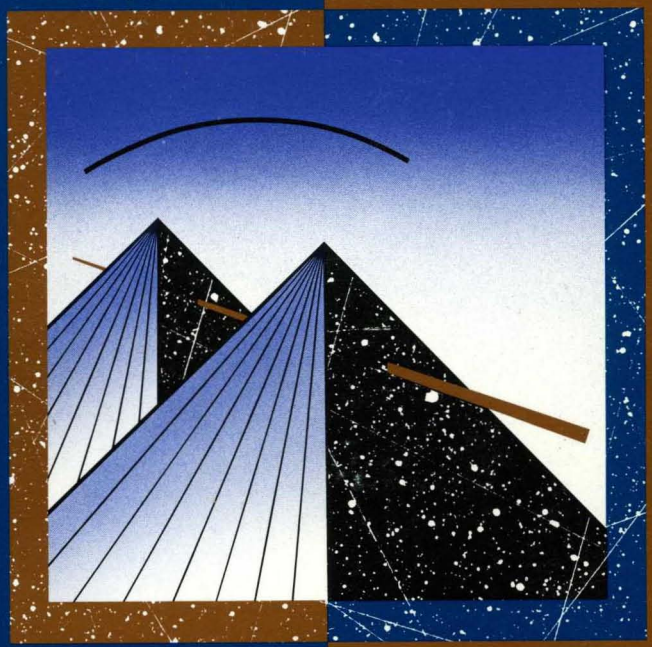
intel



Intel486™ DX2 Microprocessor Data Book

PRELIMINARY

Intel486™ DX2 MICROPROCESSOR

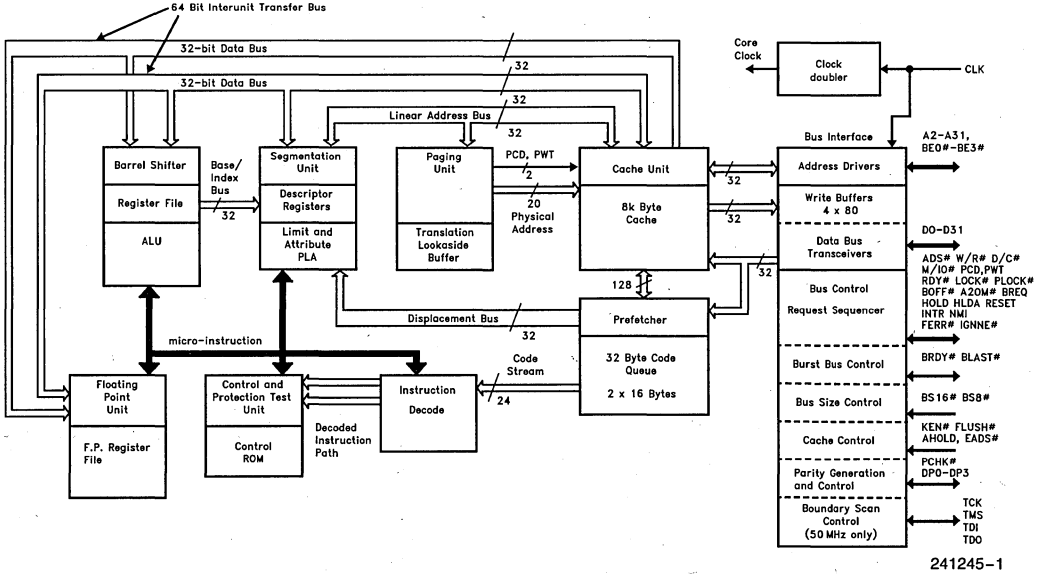
- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows**
  - **— UNIX*** System V/Intel386**
  - **— iRMX®, iRMK™ Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built-In Self Test**
  - **— Hardware Debugging Support**
- **168-Pin Grid Array Package**
  - **— Pin Compatible with Intel486™ DX Microprocessor**
- **IEEE 1149.1 Boundary Scan Compatibility**
- **High Performance Design**
  - **— 50 MHz/66 MHz Core Speed Using 25 MHz/33 MHz Bus Clocks**
  - **— RISC Integer Core with Frequent Instructions Executing in One Core Clock**
  - **— 80, 106 Mbyte/sec Burst Bus**
  - **— Dynamic Bus Sizing for 8-, 16-, and 32-Bit Busses**
  - **— Complete 32-Bit Architecture**
- **Multiprocessor Support**
  - **— Cache Consistency Protocols**
  - **— Support for Second Level Cache**

The Intel486 DX2 CPU offers the highest performance for DOS, OS/2, Windows, and UNIX System V/Intel386 applications. It is 100% binary compatible with the Intel386™ CPU. Over one million transistors integrate the RISC integer core, 8 Kbyte cache memory, floating point hardware, and memory management on-chip while retaining binary compatibility with previous members of the Intel386/Intel486 architectural family. The RISC integer core executes frequently-used instructions in one core clock cycle, providing leadership performance levels. An 8 Kbyte unified code and data cache allow the high performance levels to be sustained. A 106 MByte/sec burst bus at 33 MHz bus clock ensures high system throughput even with inexpensive DRAMs.

New features enhance multiprocessing systems; new instructions speed manipulation of memory-based semaphores; and 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.

Intel486™ DX2 Microprocessor Pipelined 32-Bit Microarchitecture



241245-1

 February 1992

Order Number: 241245-001

intel®

Quick Reference to Chapters

Intel486™ DX2 Microprocessor

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

PAGE

CONTENTS

PAGE

1.0 INTRODUCTION

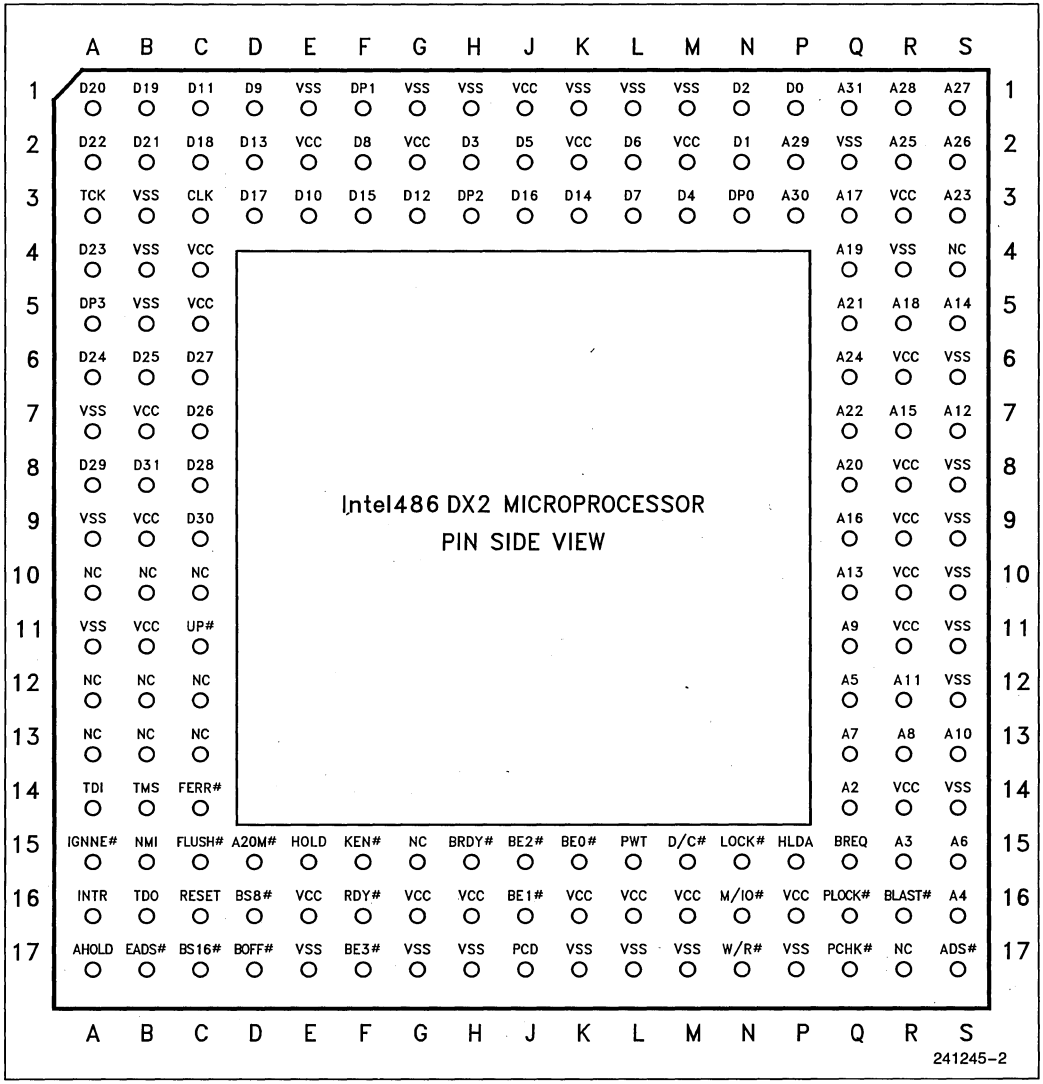


Figure 1.1

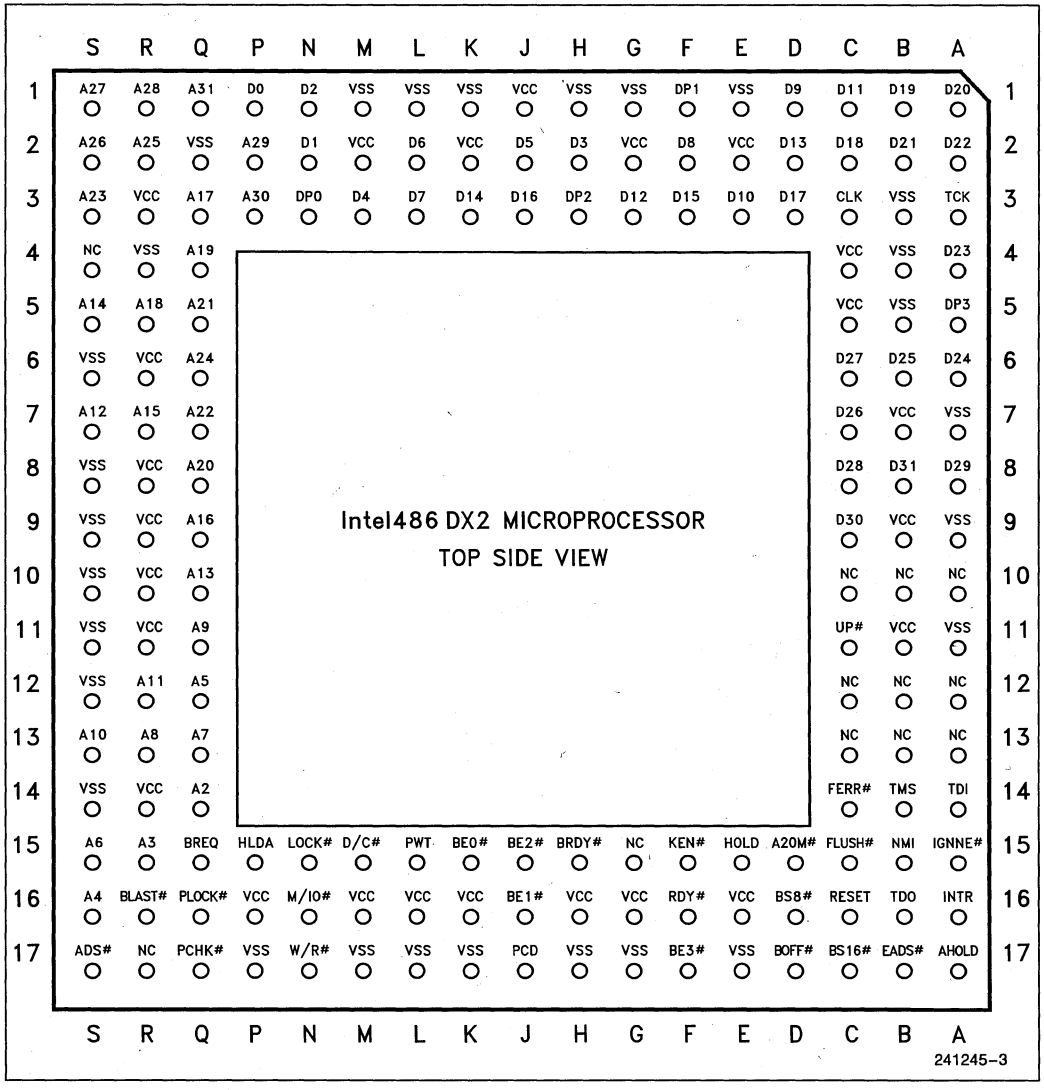

S R Q P N M L K J H G F E D C B A
1 A27 A28 A31 D0 D2 VSS VSS VSS VCC VSS VSS DP1 VSS D9 D11 D19 D20 1
2 A26 A25 VSS A29 D1 VCC D6 VCC D5 D3 VCC D8 VCC D13 D18 D21 D22 2
3 A23 VCC A17 A30 DP0 D4 D7 D14 D16 DP2 D12 D15 D10 D17 CLK VSS TCK 3
4 NC VSS A19 VCC VSS D23 4
5 A14 A18 A21 VCC VSS DP3 5
6 VSS VCC A24 D27 D25 D24 6
7 A12 A15 A22 D26 VCC VSS 7
8 VSS VCC A20 D28 D31 D29 8
9 VSS VCC A16 D30 VCC VSS 9
10 VSS VCC A13 NC NC NC 10
11 VSS VCC A9 UP# VCC VSS 11
12 VSS A11 A5 NC NC NC 12
13 A10 A8 A7 NC NC NC 13
14 VSS VCC A2 FERR# TMS TDI 14
15 A6 A3 BREQ HLDA LOCK# D/C# PWT BE0# BE2# BRDY# NC KEN# HOLD A20M# FLUSH# NMI IGNNE# 15
16 A4 BLAST# PLOCK# VCC M/IO# VCC VCC VCC BE1# VCC VCC RDY# VCC BS8# RESET TDO INTR 16
17 ADS# NC PCHK# VSS W/R# VSS VSS VSS PCD VSS VSS BE3# VSS BOFF# BS16# EADS# AHOLD 17
S R Q P N M L K J H G F E D C B A
Intel486 DX2 MICROPROCESSOR
TOP SIDE VIEW
241245-3

