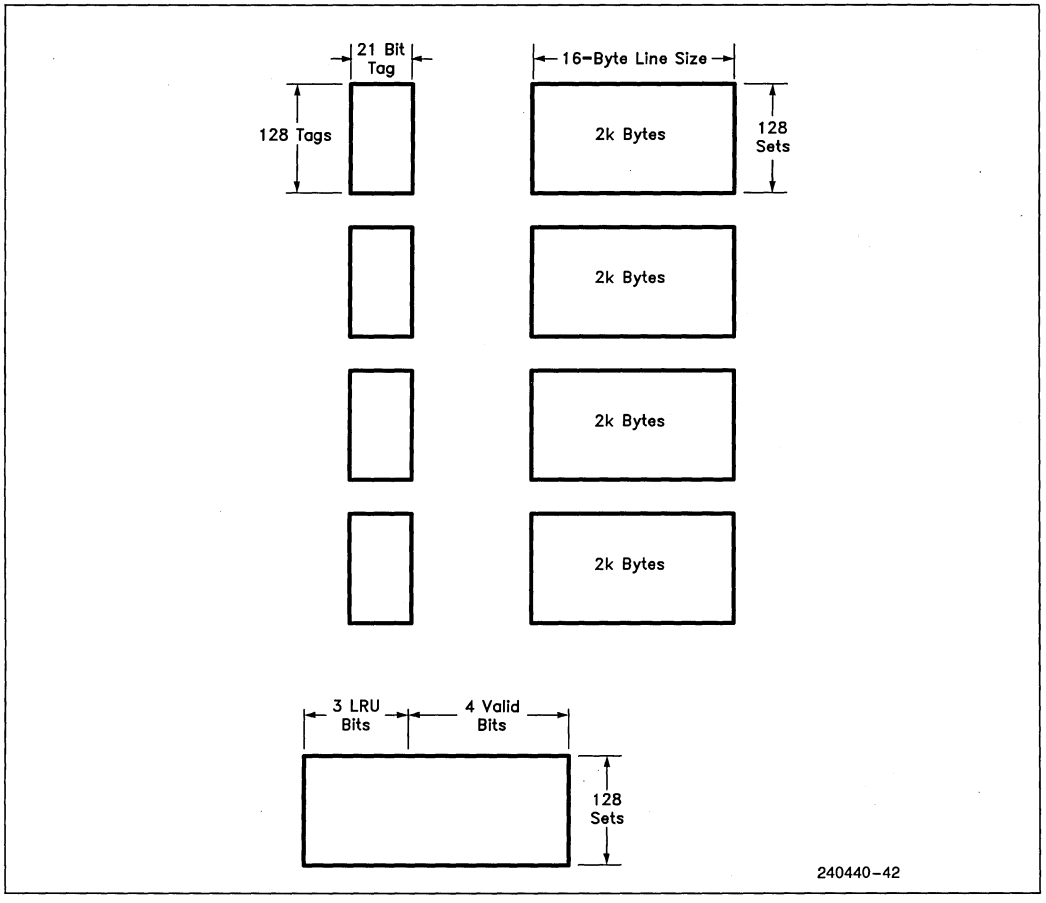


Figure 8.3. On-Chip Cache Physical Organization

The largest block is the data block. The data block is logically organized as 128 sets, each containing four 16-byte lines or entries. Addressing into the data block consists of two components, the set select and the entry select. The set select is seven bits wide since there are 128 sets. The entry select is two bits wide to select one of the four entries.

The second block is the tag array. The tag array is physically organized as 128 sets, each containing four 21-bit wide data words.

The third block consists of 128 seven-bit quantities which contain the valid bits and the bits used in the pseudo least recently used (LRU) replacement algorithm called the LRU bits. There are four valid bits, one for each line in the set. There are three LRU bits for each of the 128 sets. Refer to Section 5 for a description of the replacement algorithm.

The 486 microprocessor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

8.2.2 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.4 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

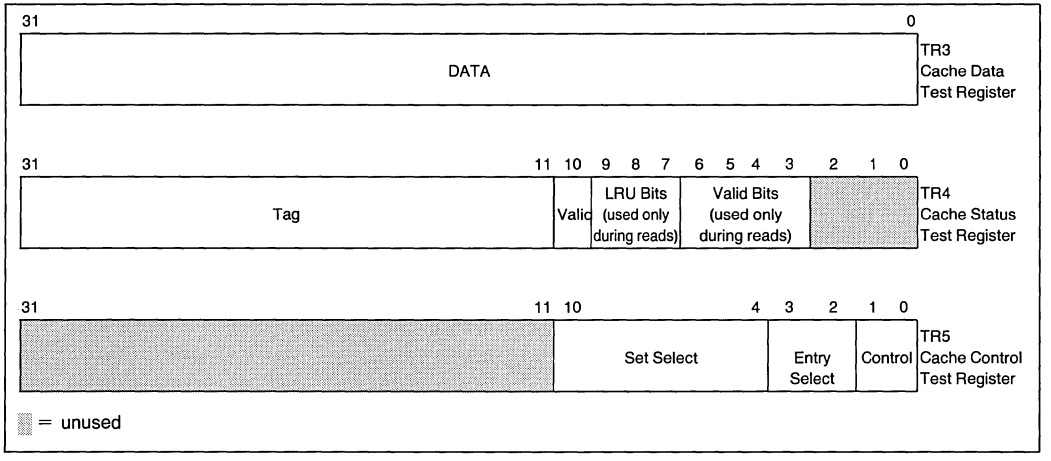


Figure 8.4. Cache Test Registers

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CE and WT bits in control register must be reset to 0 to disable the cache. See Section 5).

8.2.3 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.5.

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write / Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```
;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CR0 already says to enable reads/write to TR5
;
; fill the cache buffer
        mov esi,0         ; set up command
        mov tr5,esi       ; load to TR5
        mov tr3,eax       ; load data into cache fill buffer
        mov esi,4
        mov tr5,esi
        mov tr3,ebx
        mov esi,8
        mov tr5,esi
        mov tr3,ecx
        mov esi,0ch
        mov tr5,esi
        mov tr3,edx
;
; load the Cache Status Register
;
        mov tr4,edi       ; load 21-bit tag and valid bit
;
; perform the cache write
;
        mov esi,1
        mov tr5,esi       ; write the cache (set 0, entry 0)
```

An example assembly language sequence to perform a cache read is:

```
;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
        mov esi,2
        mov tr5,esi       ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
        mov esi,0
        mov tr5,esi
        mov eax,tr3
        mov esi,4
        mov tr5,esi
        mov ebx,tr3
        mov esi,8
        mov tr5,esi
        mov ecx,tr3
        mov esi,0ch
        mov tr5,esi
        mov edx,tr3
;
; read the status from TR4
;
        mov edi,tr4
```

Figure 8.5 Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CE bit is control register 0 is reset to 0). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM.

8.2.4 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.5.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the 486 microprocessor whether the cache is enabled or not.