Figure 1.2

Pin Cross Reference by Pin Name

Address		Data		Control		Test(1)		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	TCK	A3	A10	B7	A7
A_3	R15	D_1	N2	ADS#	S17	TDI	A14	A12	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	TDO	B16	A13	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	TMS	B14	B12	C4	B3
A_6	S15	D_4	M3	BE1#	J16			B13	C5	B4
A_7	Q13	D_5	J2	BE2#	J15			C10	E2	B5
A_8	R13	D_6	L2	BE3#	F17			C13	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16			G15	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17			R17	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15			S4	H16	G17
A_{12}	S7	D_{10}	E3	BREQ#	Q15				J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16				K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17				K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3				L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15				M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3				M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1				P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3				R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5				R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17				R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	C14				R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15				R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15				R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15				R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15					S10
A_{27}	S1	D_{25}	B6	INTR	A16					S11
A_{28}	R1	D_{26}	C7	KEN#	F15					S12
A_{29}	P2	D_{27}	C6	LOCK#	N15					S14
A_{30}	P3	D_{28}	C8	M/IO#	N16					
A_{31}	Q1	D_{29}	A8	NMI	B15					
		D_{30}	C9	PCD	J17					
		D_{31}	B8	PCHK#	Q17					
				PWT	L15					
				PLOCK#	Q16					
				RDY#	F16					
				RESET	C16					
				RES_A(1)	B10					
				RES_B(1)	C12					
				UP#(1)	C11					
				W/R#	N17					

NOTE:
1. These pins were No-Connects on the 25 MHz and 33 MHz Intel486 DX microprocessors. For compatibility with old designs they can still be left 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 for the bus interface unit and is multiplied by two (2x) to provide the internal frequency for the Intel486 DX2. 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 not driven during bus or address hold.
BE0–3#	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 Intel486 DX2 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 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 Intel486 DX2 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 Intel486 DX2 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.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
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.

LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 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.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
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 interrupted 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.
INTERRUPTS		
RESET	I	The *reset* input forces the Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 microprocessor has internally generated a bus request. BREQ is generated whether or not the Intel486 DX2 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 Intel486 DX2 microprocessor bus. In response to HOLD going active the Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the Intel486 DX2 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS ARBITRATION (Continued)		
BOFF#	I	The *backoff* input forces the Intel486 DX2 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 Intel486 DX2 microprocessor's address bus for a cache invalidation cycle. The Intel486 DX2 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}.
EADS#	I	This signal indicates that a *valid external address* has been driven onto the Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 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. FLUSH# being sampled low in the clock before the falling edge of RESET causes the Intel486 DX2 microprocessor to enter the tri-state test mode.
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 disable bit (CD) 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 Intel387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
NUMERIC ERROR REPORTING (Continued)		
IGNNE#	I	When the *ignore numeric error* pin is asserted the Intel486 DX2 microprocessor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the Intel486 DX2. When IGNNE# is deasserted the Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 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 Intel486 DX2 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. For proper operation, A20M# should be sampled high at the falling edge of RESET.
TEST ACCESS PORT		
TCK	I	*Test Clock* is an input to the Intel486 DX2 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.
TDI	I	*Test Data Input* is the serial input used to shift JTAG instructions and data into the component. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR TAP controller states. During all other tap controller states, TDI is a "don't care".
TDO	O	*Test Data Output* is the serial output used to shift JTAG instructions and data out of the component. TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.
TMS	I	*Test Mode Select* is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic. TMS is sampled on the rising edge of TCK. To guarantee deterministic behavior of the TAP controller TMS is provided with an internal pull-up resistor.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
POWER DOWN MODE (UPGRADE PROCESSOR SUPPORT)		
UP#	I	The *Upgrade Present* pin forces the Intel486 DX2 to 3-state all of its outputs and enter the power down mode. When the Upgrade Present pin is sampled asserted by the CPU in the clock before the falling edge of RESET, the power down mode is enabled. UP# has no effect on the power down status except during this edge. The CPU is also forced to 3-state all of it's outputs immediately in response to this signal. The UP# signal must remain asserted in order to keep the pins 3-stated. UP# is active low and is provided with an internal pull-up resistor.

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	
RES_B	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
RES_A	LOW	Asynchronous
UP#	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

Table 1-4. Test Pins

Name	Input or Output	Sampled/ Driven On
TCK	Input	N/A
TDI	Input	Rising Edge of TCK
TDO	Output	Falling Edge of TCK
TMS	Input	Rising Edge of TCK

2.0 ARCHITECTURAL OVERVIEW

The Intel486 DX2 microprocessor is a fully compatible member of the Intel486 family.

The Intel486 microprocessor family is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The Intel486 DX microprocessor contains all the features of the Intel386™ microprocessor with enhancements to increase performance. The instruction set includes the complete Intel386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the Intel386 microprocessor MMU. The Intel486 DX microprocessor brings the Intel387™ math coprocessor on-chip. All software written for the Intel386 microprocessor, Intel387 math coprocessor and previous members of the 86/87 architectural family will run on the Intel486 DX microprocessor without any modifications.

Several enhancements have been added to the Intel486 DX 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. A clock doubler has been added to speed up internal operations to twice that of an Intel486 DX microprocessor running with the same bus clock. 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 triple that of a Intel386 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the Intel486 DX microprocessor operates as a very fast 8086. Real Mode is required primarily to set up 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 Intel486 DX2 microprocessor has features to facilitate high performance hardware designs. The 1x bus clock eases high frequency board level designs. While the 2x clock doubler improves execution performance without increasing the board design complexity. This 2x clock doubler enhances all operations operating out of the cache and/or not blocked by external bus accesses. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The Intel486 DX microprocessor register set includes all the registers contained in the Intel386 microprocessor and the Intel387 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

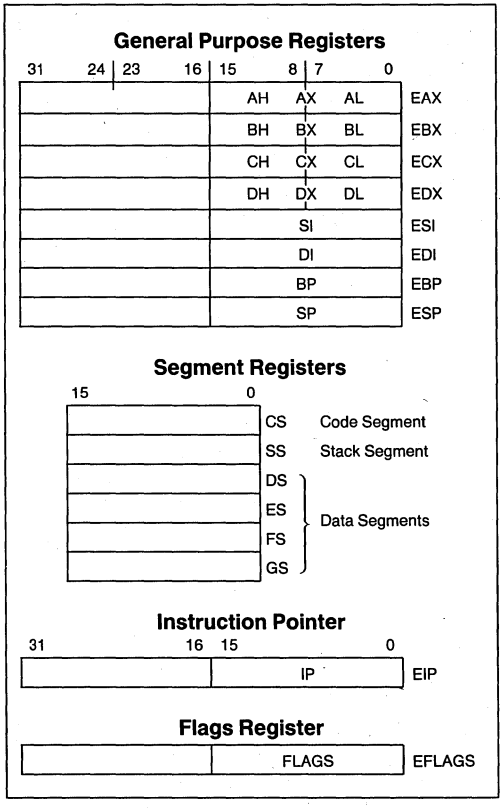


Figure 2.1. Base Architecture Registers

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 Intel486 DX2 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.

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 Intel486 DX2 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

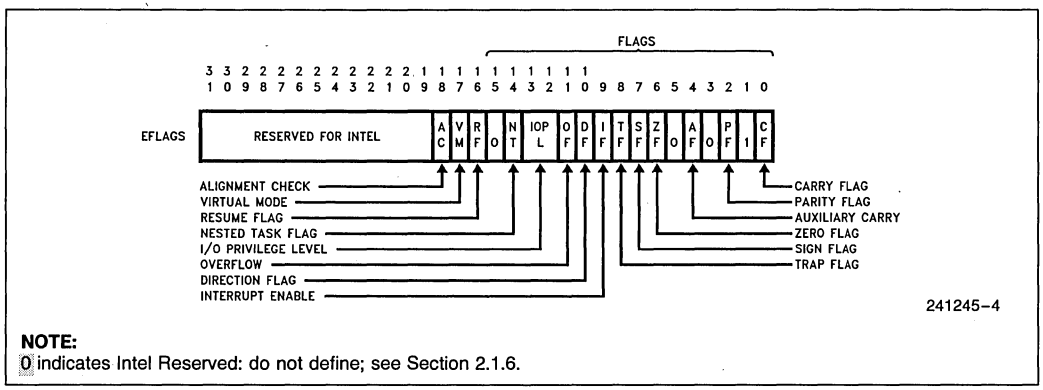
The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers 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



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

Figure 2.2. Flags Register

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 Intel486 DX2 microprocessor. The lower 16 bits (bits 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.

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 Intel486 DX microprocessor contains a new bit not previously defined. 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 Intel486 DX 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 Intel486 DX microprocessor generate misaligned references, even if their memory address is aligned. For example, on the Intel486 DX microprocessor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The Intel486 DX microprocessor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The Intel486 DX 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 Intel486 DX microprocessor is in Protected Mode, the Intel486 DX 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 Intel486 DX 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.

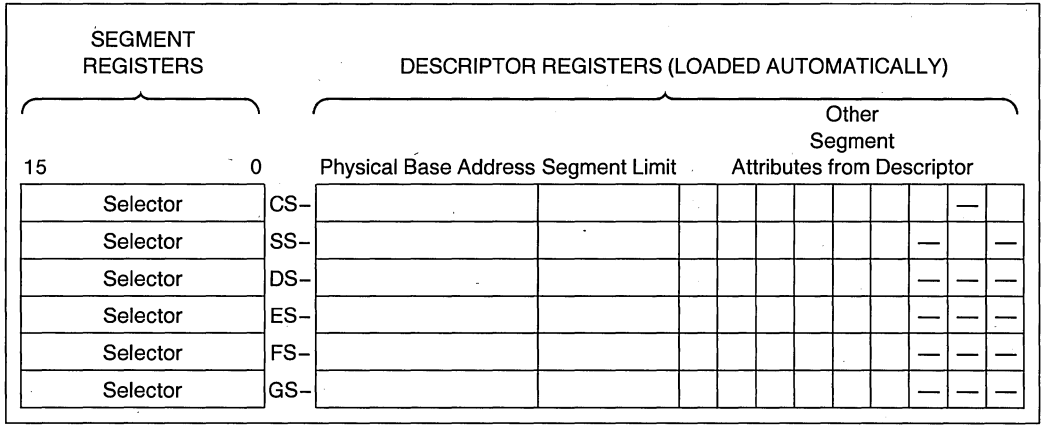


Figure 2.3. Intel486™ DX Microprocessor Segment Registers and Associated Descriptor Cache Registers

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 (IDTR), 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 Intel486 DX microprocessor's CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

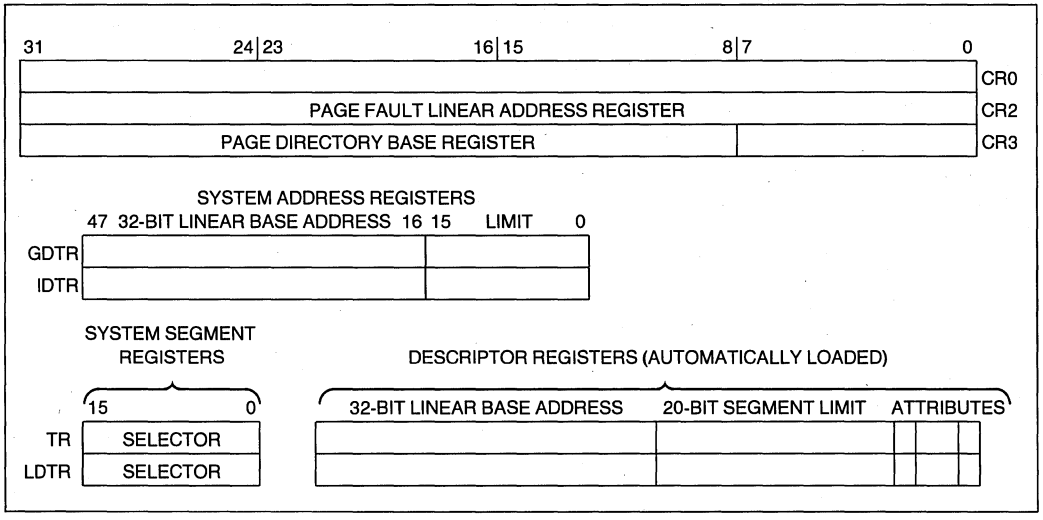


Figure 2.4. System Level Registers

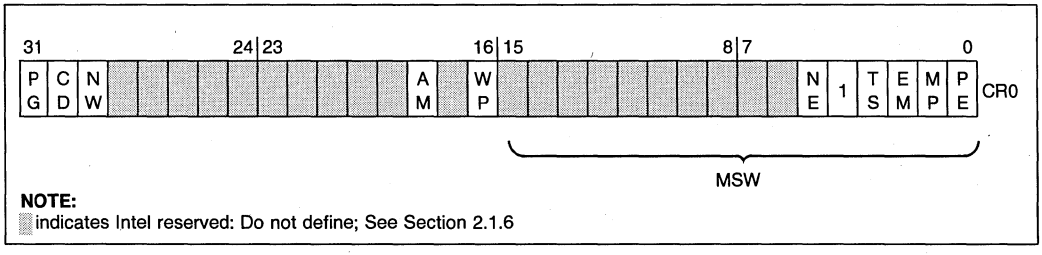


Figure 2.5. Control Register 0

Intel486 DX Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (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

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	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 Intel486 DX 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 Intel486 DX 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.

CD (Cache Disable, bit 30)

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

The state of the CD 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 CD=0 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.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW=0). When NW=0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW=0. 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 NW=1, 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 NW=1 the only write cycles that reach the external bus are cache misses. Write hits with NW=1 will never update main memory. Invalidate cycles are ignored when NW=1.

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 Intel386 microprocessor compatible mode.

Intel386 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 Intel486 DX microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The Intel386 microprocessor allows a read-only page to be written from privilege levels 0–2. The Intel486 DX microprocessor is compatible with the Intel386 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 Intel387 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel386 and Intel486 DX microprocessors, except for ET and NE, are upward 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.

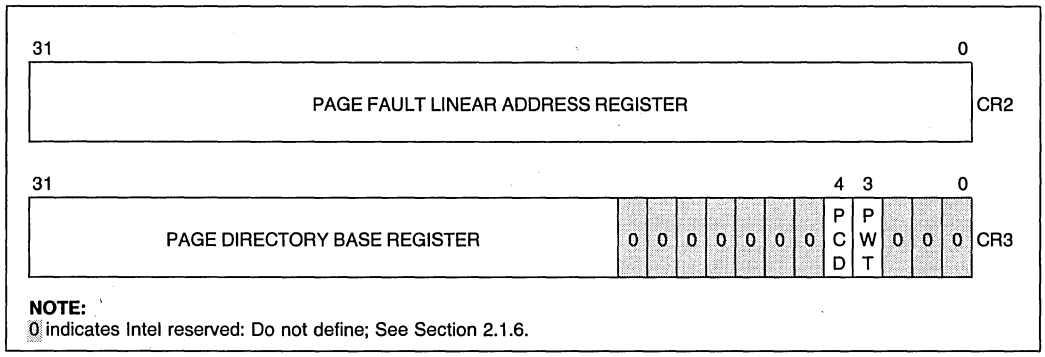


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The Intel486 DX microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the Intel486 DX 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 when the PDE is being updated. When paging is disabled (PG = 0 in CR0), PCD and PWT are assumed to be 0, regardless of their state in CR3.

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).

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, Intel386 and Intel486 DX 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.

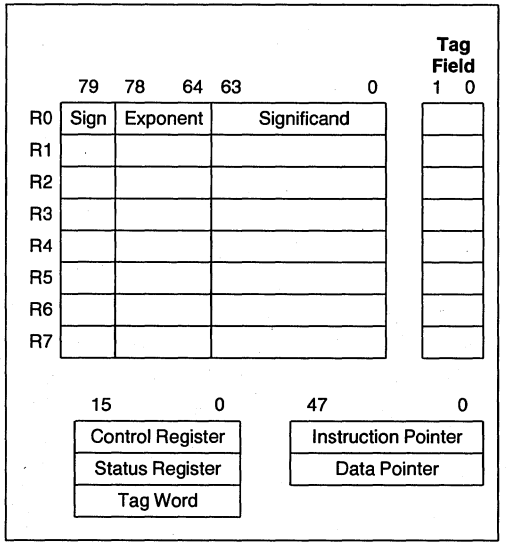


Figure 2.7. Floating Point Registers

The operation of the Intel486 DX microprocessor's on-chip floating point unit is exactly the same as the Intel387 math coprocessor. Software written for the Intel387 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 Intel486 DX 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 Intel486 DX microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

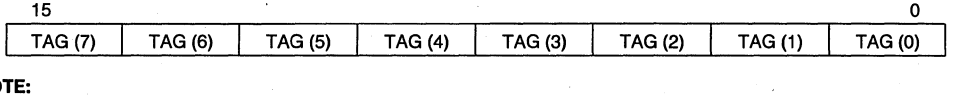
Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use.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

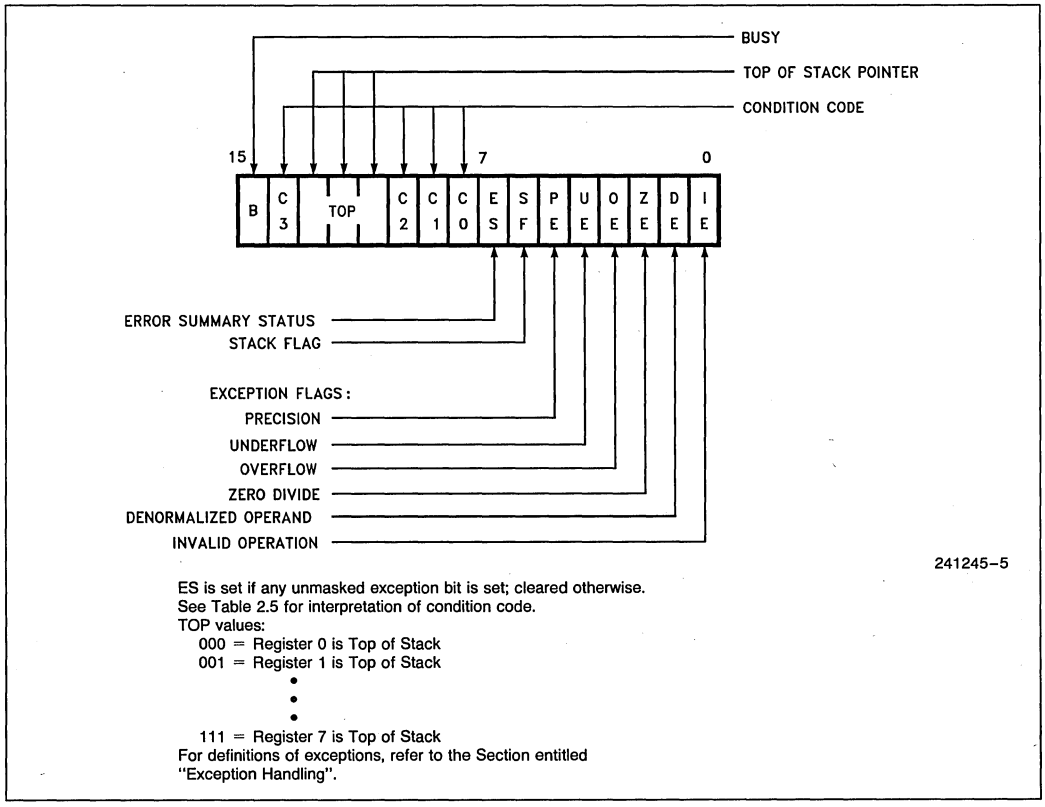


Figure 2.9. FPU Status Word

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			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, 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	
0	0	0	0	0	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	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 Intel486 DX 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 Intel486 DX microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the Intel486 DX and Intel486 microprocessors 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

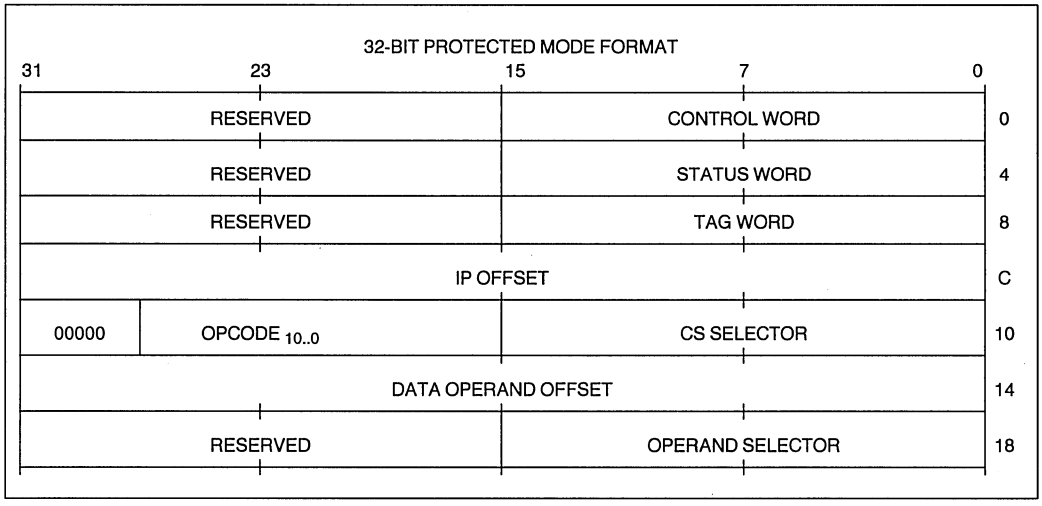


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

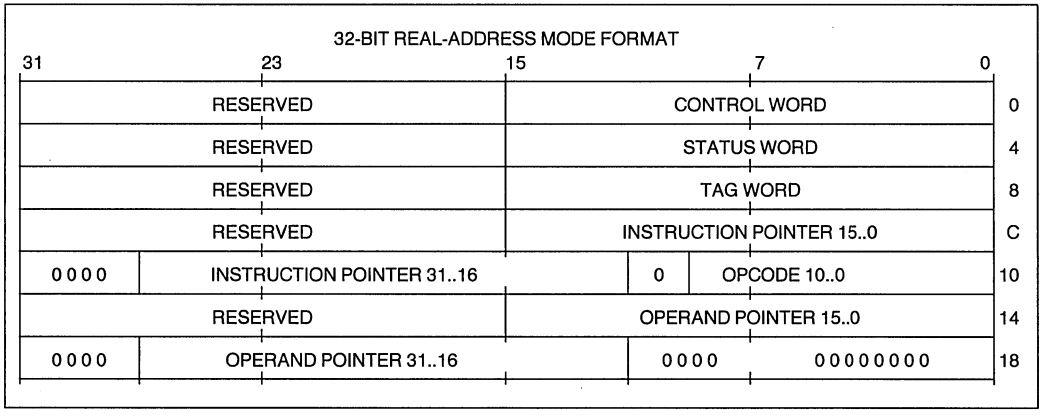


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

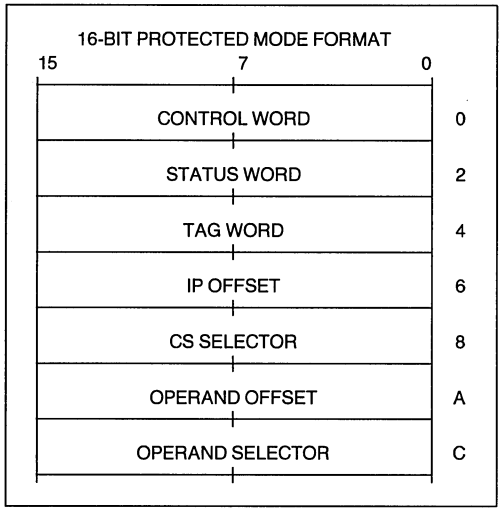


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

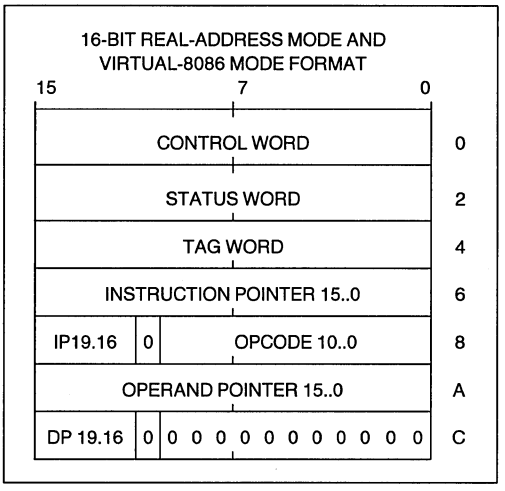
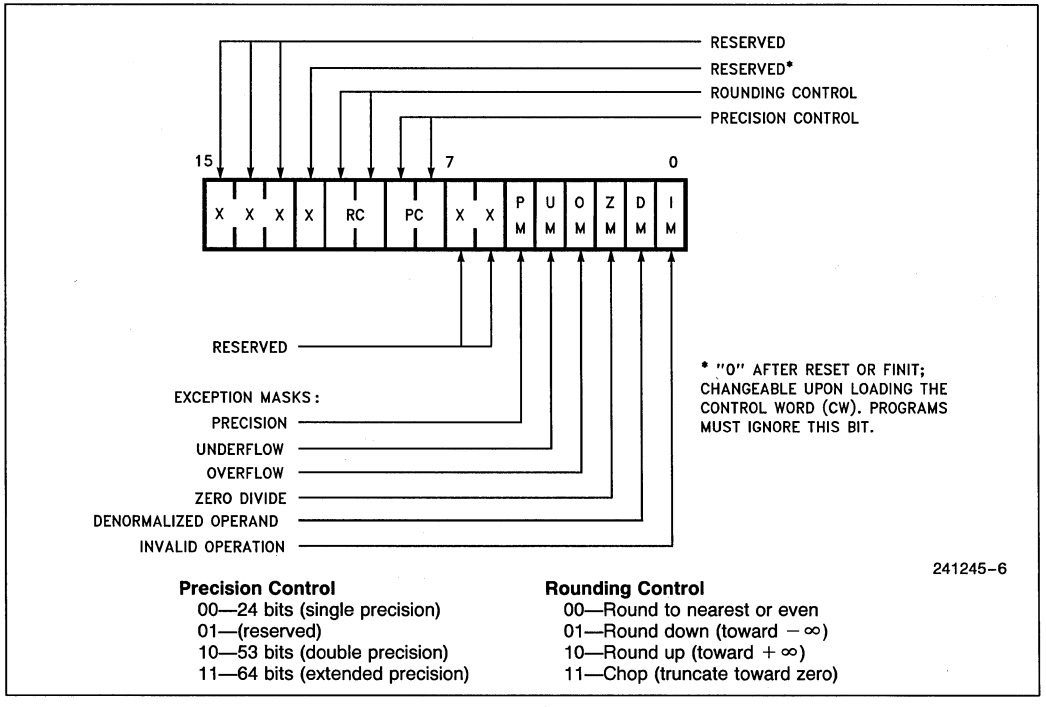


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

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.