8.2.7 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 Translation Lookaside Buffer (TLB) Testing

The 486 microprocessor TLB testability hooks are similar to those in the 386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the 486 microprocessor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The 486 microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.6.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

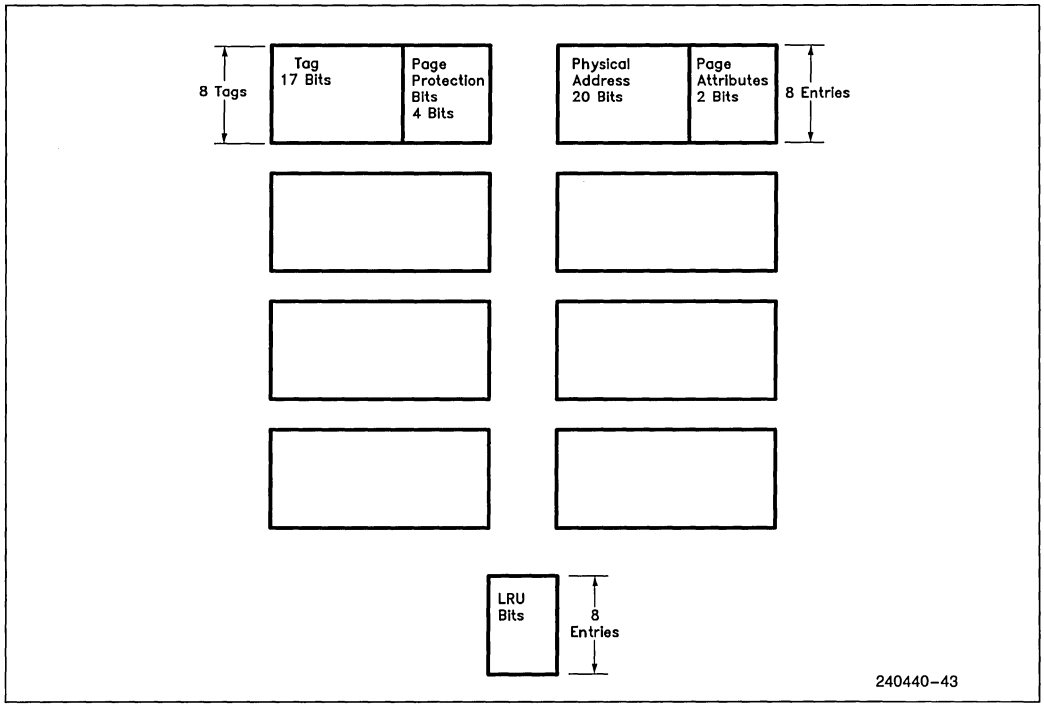


Figure 8.6. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.7. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

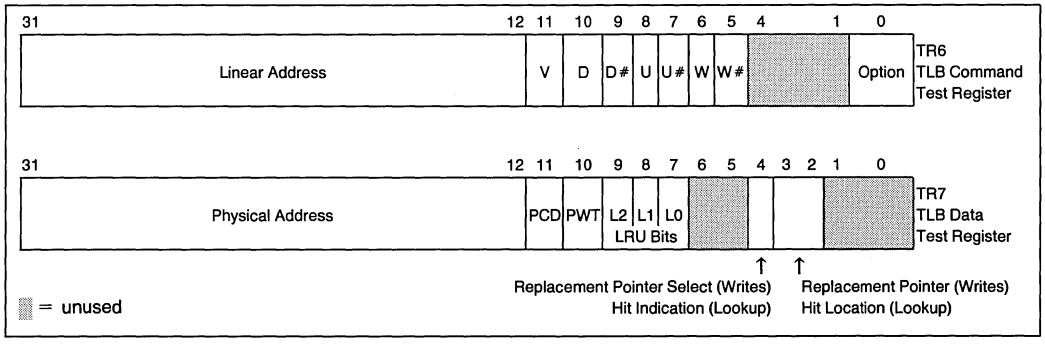


Figure 8.7. TLB Test Registers

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit# (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the 386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will not result in a hit. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 Tristate Output Test Mode

The 486 microprocessor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the 486 microprocessor is in the tristate output test mode external testing can be used to test board connections.

The tristate test mode is invoked by driving FLUSH# low in the clock prior to RESET going low (see Figure 8.8). The 486 microprocessor remains in the tristate test mode until the next RESET.

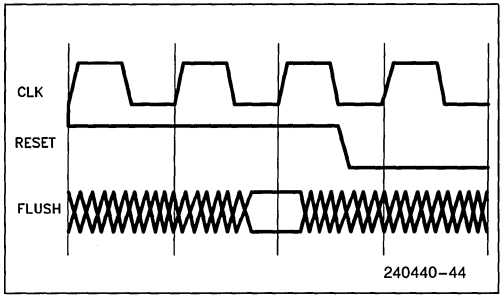


Figure 8.8. Tristate Test

9.0 DEBUGGING SUPPORT

The 486 Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1) the code execution breakpoint opcode (0CCH),
2) the single-step capability provided by the TF bit in the flag register, and
3) the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the 486 Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The 486 Microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

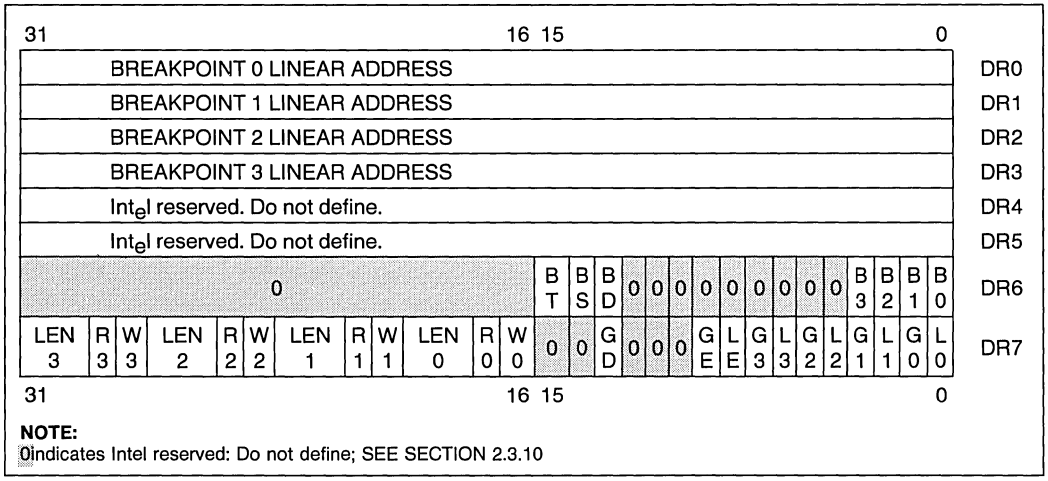
Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. 486 Microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:



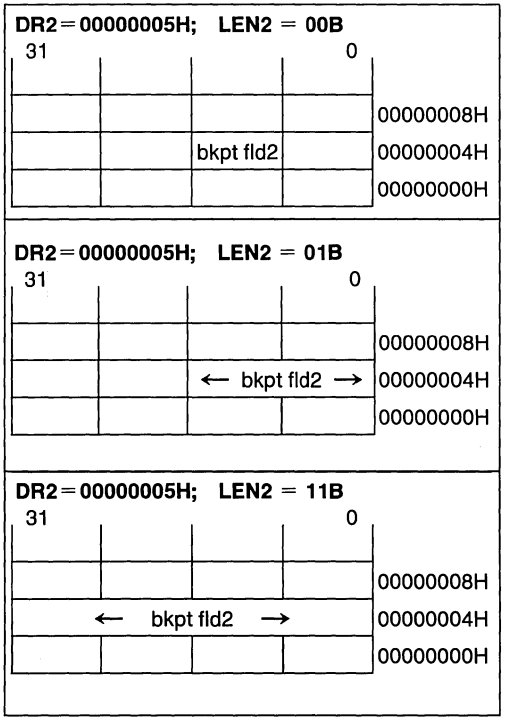
NOTE:
0indicates Intel reserved: Do not define; SEE SECTION 2.3.10

Figure 9.1. Debug Registers

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i=0−3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

If either GE or LE is set, any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the 486 Microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

If exact data breakpoint match is not selected, data breakpoints may not be reported until several instructions later or may not be reported at all. When enabling a data breakpoint, it is therefore recommended to enable the exact data breakpoint match.

When the 486 Microprocessor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The 486 Microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly, whether or not exact data breakpoint match is selected.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the 486 Microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the 486 Microprocessor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local

breakpoint registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All 486 Microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD=1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a 486 Microprocessor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the 486 microprocessor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the 486 microprocessor instructions.

10.1 i486™ Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Tables 10.1 through 10.3 by the processor clock period (e.g., 40 ns for a 25 MHz 486 microprocessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The 486 microprocessor instruction clock count tables give clock counts assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the 486 microprocessor to run an external bus cycle. The 486 microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of clocks for the second and subsequent cycles in a burst read.

w = The number of clocks for a write.

The fastest bus the 486 microprocessor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add clocks to reads until the bus is available.
2. Accesses are aligned. Add three clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or pre-fetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the clock counts shown.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, add 1 clock to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b clocks.
7. No write buffer delay. Add w clocks (on average) to wait for 1 write buffer to empty.
8. Displacement and immediate not used together. If displacement and immediate used together add 1 clock to the clock count shown.
9. No invalidate cycles. Add a delay of 1 clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the 486 CPU needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 clocks to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) clocks to the cache miss penalty.

Table 10.1. i486™ Microprocessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W | 11 reg1 reg2	1		
reg2 to reg1	1000101w | 11 reg2 reg1	1		
memory to reg	1000101w | mod reg r/m	1	2	
reg to memory	1000100w | mod reg r/m	1		
Immediate to reg2	1100011w | 11000 reg2 | immediate data	1		
or	1011w reg2 | immediate data	1		
Immediate to Memory	1100011w | mod 000 r/m | displacement immediate	1		
Memory to Accumulator	1010000w | full displacement	1	2	
Accumulator to Memory	1010001w | full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 | 1011z11w | 11 reg1 reg2	3		
memory to reg1	00001111 | 1011z11w | mod reg r/m	3	2	
PUSH = Push				
reg2	11111111 | 11 110 reg2	4		
or	01010 reg2	1		
memory	11111111 | mod 110 r/m	4	1	1
immediate	011010s0 | immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg2	10001111 | 11 000 r/m	4	1	
or	01011 reg2	1	2	
memory	10001111 | mod 000 r/m	6	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w | 11 reg1 reg2	3		2
Accumulator with reg2	10010 reg2	3		2
Memory with reg	1000011w | mod reg r/m	5		2
LEA = Load EA to Register	10001101 | mod reg r/m			
no index register		1		
with index register		2		

z	instruction
0	MOVZX
1	MOVSX

Instruction	TTT
ADD = Add	000
ADC = Add with Carry	010
AND = Logical AND	100
OR = Logical OR	001
SUB = Subtract	101
SBB = Subtract with Borrow	011
XOR = Logical Exclusive OR	110

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
reg1 to reg2	00TTT00w 11 reg1 reg2	1		
reg2 to reg1	00TTT01w 11 reg1 reg2	1		
memory to register	00TTT01w mod reg r/m	2	2	
register to memory	00TTT00w mod reg r/m	3	6/2	U/L
immediate to register	100000sw 11 TTT reg immediate register	1		
immediate to accumulator	00TTT10w immediate data	1		
immediate to memory	100000sw mod TTT r/m immediate data	3	6/2	U/L
Instruction / **TTT**: INC = Increment 000; DEC = Decrement 001				
reg2	1111111w 11 TTT reg2	1		
or	01TTT reg2	1		
memory	1111111w mod TTT reg2	3	6/2	U/L
Instruction / **TTT**: NOT = Logical Complement 010; NEG = Negate 011				
reg2	1111011w 11 TTT reg2	1		
memory	1111011w mod TTT r/m	3	6/2	U/L
CMP = Compare				
reg1 with reg2	0011100w 11 reg1 reg2	1		
reg2 with reg1	0011101w 11 reg1 reg2	1		
memory with register	0011100w mod reg r/m	2	2	
register with memory	0011101w mod reg r/m	2	2	
immediate with register	100000sw 11 111 reg immediate data	1		
immediate with acc.	0011110w immediate data	1		
immediate with memory	100000sw mod 111 r/m immediate data	2	2	
TEST = Logical Compare				
reg1 and reg2	1000010w 11 reg1 reg2	1		
memory and register	1000010w mod reg r/m	2	2	
immediate and register	1111011w 11 000 reg immediate data	1		
immediate and acc.	1010100w immediate data	1		
immediate and memory	1111011w mod 000 r/m immediate data	2	2	
MUL = Multiply (unsigned)				
acc. with register	1111011w 11 100 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
IMUL = Integer Multiply (signed)				
acc. with register	1111011w | 11 101 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 101 r/m			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
reg1 with reg2	00001111 | 10101111 | 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
register with memory	00001111 | 10101111 | mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
reg1 with imm. to reg2	011010s1 | 11 reg1 reg2 | immediate data			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
mem. with imm. to reg.	011010s1 | mod reg r/m | immediate data			
Multiplier-Byte		13/18	2	MN/MX, 3
Word		13/26	2	MN/MX, 3
Dword		13/42	2	MN/MX, 3
DIV = Divide (unsigned)				
acc. by register	1111011w | 11 110 reg			
Divisor-Byte		16		
Word		24		
Dword		40		
acc. by memory	1111011w | mod 110 r/m			
Divisor-Byte		16		
Word		24		
Dword		40		
IDIV = Integer Divide (signed)				
acc. by register	1111011w | 11 111 reg			
Divisor-Byte		19		
Word		27		
Dword		43		
acc. by memory	1111011w | mod 111 r/m			
Divisor-Byte		20		
Word		28		
Dword		44		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
CBW = Convert Byte to Word	10011000	3		
CWD = Convert Word to Dword	10011001	3		
	Instruction / **TTT**: ROL = Rotate Left 000; ROR = Rotate Right 001; RCL = Rotate through Carry Left 010; RCR = Rotate through Carry Right 011; SHL/SAL = Shift Logical/Arithmetic Left 100; SHR = Shift Logical Right 101; SAR = Shift Arithmetic Right 111			
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)				
reg by 1	1101000w | 11 TTT reg	3		
memory by 1	1101000w | mod TTT r/m	4	6	
reg by CL	1101001w | mod TTT reg	3		
memory by CL	1101001w | mod TTT r/m	4	6	
reg by immediate count	1100000w | 11 TTT reg | immediate 8-bit data	2		
mem by immediate count	1100000w | mod TTT r/m | immediate 8-bit data	4	6	
Through Carry (RCL and RCR)				
reg by 1	1101000w | 11 TTT reg	3		
memory by 1	1101000w | mod TTT r/m	4	6	
reg by CL	1101001w | 11 TTT reg	8/30		MN/MX, 4
memory by CL	1101001w | mod TTT r/m	9/31		MN/MX, 5
reg by immediate count	1100000w | 11 TTT reg | immediate 8-bit data	8/30		MN/MX, 4
mem by immediate count	1100000w | mod TTT r/m | immediate 8-bit data	9/31		MN/MX, 5
	Instruction / **TTT**: SHLD = Shift Left Double 100; SHRD = Shift Right Double 101			
register with immediate	00001111 | 10TTT100 | 11 reg r/m | imm 8-bit data	2		
memory by immediate	00001111 | 10TTT100 | mod reg r/m | imm 8-bit data	3	6	
register by CL	00001111 | 10TTT101 | 11 reg r/m	3		
memory by CL	00001111 | 10TTT101 | mod reg r/m	4	5	
BSWAP = Byte Swap	00001111 | 11001 reg	1		
XADD = Exchange and Add				
reg1, reg2	00001111 | 1100000w | 11 reg2 reg1	3		
memory, reg2	00001111 | 1100000w | mod reg2 mem	4	6/2	U/L
CMPXCHG = Compare and Exchange				
reg1, reg2	00001111 | 1010011w | 11 reg2 reg1	6		
memory, reg2	00001111 | 1010011w | mod reg2 mem	7/10	2	6
NOP = No Operation	10010000	3		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment)				
NOTE: Times are jump taken/not taken				
Jccc = Jump on ccc				
8-bit displacement	0111tttn | 8-bit disp.	3/1		T/NT, 23
full displacement	00001111 | 1000tttn | full displacement	3/1		T/NT, 23
NOTE: Times are jump taken/not taken				
SETcccc = Set Byte on cccc (Times are cccc true/false)				
reg2	00001111 | 1001tttn | 11 000 reg2	4/3		
memory	00001111 | 1001tttn | mod 000 r/m	3/4		
LOOP = LOOP CX Times	11100010 | 8-bit disp.	7/6		L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit disp.	9/6		L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	11100000 | 8-bit disp.	9/6		L/NL, 23
JCXZ = Jump on CX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
JECXZ = Jump on ECX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)				
JMP = Unconditional Jump (within segment)				
Short	11101011 | 8-bit disp.	3		7, 23
Direct	11101001 | full displacement	3		7, 23
Register Indirect	11111111 | 11 100 reg	5		7, 23
Memory Indirect	11111111 | mod 100 r/m	5	5	7
CALL = Call (within segment)				
Direct	11101000 | full displacement	3		7, 23
Register Indirect	11111111 | 11 010 reg	5		7, 23
Memory Indirect	11111111 | mod 010 r/m	5	5	7
RET = Return from CALL (within segment)				
	11000011	5	5	
Adding Immediate to SP	11000010 | 16-bit disp.	5	5	

Mnemonic cccc	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment) (Continued)				