Precision Control
00—24 bits (single precision)
01—(reserved)
10—53 bits (double precision)
11—64 bits (extended precision)

Rounding Control
00—Round to nearest or even
01—Round down (toward $-\infty$)
10—Round up (toward $+\infty$)
11—Chop (truncate toward zero)

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 Intel486 DX 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 Intel486 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 Intel486 DX microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the Intel486 microprocessor-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED Intel486 MICROPROCESSOR REGISTER BITS.

2.2 Instruction Set

The Intel486 DX 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 Intel486 DX microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All Intel486 DX 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 Intel486 DX 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 Intel486 DX, Intel486 or Intel386 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

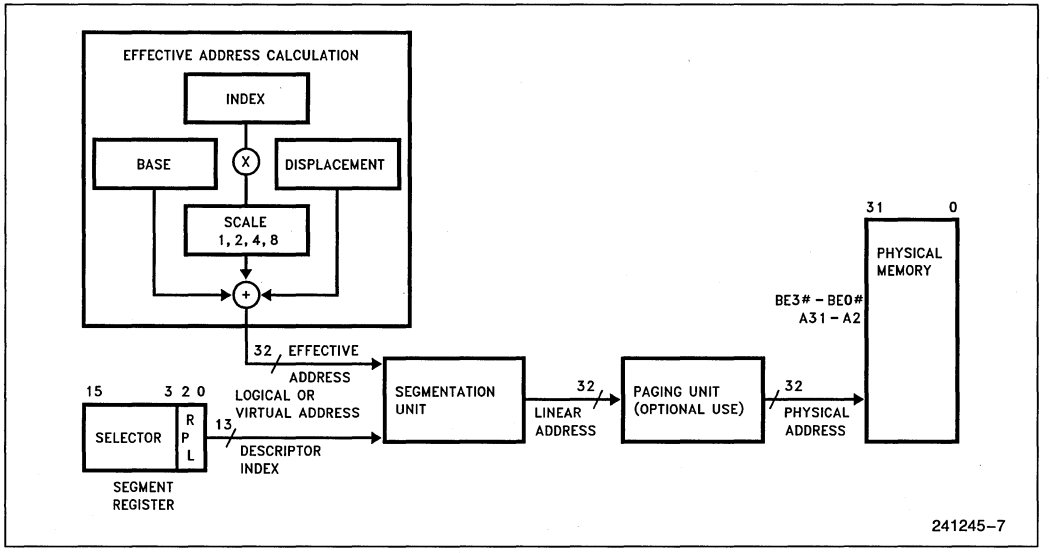


Figure 2.16. Address Translation

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 Intel486 DX 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 provide 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 Intel486 DX microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Intel486 DX 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 Intel486 DX 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 operand's offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's 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 operand's 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 operand's offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

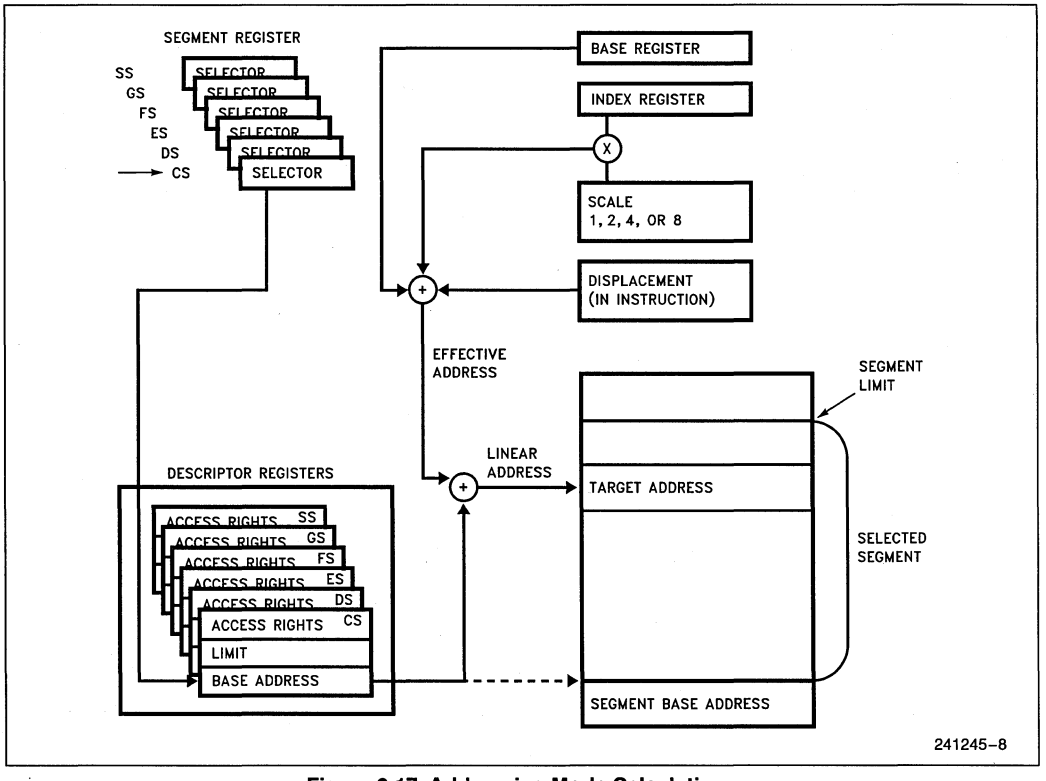


Figure 2.17. Addressing Mode Calculations

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX Microprocessor addressing modes.

When executing 32-bit code, the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

The Intel486 DX microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data. Refer to Table 2.14.

Table 2.13. Intel486™ DX Microprocessor Data Types

Data Format	Supported by Base Registers	Supported by FPU	Range	Precision
Byte	X		0–255	8 bits
Word	X		0–64K	16 bits
Dword	X		0–4G	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	$\pm 10^{\pm 18}$	18 Digits
Single Precision Real		X	$\pm 10^{\pm 38}$	24 Bits
Double Precision Real		X	$\pm 10^{\pm 308}$	53 Bits
Extended Precision Real		X	$\pm 10^{\pm 4932}$	64 Bits

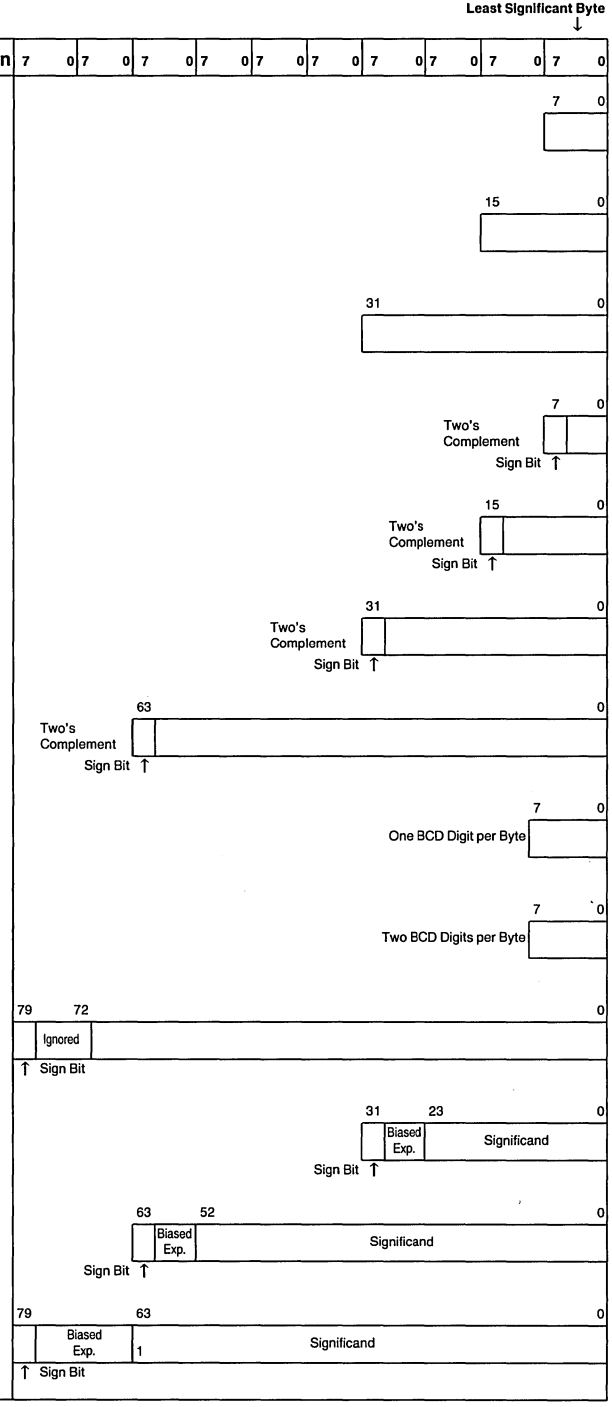
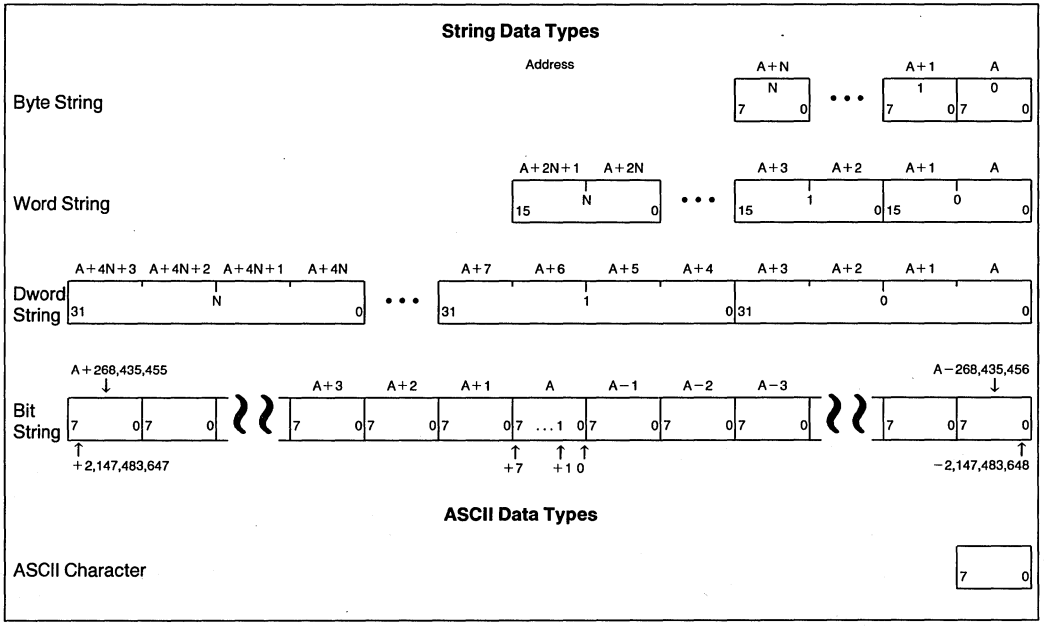


Table 2.14. String and ASCII Data Types



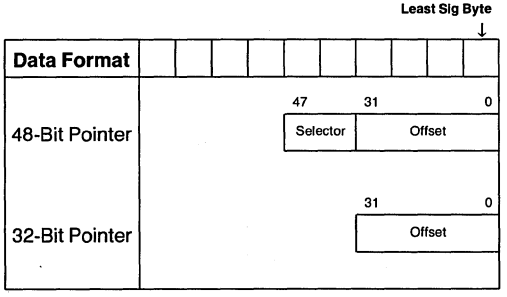
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

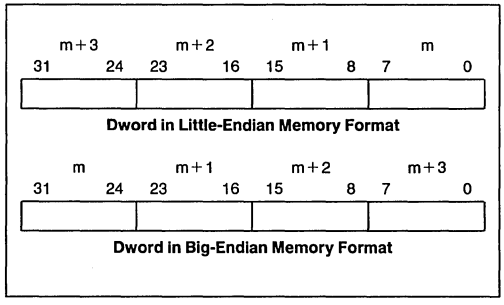


Figure 2.18. Big vs Little Endian Memory Format

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 Intel486 DX 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 Intel486 DX microprocessor and shows where the return address points.

The Intel486 DX 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 sup-

plied to the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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	18–31			
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 Intel486 DX 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 Intel486 DX 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 9.2.

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 Intel486 DX microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Intel486 DX microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. Intel486™ DX 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 Intel486 DX microprocessor if, in the course of executing an instruction, the Intel486 DX microprocessor detects a problematic condition. The Intel486 DX microprocessor then immediately invokes the appropriate exception service routine. The state of the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor invokes the appropriate exception service routine. The Intel486 DX 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 Intel486 DX microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the Intel486 DX 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 Intel486 DX 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 Intel486™ DX 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. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions 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 Intel486 microprocessor family. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the Intel486 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 Intel486 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.