ENTER = Enter Procedure	11001000 | 16-bit disp., 8-bit level			
Level = 0		14		
Level = 1		17		
Level (L) > 1		17+3L		8
LEAVE = Leave Procedure	11001001	5	1	
MULTIPLE-SEGMENT INSTRUCTIONS				
MOV = Move				
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	0/3	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	2/5	RV/P, 9
segment reg. to reg.	10001100 | 11 sreg3 reg	3		
segment reg. to memory	10001100 | mod sreg3 r/m	3		
PUSH = Push				
segment reg. (ES, CS, SS, or DS)	000 sreg2 110	3		
segment reg. (FS or GS)	00001111 | 10 sreg3 000	3		
POP = Pop				
segment reg. (ES, SS, or DS)	000 sreg2 111	3/9	2/5	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3 001	3/9	2/5	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	7/10	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	7/10	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	7/10	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	7/10	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	7/10	RV/P, 9
CALL = Call				
Direct intersegment	10011010 | unsigned full offset, selector	18	2	R, 7, 22
to same level		20	3	P, 9
thru Gate to same level		35	6	P, 9
to inner level, no parameters		69	17	P, 9
to inner level, x parameter (d) words		77+4X	17+n	P, 11, 9
to TSS		37+TS	3	P, 10, 9
thru Task Gate		38+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	8	R, 7
to same level		20	10	P, 9
thru Gate to same level		35	13	P, 9
to inner level, no parameters		69	24	P, 9
to inner level, x parameter (d) words		77+4X	24+n	P, 11, 9
to TSS		37+TS	10	P, 10, 9
thru Task Gate		38+TS	10	P, 10, 9
RET = Return from CALL				
intersegment	11001011	13	8	R, 7
to same level		18	9	P, 9
to outer level		33	12	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.	14	8	R, 7
to same level		17	9	P, 9
to outer level		33	12	P, 9

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)				
JMP = Unconditional Jump				
Direct intersegment	11101010 unsigned full offset, selector	17	2	R, 7, 22
to same level		19	3	P, 9
thru Call Gate to same level		32	6	P, 9
thru TSS		42+TS	3	P, 10, 9
thru Task Gate		43+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	9	R, 7, 9
to same level		18	10	P, 9
thru Call Gate to same level		31	13	P, 9
thru TSS		41+TS	10	P, 10, 9
thru Task Gate		42+TS	10	P, 10, 9
BIT MANIPULATION				
BT = Test bit				
register, immediate	00001111 | 10111010 | 11 100 reg imm. 8-bit data	3		
memory, immediate	00001111 | 10111010 | mod 100 r/m imm. 8-bit data	3	1	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3		
memory, reg	00001111 | 10100011 | mod reg r/m	8	2	
Instruction / **TTT**: BTS = Test Bit and Set: 101; BTR = Test Bit and Reset: 110; BTC = Test Bit and Compliment: 111				
register, immediate	00001111 | 10111010 | 11 TTT reg imm. 8-bit data	6		
memory, immediate	00001111 | 10111010 | mod TTT r/m imm. 8-bit data	8	2/0	U/L
reg1, reg2	00001111 | 10TTT011 | 11 reg2 reg1	6		
memory, reg	00001111 | 10TTT011 | mod reg r/m	13	3/1	U/L
BSF = Scan Bit Forward				
reg1, reg2	00001111 | 10111100 | mod reg2 reg1	6/42		MN/MX, 12
memory, reg	00001111 | 10111100 | mod reg r/m	7/43	2	MN/MX, 13
BSR = Scan Bit Reverse				
reg1, reg2	00001111 | 10111101 | mod reg2 reg1	6/103		MN/MX, 14
memory, reg	00001111 | 10111101 | mod reg r/m	7/104	1	MN/MX, 15
STRING INSTRUCTIONS				
CMPS = Compare Byte Word	1010011w	8	6	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	2	
MOVS = Move Byte/Word	1010010w	7	2	16
SCAS = Scan Byte/Word	1010111w	6	2	
STOS = Store Byte/Word from AL/AX/EX	1010101w	5		
XLAT = Translate String	11010111	4	2	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
REPEATED STRING INSTRUCTIONS				
Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REPNE CMPS = Compare String (Find Match)	11110010 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REP LODS = Load String	11110010 1010110w			
C = 0		5		
C > 0		7+4c		16, 18
REP MOVS = Move String	11110010 1010010w			
C = 0		5		
C = 1		13	1	16
C > 1		12+3c		16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011 1010111w			
C = 0		5		
C > 0		7+5c		20
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010 1010111w			
C = 0		5		
C > 0		7+5c		20
REP STOS = Store String	11110010 1010101w			
C = 0		5		
C > 0		7+4c		
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
STC = Set Carry Flag	11111001	2		
CMC = Complement Carry Flag	11110101	2		
CLD = Clear Direction Flag	11111100	2		
STD = Set Direction Flag	11111101	2		
CLI = Clear Interrupt Enable Flag	11111010	5		
STI = Set Interrupt Enable Flag	11111011	5		
LAHF = Load AH into Flag	10011111	3		
SAHF = Store AH into Flags	10011110	2		
PUSHF = Push Flags	10011100	4/3		RV/P
POPF = Pop Flags	10011101	9/6		RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust for Add	00110111	3		
AAS = ASCII Adjust for Subtract	00111111	3		
AAM = ASCII Adjust for Multiply	11010100 00001010	15		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
DECIMAL ARITHMETIC (Continued)				
AAD = ASCII Adjust for Divide	11010101 | 00001010	14		
DAA = Decimal Adjust for Add	00100111	2		
DAS = Decimal Adjust for Subtract	00101111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	11110100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	00001111 | 00100010 | 11 000 reg	16	2	
CR2/CR3 from register	00001111 | 00100010 | 11 eee reg	4		
Reg from CR0–3	00001111 | 00100000 | 11 eee reg	4		
DR0–3 from register	00001111 | 00100011 | 11 eee reg	11		
DR6–7 from register	00001111 | 00100011 | 11 eee reg	11		
Register from DR6–7	00001111 | 00100001 | 11 eee reg	10		
Register from DR0–3	00001111 | 00100001 | 11 eee reg	10		
TR3 from register	00001111 | 00100110 | 11 011 reg	6		
TR4–7 from register	00001111 | 00100110 | 11 eee reg	4		
Register from TR3	00001111 | 00100100 | 11 011 reg	3		
Register from TR4–7	00001111 | 00100100 | 11 eee reg	4		
CLTS = Clear Task Switched Flag	00001111 | 00000110	7	2	
INVD = Invalidate Data Cache	00001111 | 00001000	4		
WBINVD = Write-Back and Invalidate Data Cache	00001111 | 00001001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	00001111 | 00000001 | mod 11 mem	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	01100111	1		
LOCK = Bus Lock Prefix	11110000	1		
Operand Size Prefix	01100110	1		
Segment Override Prefix				
CS:	00101110	1		
DS:	00111110	1		
ES:	00100110	1		
FS:	01100100	1		
GS:	01100101	1		
SS:	00110110	1		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	01100011 | 11 reg r/m	9		
From memory	01100011 | mod reg r/m	9		
LAR = Load Access Rights				
From register	00001111 | 00000010 | 11 reg r/m	11	3	
From memory	00001111 | 00000010 | mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	00001111 | 00000001 | mod 010 r/m	11	5	
LIDT = Load Interrupt Descriptor				
Table register	00001111 | 00000001 | mod 011 r/m	11	5	
LLDT = Load Local Descriptor				
Table register from reg.	00001111 | 00000000 | 11 010 reg	11	3	
Table register from mem.	00001111 | 00000000 | mod 010 r/m	11	6	
LMSW = Load Machine Status Word				
From register	00001111 | 00000001 | 11 110 reg	13		
From memory	00001111 | 00000001 | mod 110 r/m	13	1	
LSL = Load Segment Limit				
From register	00001111 | 00000011 | 11 reg r/m	10	3	
From memory	00001111 | 00000011 | mod reg r/m	10	6	
LTR = Load Task Register				
From Register	00001111 | 00000000 | 11 001 reg	20		
From Memory	00001111 | 00000000 | mod 001 r/m	20		
SGDT = Store Global Descriptor Table				
	00001111 | 00000001 | mod 000 r/m	10		
SIDT = Store Interrupt Descriptor Table				
	00001111 | 00000001 | mod 001 r/m	10		
SLDT = Store Local Descriptor Table				
To register	00001111 | 00000000 | 11 000 reg	2		
To memory	00001111 | 00000000 | mod 000 r/m	3		
SMSW = Store Machine Status Word				
To register	00001111 | 00000000 | mod 100 reg	2		
To memory	00001111 | 00000000 | mod 100 r/m	3		
STR = Store Task Register				
To register	00001111 | 00000000 | 11 001 reg	2		
To memory	00001111 | 00000000 | mod 001 r/m	3		