The LOCK prefix on the Intel486 microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Intel486 microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Intel486 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 Intel486 DX2 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
CMPXCHG, XADD	Mem, Reg

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 Intel486 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 Intel486 microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

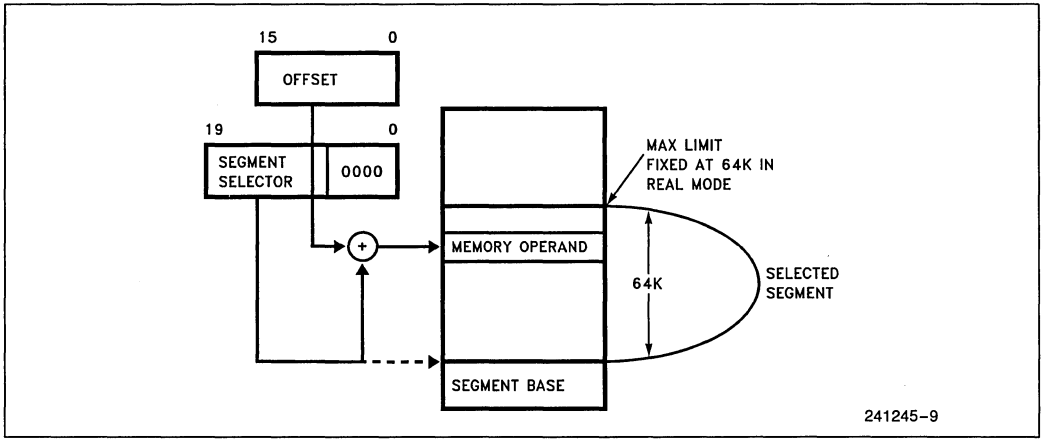


Figure 3.1. Real Address Mode Addressing

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, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

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, 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 Intel486 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, 17, 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 Intel486 DX2 microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case in protected mode, the 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).

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). If these conditions are not met, the Intel486 DX2 CPU is unable to execute the NMI and executes another shutdown cycle. In this case, the processor remains in the shutdown and can only exit via the RESET input.

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

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

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the Intel486 DX2 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 Intel486 DX2 microprocessor to run all of the existing 8086, 80286 and Intel386 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 Intel486 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 Intel486 microprocessor family. 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 Intel486 DX2 addressing mechanism with paging enabled.

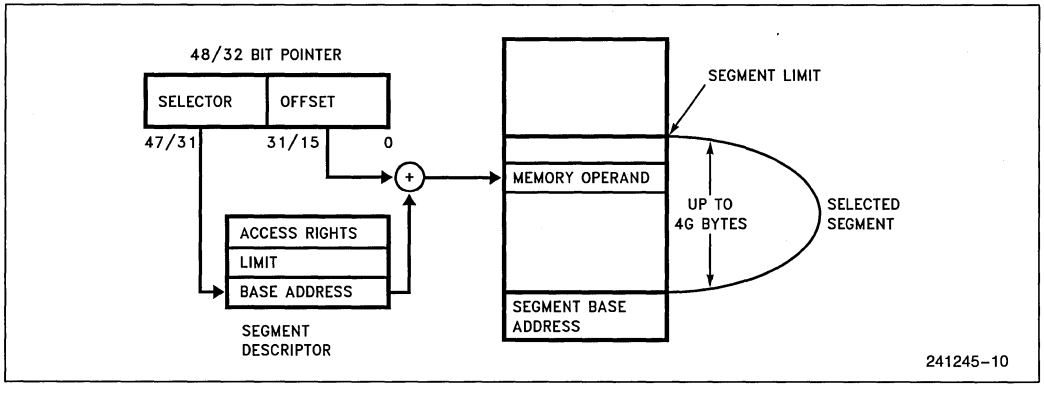


Figure 4.1. Protected Mode Addressing

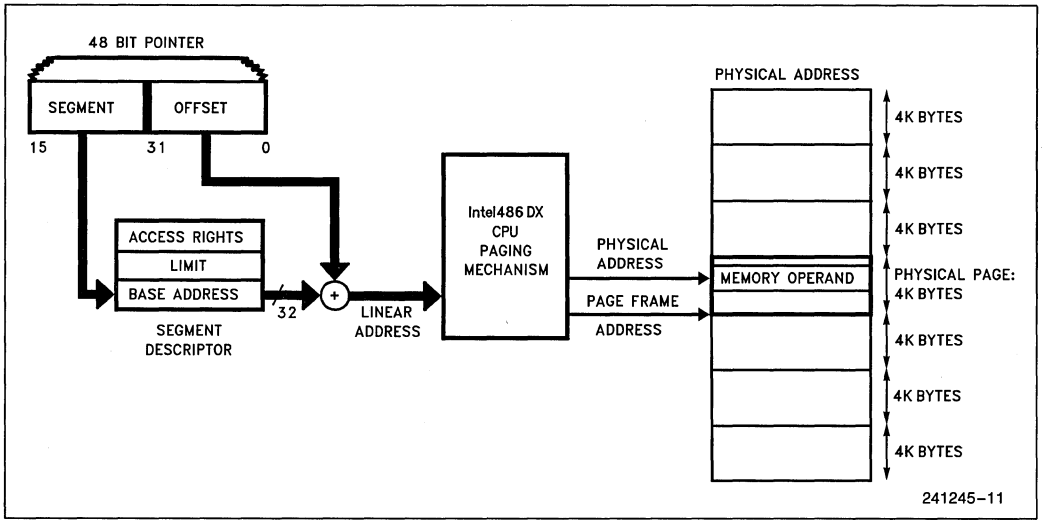


Figure 4.2. Paging and Segmentation

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 a Intel486 microprocessor system. There are three types of tables on the Intel486 DX2 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.

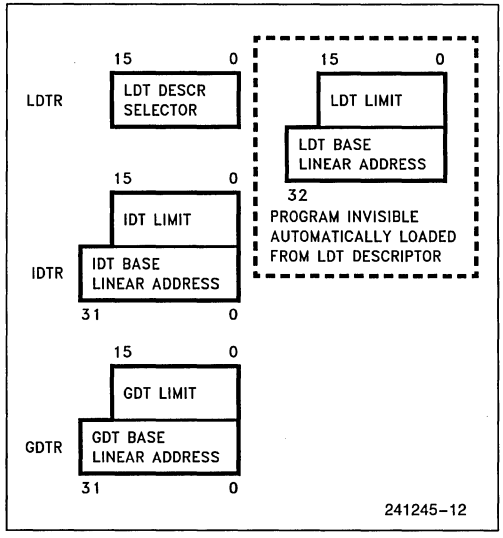


Figure 4.3. Descriptor Table Registers

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 Intel486 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 provides 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 Intel486 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).**

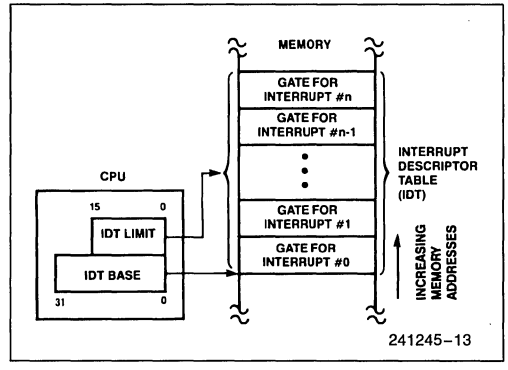


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

4.3.4.1 Descriptor Attribute Bits

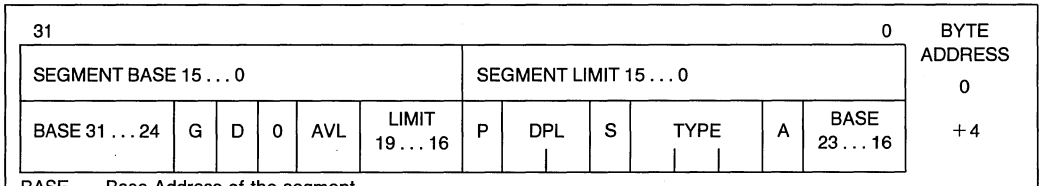
The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These

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 Intel486 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 Intel486 DX 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 Intel486 CPU Code, Data Descriptors (S=1)

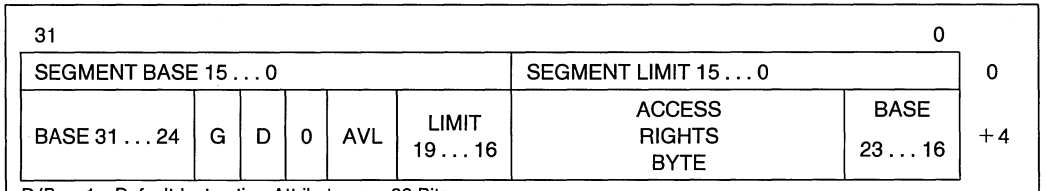
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



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. Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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. Intel486 DX 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)

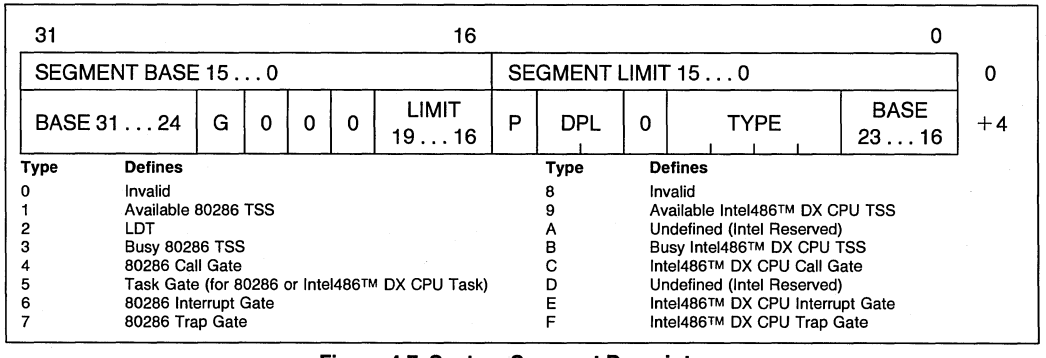
A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used 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 an Intel486 DX 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.

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.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Intel486™ DX CPU TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy Intel486™ DX CPU TSS
4	80286 Call Gate	C	Intel486™ DX CPU Call Gate
5	Task Gate (for 80286 or Intel486™ DX CPU Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	Intel486™ DX CPU Interrupt Gate
7	80286 Trap Gate	F	Intel486™ DX CPU Trap Gate

Figure 4.7. System Segment Descriptors

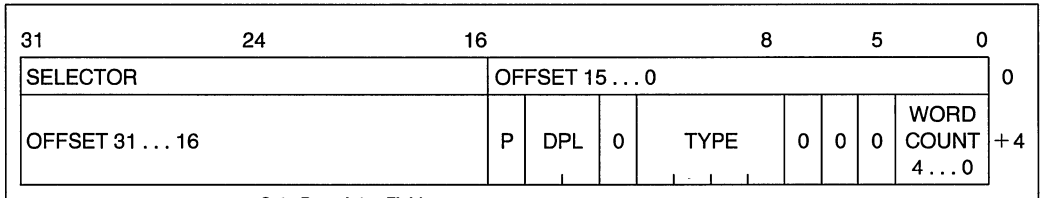
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

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 Intel486 DX2 Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Intel486 DX2 microprocessor, the Intel486 DX 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 Intel486 DX microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the Intel486 DX microprocessors. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Intel486 DX microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



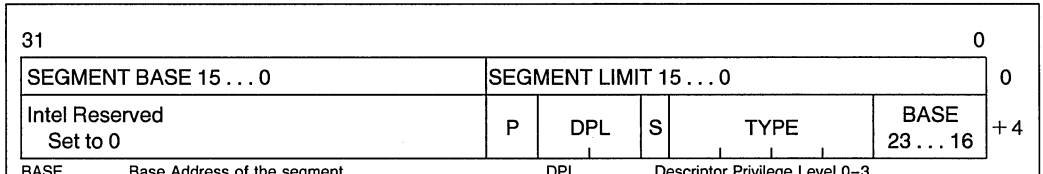
Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or Intel486™ DX CPU task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	Intel486™ DX CPU call gate
	E	Intel486™ DX CPU interrupt gate
	F	Intel486™ DX 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 Intel486™ DX 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 Intel486™ DX CPU	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats



BASE	Base Address of the segment	DPL	Descriptor Privilege Level 0–3
LIMIT	The length of the segment	S	System Descriptor 0=System 1=User
P	Present Bit 1=Present 0=Not Present	TYPE	Type of Segment

Figure 4.9. 80286 Code and Data Segment Descriptors

By supporting 80286 system segments the Intel486 DX microprocessor is able to execute 80286 application programs on an Intel486 DX microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are Intel486 DX 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 Intel486 DX microprocessor-style 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 Intel486 DX 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.

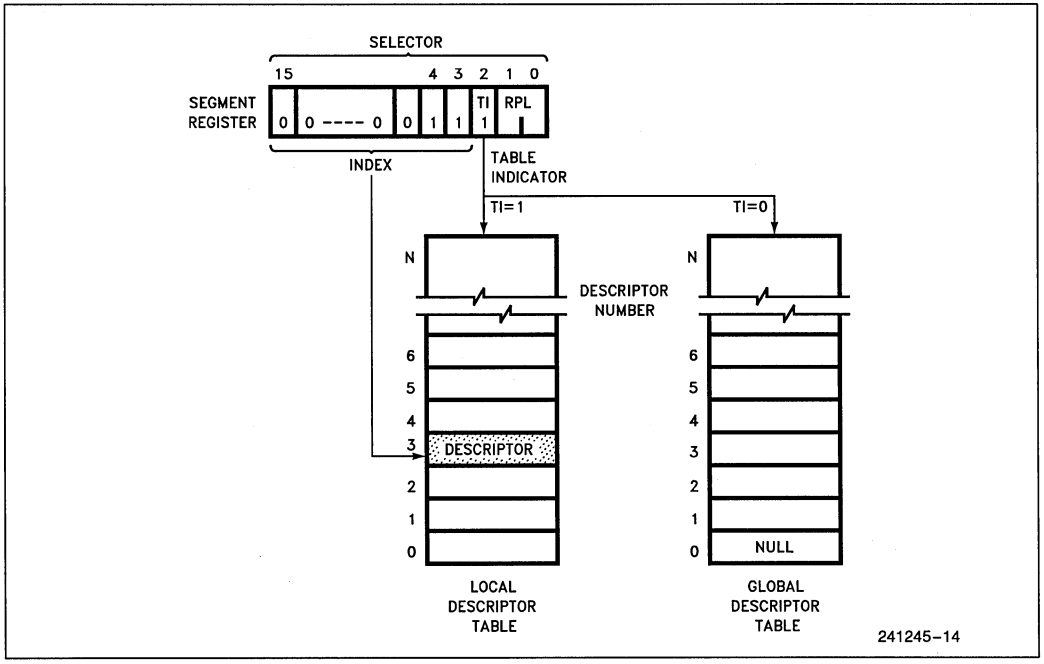
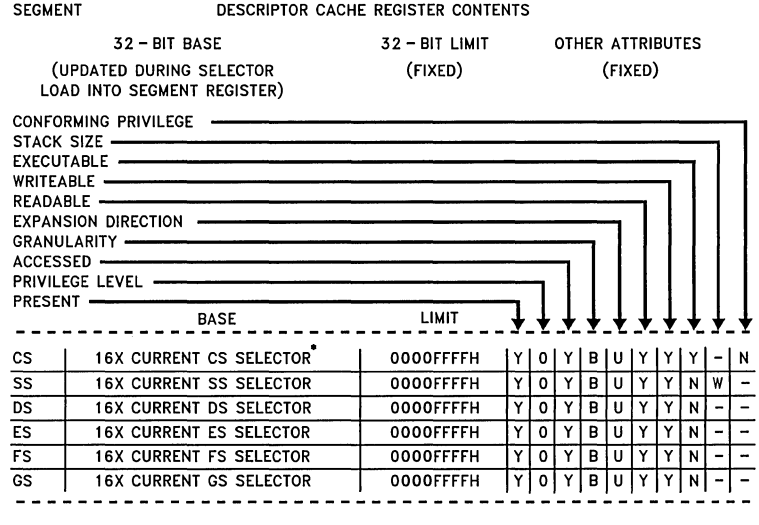


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Intel486 DX 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.

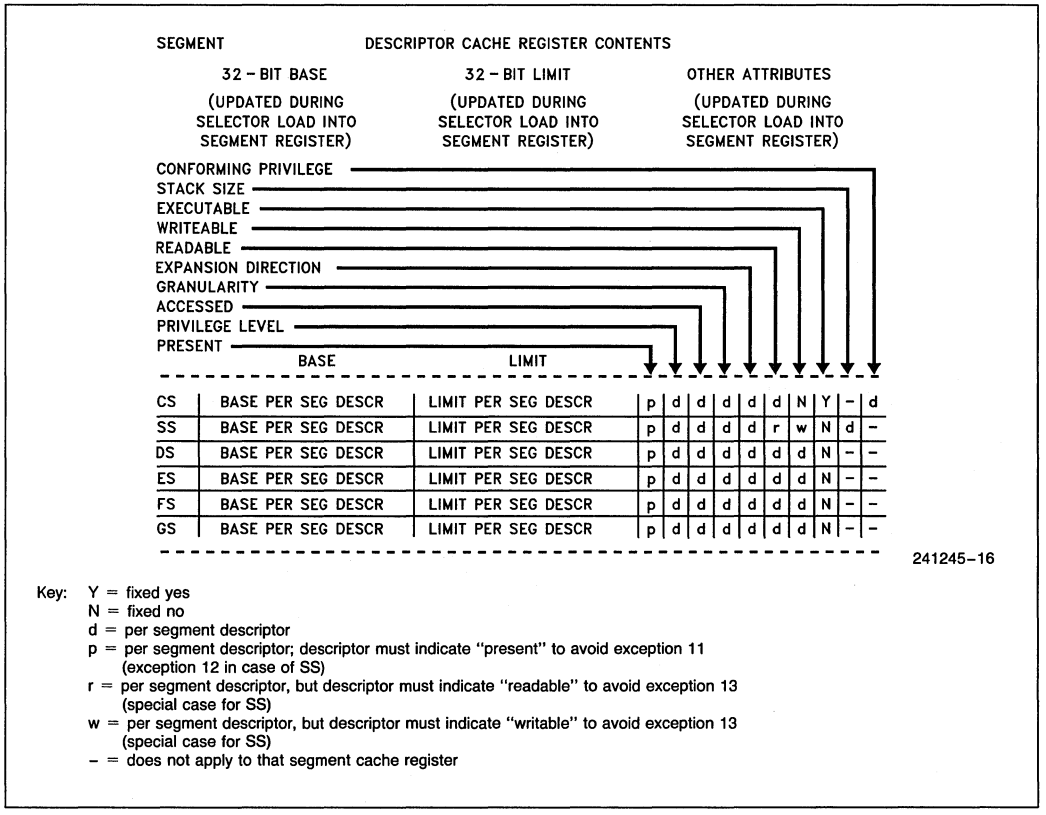


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

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)

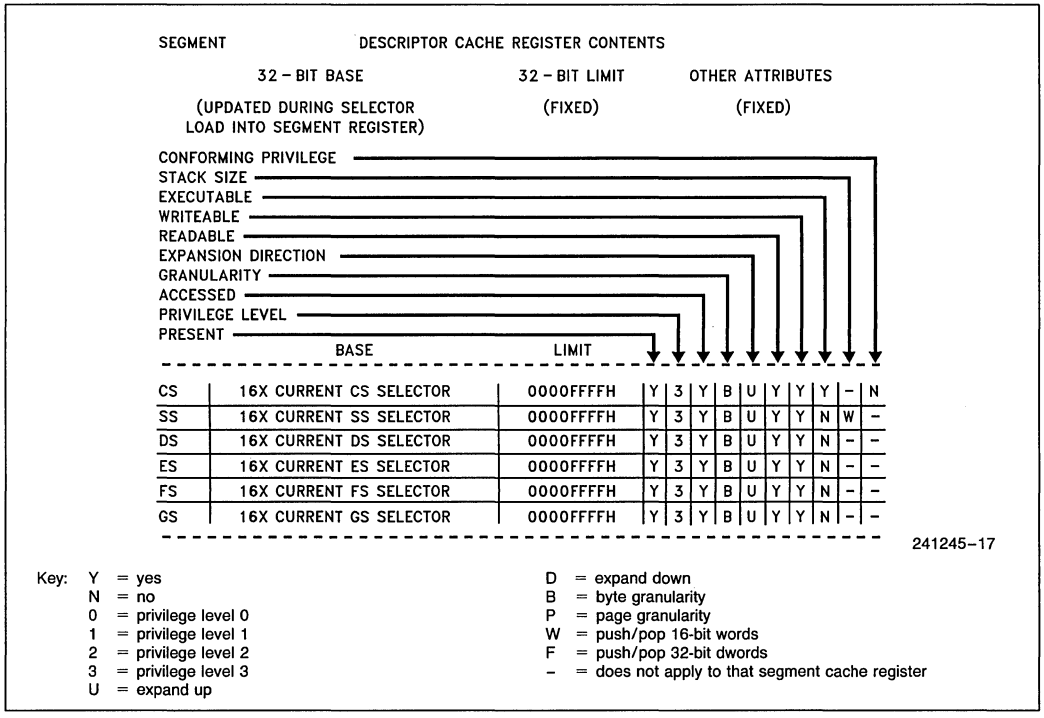
When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the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

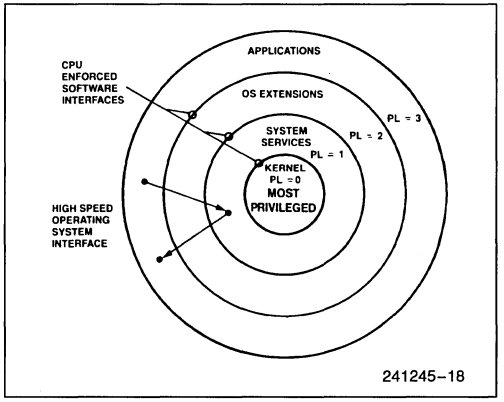


Figure 4.14. Four-Level Hierarchical Protection

The Intel486 DX 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 Intel486 DX Microprocessor provides the protection as part of its integrated Memory Management Unit. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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

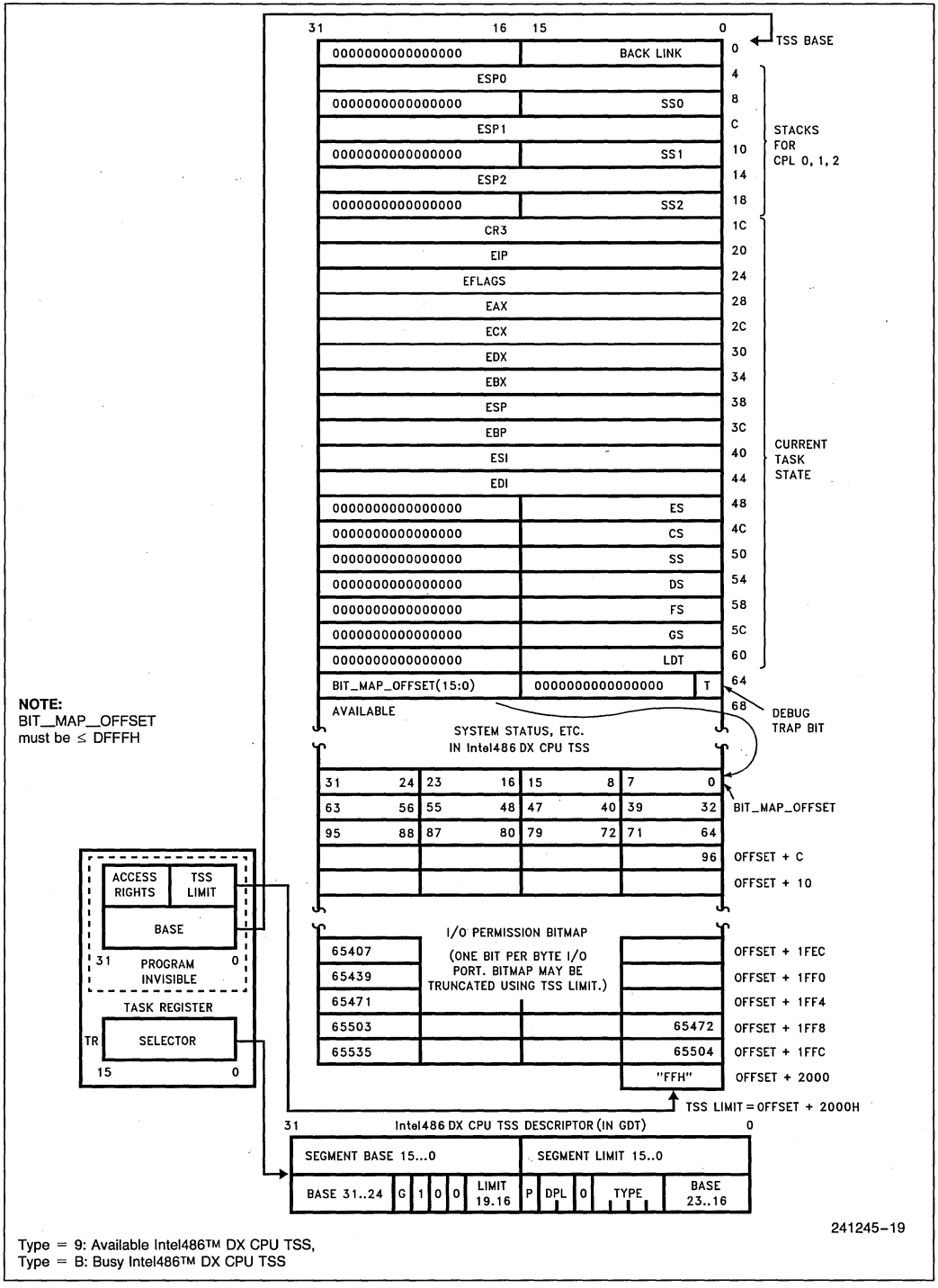
At any point in time, a task on the Intel486 DX Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different 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 an Intel486 DX Microprocessor TSS, the I/O Permission Bitmap (part of an Intel486 DX 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.