VERR = Verify Read Access				
Register	00001111 | 00000000 | 11 100 r/m	11	3	
Memory	00001111 | 00000000 | mod 100 r/m	11	7	
VERW = Verify Write Access				
To register	00001111 | 00000000 | mod 101 reg	11	3	
To memory	00001111 | 00000000 | mod 101 r/m	11	7	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty If Cache Miss	Notes
INTERRUPT INSTRUCTIONS				
INT n = Interrupt Type n	11001101 | type	INT+4/0		RV/P, 21
INT 3 = Interrupt Type 3	11001100	INT+0		21
INTO = Interrupt 4 if Overflow Flag Set	11001110			
Taken		INT+2		21
Not Taken		3		21
BOUND = Interrupt 5 if Detect Value Out Range	01100010 | mod reg r/m			
If in range		7	7	21
If out of range		INT+24	7	21
IRET = Interrupt Return	11001111			
Real Mode/Virtual Mode		15	8	
Protected Mode				
To same level		20	11	9
To outer level		36	19	9
To nested task (EFLAGS.NT = 1)		TS+32	4	9, 10
External Interrupt		INT+11		21
NMI = Non-Maskable Interrupt		INT+3		21
Page Fault		INT+24		21
VM86 Exceptions				
CLI		INT+8		21
STI		INT+8		21
INT n		INT+9		
PUSHF		INT+9		21
POPF		INT+8		21
IRET		INT+9		
IN				
Fixed Port		INT+50		21
Variable Port		INT+51		21
OUT				
Fixed Port		INT+50		21
Variable Port		INT+51		21
INS		INT+50		21
OUTS		INT+50		21
REP INS		INT+51		21
REP OUTS		INT+51		21

Task Switch Clock Counts Table		
Method	**Value for TS**	
	Cache Hit	**Miss Penalty**
VM/486 CPU/286 TSS To 486 CPU TSS	162	55
VM/486 CPU/286 TSS To 286 TSS	143	31
VM/486 CPU/286 TSS To VM TSS	140	37

Interrupt Clock Counts Table			
Method	**Value for INT**		
	Cache Hit	**Miss Penalty**	**Notes**
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task gate	37 + TS	3	10

Abbreviations	**Definition**
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:
1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2(|m|)$,n)
   m = multiplier value (min clocks for m=0)
   n = 3/5 for ±m
4. Clocks = {quotient(count/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {quotient(count/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock).
7. Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12–13: (b = 0–3, non-zero byte number);
(i = 0–1, non-zero nibble number);
(n = 0–3, non bit number in nibble);

12. Clocks = 8+4 (b+1) + 3(i+1) + 3(n+1)
    = 6 if second operand = 0
13. Clocks = 9+4(b+1) + 3(i+1) + 3(n+1)
    = 7 if second operand = 0

For notes 14–15: (n = bit position 0–31)

14. Clocks = 7 + 3(32−n)
    6 if second operand = 0
15. Clocks = 8 + 3(32−n)
    7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved.
    (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned.
    (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.

Table 10.2. i486™ Microprocessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1 1 1 0 0 1 0 w | port number	14	9	29	27	
Variable Port	1 1 1 0 1 1 0 w	14	8	28	27	
OUT = Output to:						
Fixed Port	1 1 1 0 0 1 1 w | port number	16	11	31	29	
Variable Port	1 1 1 0 1 1 1 w	16	10	30	29	
INS = Input Byte/Word from DX Port	0 1 1 0 1 1 0 w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0 1 1 0 1 1 1 w	17	10	32	30	1
REP INS = Input String	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 0 w	16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	1 1 1 1 0 0 1 0 | 0 1 1 0 1 1 1 w	17+5c	11+5c	31+5c	30+5c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss Avg (Lower Range ... Upper Range)	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
DATA TRANSFER					
FLD = Real Load to ST(0)					
32-bit memory	11011 001 | mod 000 r/m | s-i-b/disp.	3	2		
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	3		
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	4		
ST(i)	11011 001 | 11000 ST(i)	4			
FILD = Integer Load to ST(0)					
16-bit memory	11011 111 | mod 000 r/m | s-i-b/disp.	14.5(13–16)	2	4	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	2	4(2–4)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	3	7.8(2–8)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	4	7.7(2–8)	
FST = Store Real from ST(0)					
32-bit memory	11011 001 | mod 010 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 010 r/m | s-i-b/disp.	8			2
ST(i)	11011 101 | 11010 ST(i)	3			
FSTP = Store Real from ST(0) and Pop					
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 011 r/m | s-i-b/disp.	8			2
80-bit memory	11011 011 | mod 111 r/m | s-i-b/disp.	6			
ST(i)	11011 101 | 11001 ST(i)	3			
FIST = Store Integer from ST(0)					
16-bit memory	11011 111 | mod 010 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 010 r/m | s-i-b/disp.	32.4(28–34)			
FISTP = Store Integer from ST(0) and Pop					
16-bit memory	11011 111 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
64-bit memory	11011 111 | mod 111 r/m | s-i-b/disp.	33.4(29–34)			
FBSTP = Store BCD from ST(0) and Pop	11011 111 | mod 110 r/m | s-i-b/disp.	175(172–176)			
FXCH = Exchange ST(0) and ST(i)	11011 001 | 11001 ST(i)	4			
COMPARISON INSTRUCTIONS					
FCOM = Compare ST(0) with Real					
32-bit memory	11011 000 | mod 010 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11010 ST(i)	4		1	
FCOMP = Compare ST(0) with Real and Pop					
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11011 ST(i)	4		1	

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range... Upper Range)	Penalty If Cache Miss Avg (Lower Range... Upper Range)	Concurrent Execution Avg (Lower Range... Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)					
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FTST = Compare ST(0) with 0.0	11011 001 | 1110 0100				
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 101 | 11101 1001	5		1	
FXAM = Examine ST(0)	11011 001 | 1110 0101	8			
CONSTANTS					
FLDZ = Load +0.0 into ST(0)	11011 001 | 1110 1110	4			
FLD1 = Load +1.0 into ST(0)	11011 001 | 1110 1000	4			
FLDPI = Load π into ST(0)	11011 001 | 1110 1011	8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8		2	
ARITHMETIC					
FADD = Add Real with ST(0)					
ST(0) ← ST(0) + 32-bit memory	11011 000 | mod 000 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(0) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)		7(5–17)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)		7(5–17)	
FSUB = Subtract real from ST(0)					
ST(0) ← ST(0) − 32-bit memory	11011 000 | mod 100 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	3	7(5–17)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 11101 ST(i)	10(8–20)		7(5–17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)		7(5–17)	

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range... Upper Range)	Penalty if Cache Miss Avg (Lower Range... Upper Range)	Concurrent Execution Avg (Lower Range... Upper Range)	Notes
ARITHMETIC (Continued)					
FSUBR = Subtract real reversed (Subtract ST(0) from real)					
ST(0) ← 32-bit memory − ST(0)	11011 000 | mod 101 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 11100 ST(i)	10(8–20)		7(5–17)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)		7(5–17)	
FMUL = Multiply real with ST(0)					
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	2	8	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	3	11	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16		13	
FDIV = Divide ST(0) by Real					
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 11111 ST(i)	73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73		70	3
FDIVR = Divide real reversed (Real/ST(0))					
ST(0) ← 32-bit memory/ST(0)	11011 000 | mod 111 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 11110 ST(i)	73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73		70	3
FIADD = Add Integer to ST(0)					
ST(0) ← ST(0) + 16-bit memory	11011 110 | mod 000 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUB = Subtract Integer from ST(0)					
ST(0) ← ST(0) − 16-bit memory	11011 110 | mod 100 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUBR = Integer Subtract Reversed					
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FIMUL = Multiply Integer with ST(0)					
ST(0) ← ST(0) × 16-bit memory	11011 110 | mod 001 r/m | s-i-b/disp.	25(23–27)	2	8	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	2	8	
FIDIV = Integer Divide					
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	2	70	3

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range... Upper Range)	Penalty if Cache Miss Avg (Lower Range... Upper Range)	Concurrent Execution Avg (Lower Range... Upper Range)	Notes
ARITHMETIC (Continued)					
FIDIVR = Integer Divide Reversed					
ST(0) ← 16-bit memory/ST(0)	11011 110 | mod 111 r/m | s-i-b/disp.	87(85-89)	2	70	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84-86)	2	70	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83-87)		70	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30-32)		2	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16-20)		4(2-4)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70-138)		2(2-8)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72-167)		5.5(2-18)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21-30)		7.4(2-8)	
FABS = Absolute value of ST(0)	11011 001 | 1110 0001	3			
FCHS = Change sign of ST(0)	11011 001 | 1110 0000	6			
TRANSCENDENTAL					
FCOS = Cosine of ST(0)	11011 001 | 1111 1111	313(257-354)		2	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200-273)		70	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218-303)		5(2-17)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	313(257-354)		2	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	336(292-365)		2	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140-279)		2	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196-329)		13	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171-326)		13	6
PROCESSOR CONTROL					
FINIT = Initialize FPU	11011 011 | 1110 0011	17			4
FSTSW AX = Store status word into AX	11011 111 | 1110 0000	3			5
FSTSW = Store status word into memory	11011 101 | mod 111 r/m | s-i-b/disp.	3			5
FLDCW = Load control word	11011 001 | mod 101 r/m | s-i-b/disp.	4	2		
FSTCW = Store control word	11011 001 | mod 111 r/m | s-i-b/disp.	3			5
FCLEX = Clear exceptions	11011 011 | 1110 0010	7			4
FSTENV = Store environment	11011 001 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		67			4
Real and Virtual modes 32-bit Address		67			4
Protected mode 16-bit Address		56			4
Protected mode 32-bit Address		56			4
FLDENV = Load environment	11011 001 | mod 100 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		44	2		
Real and Virtual modes 32-bit Address		44	2		
Protected mode 16-bit Address		34	2		
Protected mode 32-bit Address		34	2		

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range... Upper Range)	Penalty if Cache Miss Avg (Lower Range... Upper Range)	Concurrent Execution Avg (Lower Range... Upper Range)	Notes
PROCESSOR CONTROL (Continued)					
FSAVE = Save state	11011 101 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		154			4
Real and Virtual modes 32-bit Address		154			4
Protected mode 16-bit Address		143			4
Protected mode 32-bit Address		143			4
FRSTOR = Restore state	11011 101 | mod 100 r/m | s-i-b/				
Real and Virtual modes 16-bit Address		131	23		
Real and Virtual modes 32-bit Address		131	27		
Protected mode 16-bit Address		120	23		
Protected mode 32-bit Address		120	27		
FINCSTP = Increment Stack Pointer	11011 001 | 1111 0111	3			
FDECSTP = Decrement Stack Pointer	11011 001 | 1111 0110	3			
FFREE = Free ST(i)	11011 101 | 11000 ST(i)	3			
FNOP = No operations	11011 001 | 1101 0000	3			
WAIT = Wait until FPU ready	10011011				
(Minimum/Maximum)		1/3			

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
   If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where n = (operand/($\pi/4$)).

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the 486 Microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

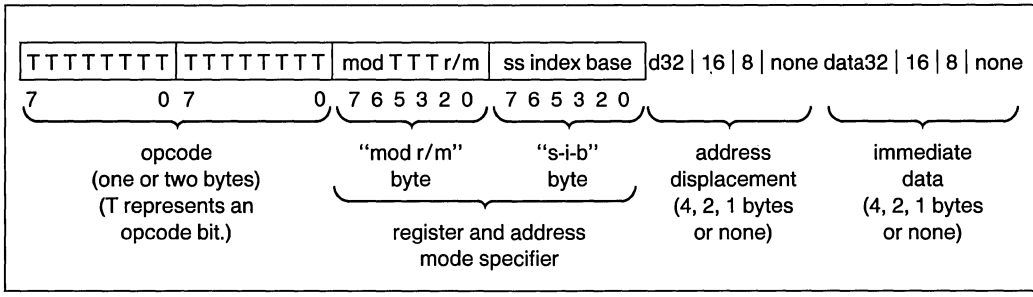


Figure 10.1. General Instruction Format

Table 10.4. Fields within i486™ Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the 486 Microprocessor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 486 Microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all 486 Microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
101	SI	ESI
101	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 486 Microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

	Instruction								Optional Fields	
	First Byte				Second Byte					
1	11011	OPA		1	mod		1	OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB		r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB		ST(i)		
4	11011	0	0	1	1	1	1	OP			
5	11011	0	1	1	1	1	1	OP			
	15–11	10	9	8	7	6	5	4 3 2 1 0			

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE i486™ MICROPROCESSOR AND THE 386™ MICROPROCESSOR PLUS THE 387™ NUMERICS COPROCESSOR

The differences between the 486 microprocessor and the 386 microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The 486 microprocessor bus is significantly faster than the 386 microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the 486 microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CE and WT) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by setting the CE and WT bit in CR0.
4. The complete 387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
5. The 486 microprocessor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).

6. Six new instructions have been added:
   Byte Swap (BSWAP)
   Exchange-and-Add (XADD)
   Compare and Exchange (CMPXCHG)
   Invalidate Data Cache (INVD)
   Write-back and Invalidate Data Cache (WBINVD)
   Invalidate TLB Entry (INVLPG)
7. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
8. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
9. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
10. The replacement algorithm for the translation lookaside buffer has been changed to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
11. Three new testability registers, TR5, TR6 and TR7, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
12. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
13. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

12.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the 486 microprocessor, and its electrical specifications.

12.1 Power and Grounding

12.1.1 POWER CONNECTIONS

The 486 microprocessor is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the 486 microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the 486 microprocessor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

12.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the 486 microprocessor. The 486 microprocessor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 486 microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

12.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

12.2 Maximum Ratings

Table 12.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 12.3 D.C. Specifications and 12.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 486 microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 12.1. Absolute Maximum Ratings

Case Temperature under Bias .....0°C to +85°C
Storage Temperature .....−65°C to +150°C
Voltage on Any Pin with
Respect to Ground.....−0.5 to V_{CC} + 0.5V
Power Dissipation at 25 MHz.....5W
Power Dissipation at 33 MHz.....6W

12.3 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 12.2. DC Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current (25 MHz)		900	mA	(Note 3)
	Power Supply Current (33 MHz)		1200		
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		16	pF	
C_O	I/O or Output Capacitance		16	pF	
C_{CLK}	CLK Capacitance		16	pF	

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn 1.0 mA
   Definition, Control 0.9 mA
3. Typical supply current 650 mA.
4. This parameter is for inputs without pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with pullups and V_{IL} = 0.45V.