Type = 9: Available Intel486™ DX CPU TSS,
Type = B: Busy Intel486™ DX CPU TSS

Figure 4.15a. Intel486™ DX Microprocessor TSS and TSS Registers

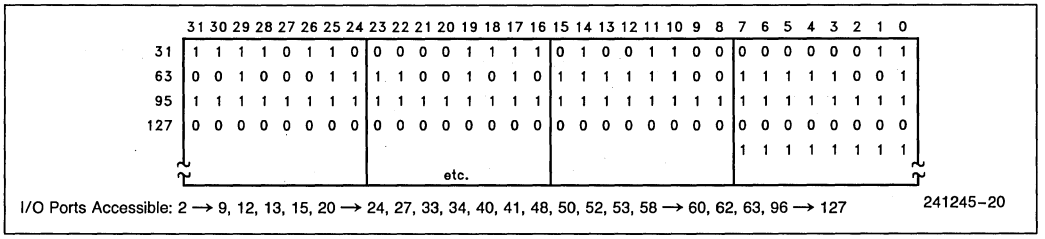


Figure 4.15b. Sample I/O Permission Bit Map

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.

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 Intel486 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.

4.4.3.4 Privilege Validation

The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 seg-

ments. 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.

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.

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

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.

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 Intel486 DX 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 Intel486 DX microprocessor directly supports this operation by providing a task switch instruction in hardware. The Intel486 DX 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 10 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 Intel486 DX microprocessor execution state while a task gate descriptor contains a TSS selector. The Intel486 DX microprocessor supports both 80286 and Intel486 DX microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of an Intel486 DX 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 Intel486 DX 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:

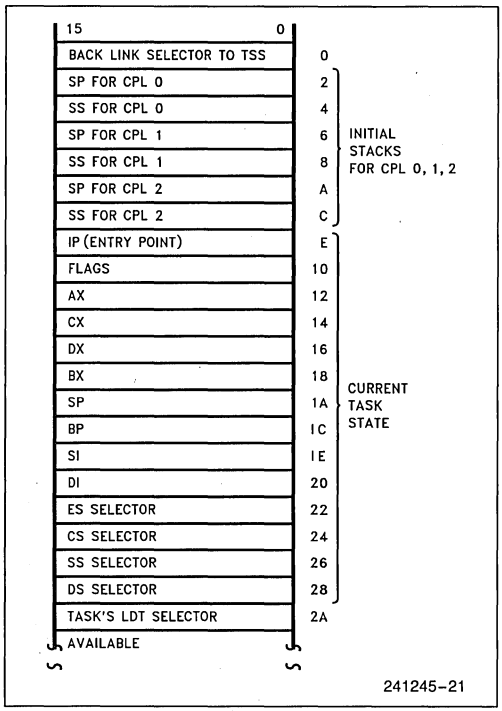


Figure 4.16. 80286 TSS

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 Intel486 DX 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 Intel486 DX Microprocessor switches tasks, it sets the TS bit. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

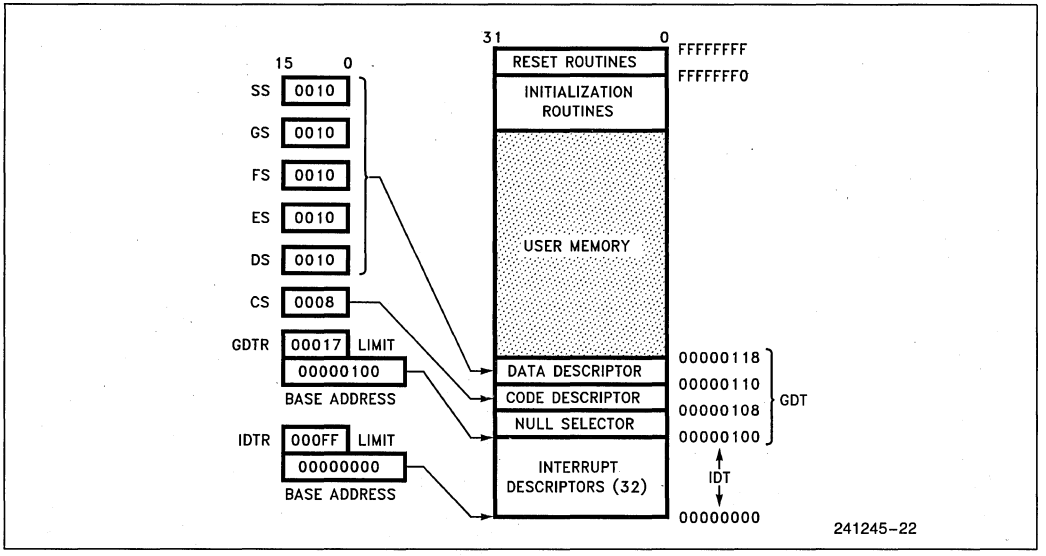


Figure 4.17. Simple Protected System

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Intel486 DX 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.

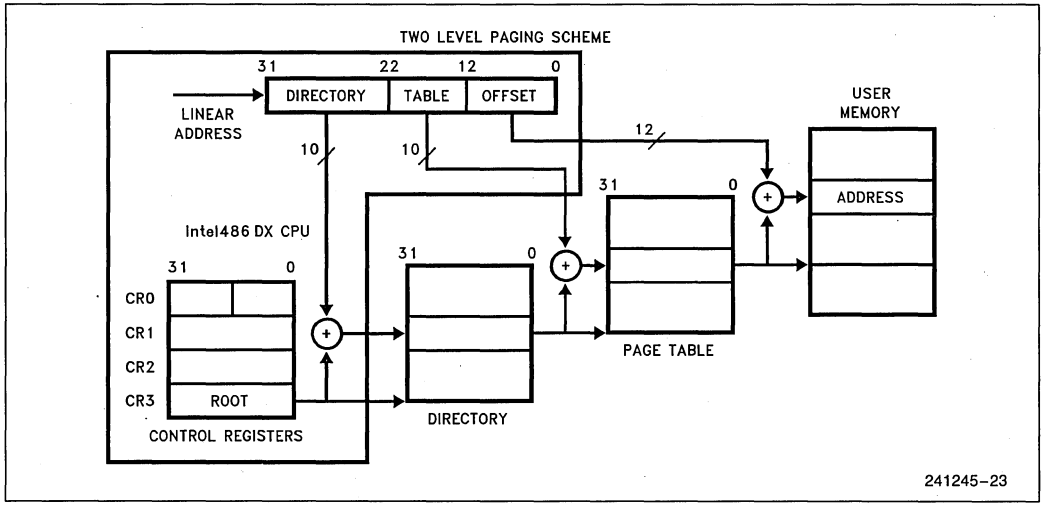


Figure 4.19. Paging Mechanism

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel386 microprocessor does not contain the WP bit. The WP bit has been added to the Intel486 DX microprocessor to protect read-only pages from supervisor write accesses. The Intel386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the Intel386 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 Intel486 DX 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 CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (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. However, when paging is disabled (PG = 0 in CR0) or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and HALT cycles), the PCD and PWT bits of CR3 are ignored. The Intel486 DX CPU assumes PCD = 0 and PWT = 0 and drives these values on the PCD and PWT pins.

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.

The PCD or PWT bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The Intel486 DX 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 Intel486 DX 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 Intel486 DX Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX on-chip cache just like normal data.