12.4 A.C. Specifications

The A.C. specifications, given in Table 12.3, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 12.1–12.3. Inputs must be driven to the voltage levels indicated by Figure 12.3 when A.C. specifications are measured. 486 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum 486 microprocessor delay times are hold times provided to external circuitry. 486 microprocessor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 486 microprocessor operation.

Table 12.3. 25 MHz i486 Microprocessor A.C. Characteristics

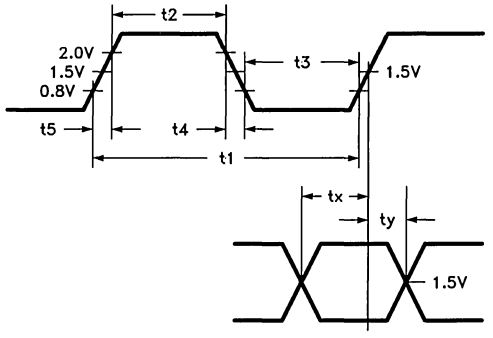
V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock Driven to 486
t_1	CLK Period	40	125	ns	12.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	12.1	at 2V
t_3	CLK Low Time	14		ns	12.1	at 0.8V
t_4	CLK Fall Time		4	ns	12.1	*
t_5	CLK Rise Time		4	ns	12.1	
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	12.2	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay	0		ns	12.2	Before Clock Edge
t_8	PCHK# Valid Delay	3	27	ns	12.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	12.2	
t_9	BLAST#, PLOCK# Float Delay	0		ns	12.2	Before Clock Edge
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	22	ns	12.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay	0		ns	12.2	Before Clock Edge
t_{12}	EADS# Setup Time	8		ns	12.3	
t_{13}	EADS# Hold Time	3		ns	12.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	12.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	12.3	
t_{16}	RDY#, BRDY# Setup Time	8		ns	12.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	12.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		ns	12.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	10		ns	12.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	12.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	12.3	

Table 12.3. 33 MHz i486 Microprocessor D.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to 486
t_1	CLK Period	30	125	ns	12.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	12.1	at 2V
t_3	CLK Low Time	11		ns	12.1	at 0.8V
t_4	CLK Fall Time		3	ns	12.1	
t_5	CLK Rise Time		3	ns	12.1	
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	19	ns	12.2	*
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay	0		ns	12.2	Before Clock Edge
t_8	PCHK# Valid Delay	3	27	ns	12.2	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	22	ns	12.2	
t_9	BLAST#, PLOCK# Float Delay	0		ns	12.2	Before Clock Edge
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	19	ns	12.2	
t_{11}	D0–D31, DP0–3 Write Data Float Delay	0		ns	12.2	Before Clock Edge
t_{12}	EADS# Setup Time	6		ns	12.3	
t_{13}	EADS# Hold Time	3		ns	12.3	
t_{14}	KEN#, BS16#, BS8# Setup Time	6		ns	12.3	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	12.3	
t_{16}	RDY#, BRDY# Setup Time	6		ns	12.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	12.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	8		ns	12.3	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.3	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	8		ns	12.3	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.3	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	12.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	12.3	



tx = input setup times
ty = input hold times, output float, valid and hold times

Figure 12.1. CLK Waveforms

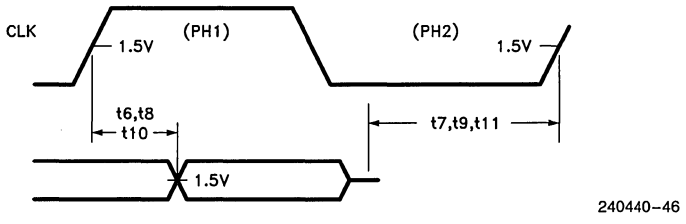
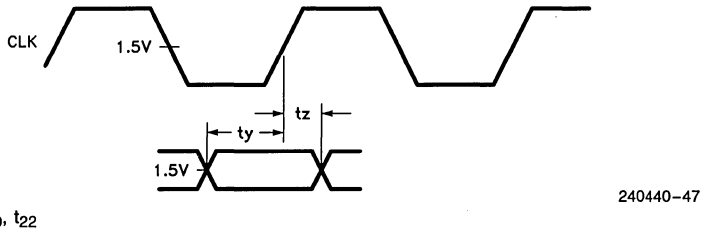


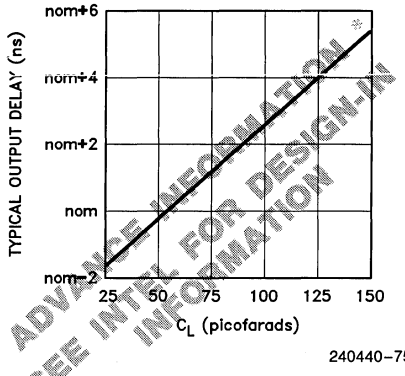
Figure 12.2. Output Waveforms



ty = t_{12}, t_{14}, t_{16}, t_{18}, t_{20}, t_{22}
tz = t_{13}, t_{15}, t_{17}, t_{19}, t_{21}, t_{23}

Figure 12.3. Input Waveforms

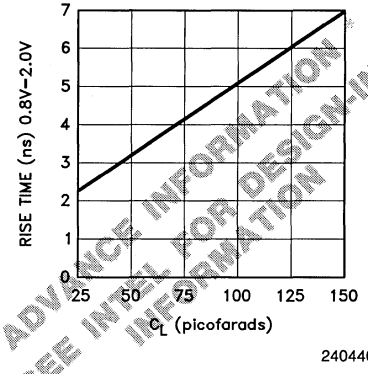
12.4.1 Typical Output Valid Delay versus Load Capacitance Under Worst Case Conditions



240440-75

NOTE:
This graph will not be linear outside of the C_L range shown. non = nominal value given in A.C. Characteristics table.

12.4.2 Typical Output Rise Time versus Load Capacitance Under Worst-Case Conditions



240440-76

NOTE:
This graph will not be linear outside of the C_L range shown.

12.5 Designing for ICD-486 (Advance Information)

The ICD-486 (In-Circuit Debugger) is a hardware assisted debugger for the 486 CPU. To use the ICD-486, the 486 CPU component must be removed from its socket replaced with the ICD-486 module. Because of the high operating frequency of 486 CPU systems, there is no buffering of signals between the 486 CPU in the ICD-486 and the target system. A direct result of the non-buffered interconnect is that the ICD-486 shares the address and data bus of the target system. In order for the ICD-486 to function properly (without the Optional Isolation Board installed), the design of the target system must meet the following restrictions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 486 CPU, other local devices, or other bus masters.
2. Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD-HLDA, AHOLD, or BOFF#.

In addition to the above restrictions, the ICD-486 has several electrical and mechanical characteristics that should be taken into consideration when designing the 486 CPU system.

Capacitive Loading: ICD-486 adds up to 30 pF to the CLK signal, and up to 20 pF to each of the other 486 CPU signals.

DC Loading: ICD-486 adds ±15 μA loading to the CLK and data bus signals and ±5 μA loading to the address and control signals.

Power Requirements: For noise immunity and CMOS latch-up protection the ICD-486 is powered by the target system through the power and ground pins of the 486 CPU socket. The circuitry on the ICD-486 draws up to 1.3A excluding the 486 CPU I_{CC}.

No Connects: Pins specified as N.C. in the 486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground, or any other signal may cause the processor or the ICD-486 to malfunction.

486 CPU Location and Orientation: The ICD-486 may require lateral clearance. Figure 12.4 shows the clearance requirements of the ICD-486.

Optional Isolation Board (OIB)

Due to its unbuffered design, the ICD-486 is susceptible to errors on the target system's bus. The OIB installs between the ICD-486 and 486 CPU socket in the target system and allows the ICD-486 to function in systems with faults (i.e., shorted signals). After electrical verification the OIB may be removed. The OIB has the following electrical and mechanical characteristics:

Buffer Characteristics: The OIB buffers the address and data busses as well as the byte enables, ADS#, W/R#, M/IO#, BLAST#, and HLDA. The buffers are advanced CMOS devices and have the following DC drive specifications: $I_{OH} = -15$ mA, $I_{OL} = 64$ mA. The propagation delay of each buffer is 5 ns max driving a 50 pF load. To guarantee proper operation with the OIB, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

Unbuffered Signals: Signals not listed above as buffered are passed through the OIB and will have additional capacitive loading due to the connectors and circuit board of up to 10 pF.

Power Requirements: The OIB is also powered by the target system through the 486 CPU socket and requires 0.5A in addition to the ICD-486 and 486 CPU requirements.

OIB Clearance Requirements: The OIB requires an extra 0.55″ of vertical clearance in the target system above the 486 CPU socket.

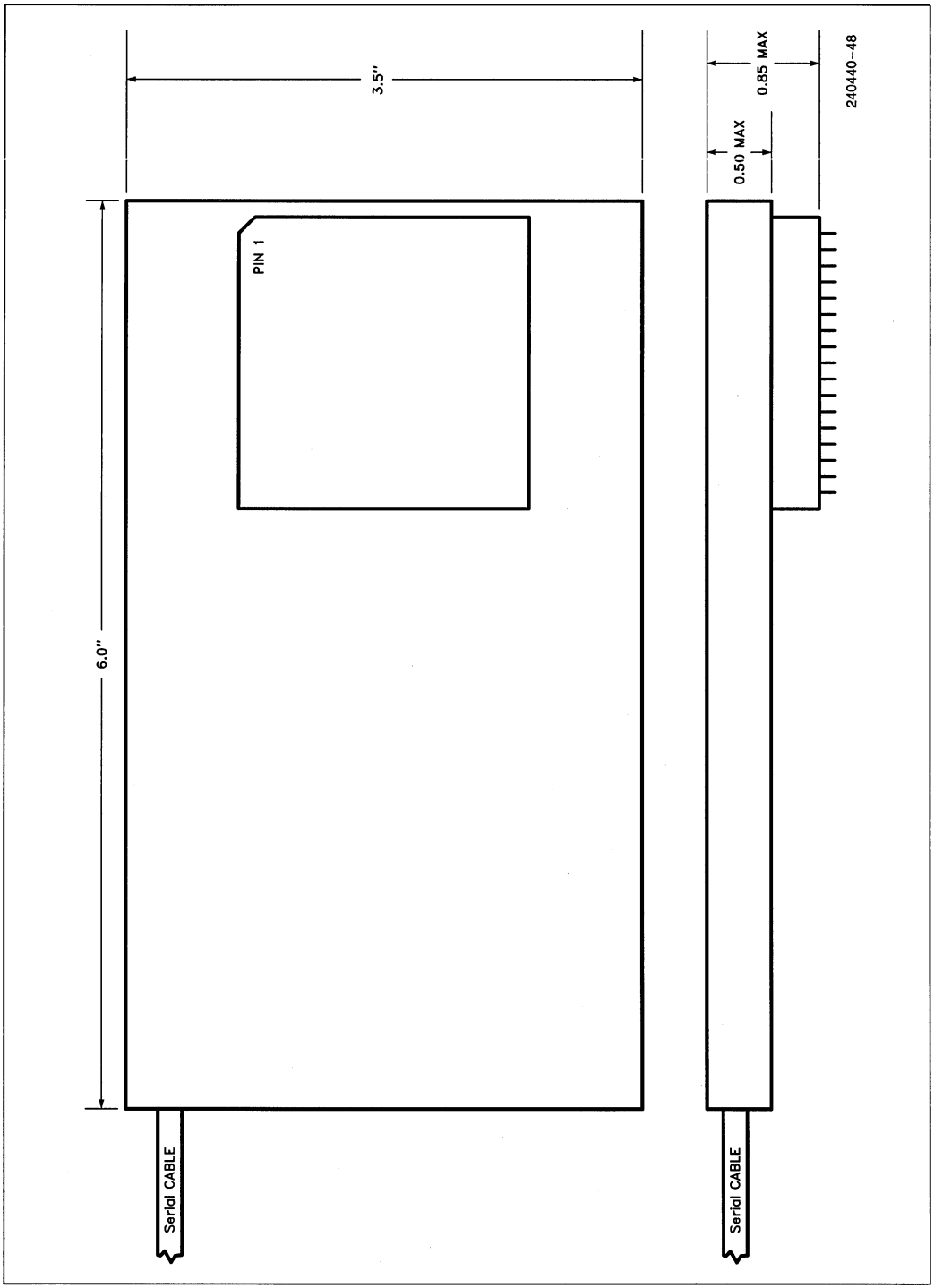
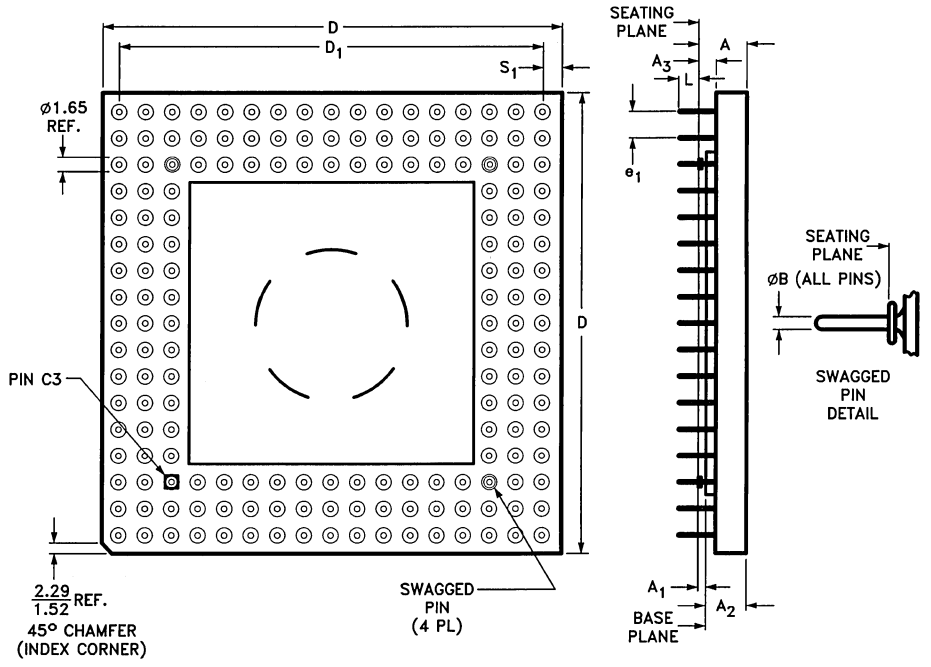


Figure 12.4. ICD-486™ CPU Dimensions

13.0 MECHANICAL DATA



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	23	0.30	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 13.1. 168 Lead Ceramic PGA Package Dimensions

Table 13.1 Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

13.1 Package Thermal Specifications

The 486 microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the 486 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A (the ambient temperature) can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P^*\theta_{CA}$$

Typical values for θ_{CA} at various airflows are given in Table 13.2 for the 1.75 sq. in., 168 pin, ceramic PGA.

Table 13.3 shows the maximum T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 12.

Table 13.2. Thermal Resistance (θ_{CA}) at Various Airflows

In °C/Watt

	Airflow-ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
θ_{CA} with Heat Sink*	13	9	5.5	5.0	3.9	3.4
θ_{CA} without Heat Sink	17	14	11	9	7.1	6.6

*0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

Table 13.3. Maximum T_A at Various Airflows

In °C

		Airflow-ft/min (m/sec)					
	f_{CLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	25.0	46	58	69	70	73	75
	33.3	43	56	67	69	72	74
T_A without Heat Sink	25.0	34	43	52	58	64	65
	33.3	30	40	49	56	62	64

*0.285" high unidirectional heat sink (Al alloy 6061, 50 mil fin width, 150 mil center-to-center fin spacing).

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

HONG KONG
Intel Semiconductor Ltd.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Order Number: 240440-001

Printed in U.S.A./XA9401A/15K/0389/SCP RJ
Microprocessors