4.5.6 PAGING OPERATION

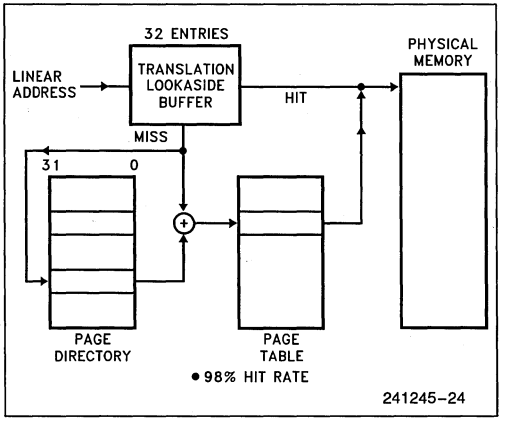


Figure 4.22. Translation Lookaside Buffer

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

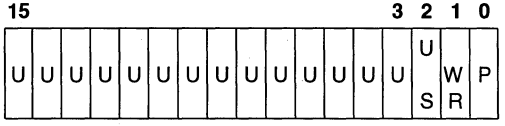


Figure 4.23a. Page Fault Error Code Format

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX Microprocessor protection mechanism. In particular, the Intel486 DX Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and an Intel486 DX Microprocessor operating system and both 80286 and Intel486 DX Microprocessor applications. Thus, in a multi-user Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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;
```

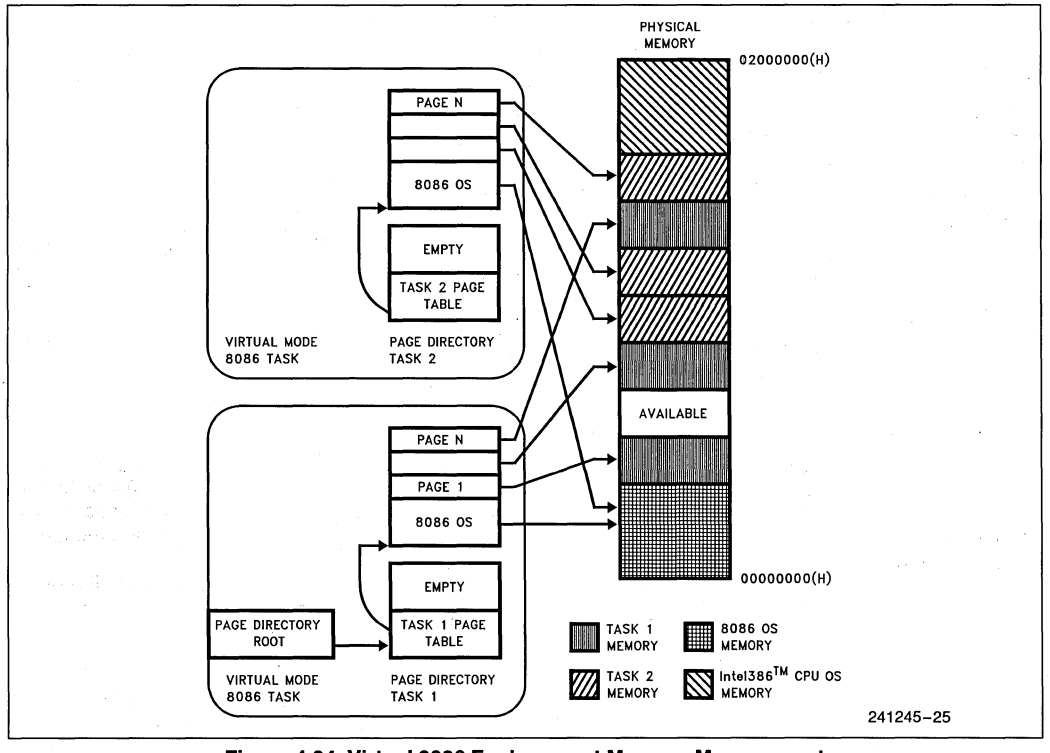


Figure 4.24. Virtual 8086 Environment Memory Management

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 **Intel486 DX Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the Intel486 DX 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 Intel486 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 Intel486 DX microprocessor operating system. The Intel486 DX 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 Intel486 DX operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Intel486 DX 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 Intel486 DX operating system. The Intel486 DX operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the Intel486 DX operating system could intercept an 8086 operating system's call to "Open a File".

A Intel486 DX 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 Intel486 DX task whose Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 an Intel486 DX 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.

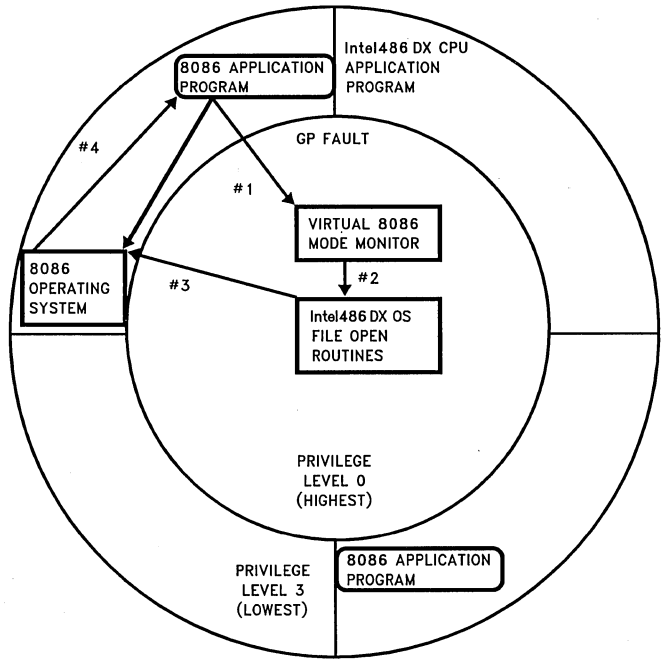
A task switch into a task described by a Intel486 DX TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a Intel486 DX 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 Intel486 DX Microprocessor Trap Gate (Type 14), or Intel486 DX 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. Intel486 DX 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 Intel486 DX 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 Intel486™ DX OS (Arrow #2)
Intel486™ DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX2 microprocessor contains an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the Intel386™/Intel486™ 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.

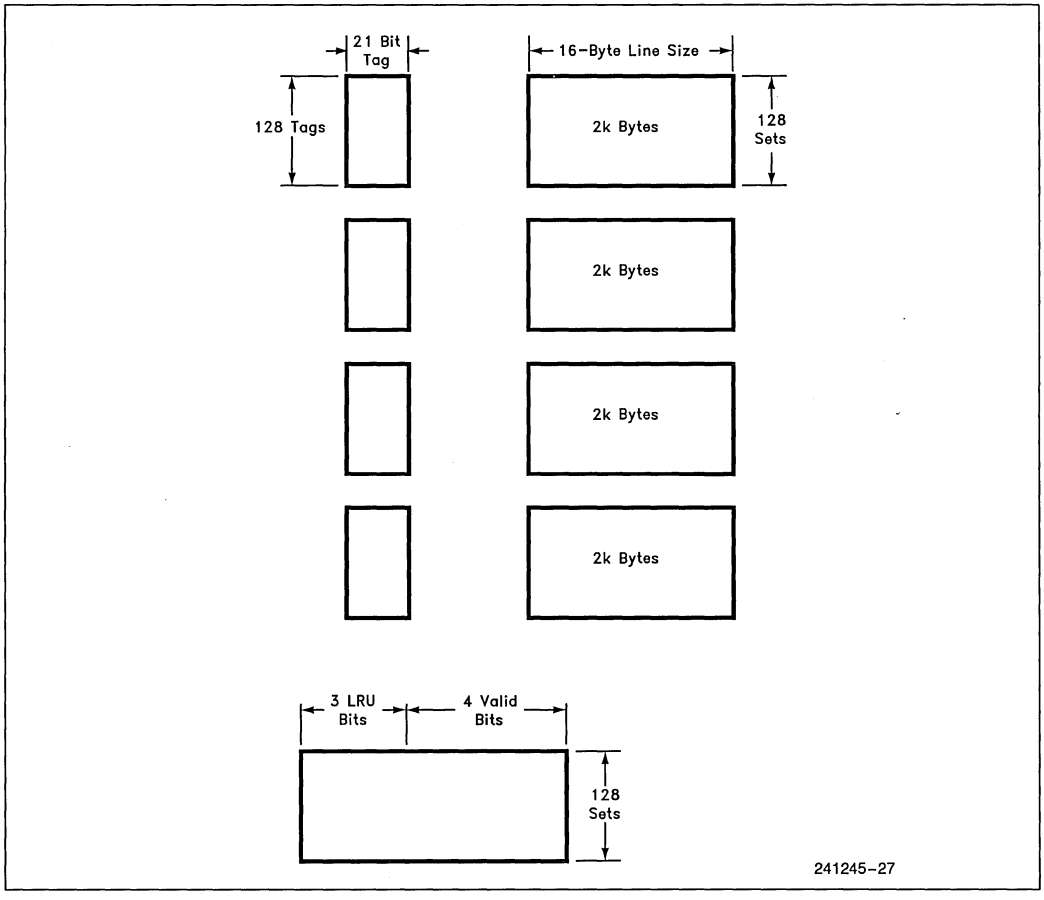


Figure 5.1. On-Chip Cache Physical Organization

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 CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW 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.

Table 5.1. Cache Operating Modes

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

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 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 CD = 1 and NW = 1. 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.

CD = 1, NW = 0

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.

CD = 0, NW = 1

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.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 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 Intel486 DX microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Intel486 DX 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 Intel486 DX microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the Intel486 DX 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 Intel486 DX microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Intel486 DX 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 Intel486 DX 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.

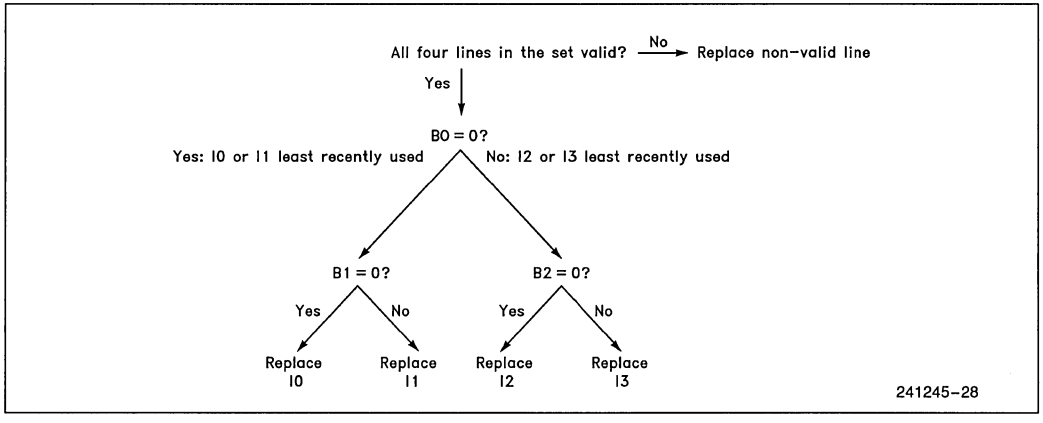


Figure 5.2. On-Chip Cache Replacement Strategy

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 Intel486 DX 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 Intel486 DX 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.

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 Intel486 DX microprocessor if the area is cacheable. The Intel486 DX microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

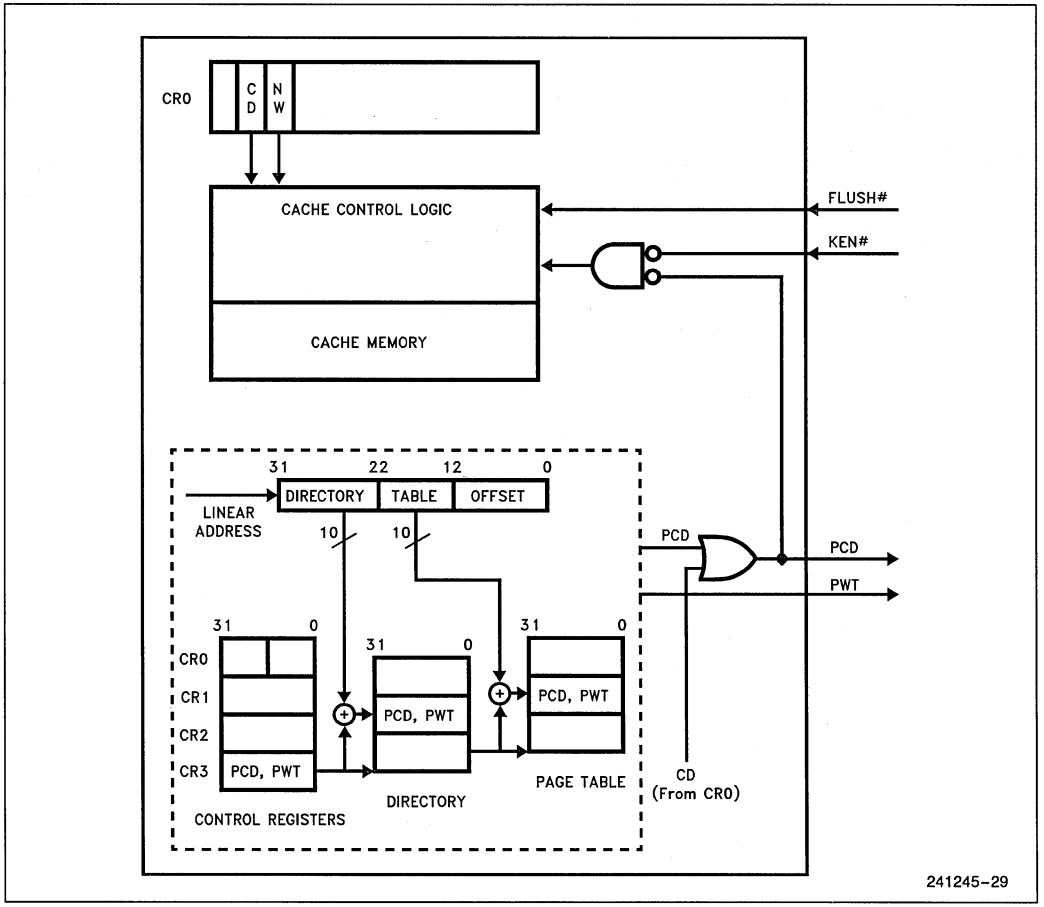


Figure 5.3. Page Cacheability

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD=1 the Intel486 DX microprocessor forces the PCD pin HIGH. If CD=0 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. These bits are assumed to be zero during real mode, whenever paging is disabled, 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 flush pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. The flush input is asynchronous but setup and hold times must be met. The flush pin should be deasserted after the cache flush is complete. Failure to deassert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, flush must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the CPU begins execution of the instruction following the OUT instruction.

Flush is recognized during HOLD just like EADS#.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the Intel486 DX 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 Intel486 DX microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the Intel486 DX 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 Intel486 DX microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The Intel486 DX microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the Intel386 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 Intel486 DX 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 Intel486 DX2 microprocessor bus has been designed to be identical with the Intel486 microprocessor bus. Several new features have been added to the Intel486 DX2 to increase performance and testability. New features include a speed doubler for the core logic, and IEEE 1149.1 boundary scan support.

The Intel486 DX2 is driven by what can be called a ½x clock, as opposed to the 1x clock in the Intel486 DX and the 2x clock in the Intel386 microprocessors. Thus a 50 MHz Intel486 DX2 is driven by a 25 MHz clock, in contrast with 25 MHz processors like the Intel486 DX2 and the Intel386 requiring 25 MHz and 50 MHz, respectively. Since the Intel486 DX has a clock doubler driving its core, but not its bus interface, it provides a simpler system design for a given performance level than the Intel486 DX. In reality, this permits dramatic increases in performance by just plugging in the Intel486 DX2 in a system that had already been designed for the Intel486. This performance is supplied because the bus interface is identical to that of an Intel486 DX and all of the core is running at twice the speed of the comparable Intel486 DX. This speedup includes the internal cache memory, the floating point unit, the instruction decode unit, the ALU and everything except the bus interface.

Like the Intel386 microprocessor, the Intel486 DX 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 Intel386 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.

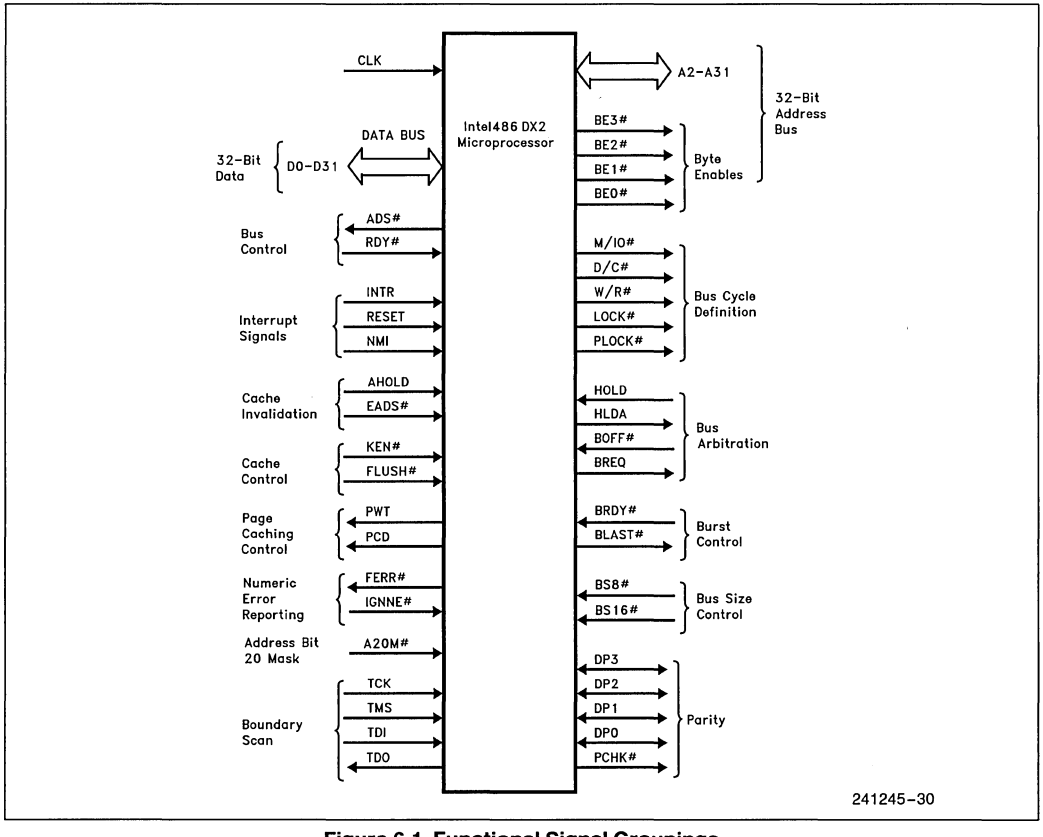


Figure 6.1. Functional Signal Groupings

The Intel486 DX 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 Intel486 DX microprocessor has a bus hold feature similar to that of the Intel386 microprocessor. During bus hold, the Intel486 DX microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The Intel486 DX 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 Intel486 DX microprocessor input and output signals arranged by functional 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 Intel486 DX microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

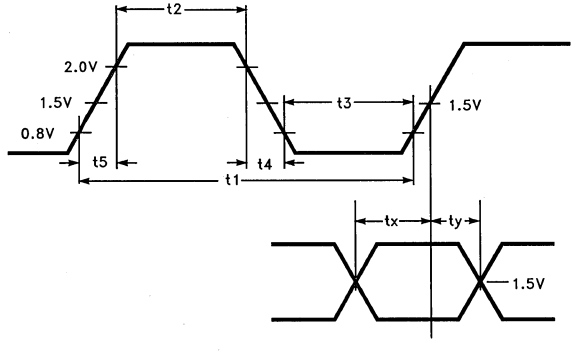
CLK provides the fundamental timing and the internal operating frequency for the Intel486 DX2 microprocessor. All external timing parameters are specified with respect to the **rising edge** of CLK.

The Intel486 DX2 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 four 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 Intel486 DX 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 Intel486 DX 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.



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

241245-31

Figure 6.2. CLK waveform

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 Intel486 DX 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.

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 Intel486 DX microprocessor. Even parity information must be driven back to the Intel486 DX microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor. The Intel486 DX 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 Intel486 DX microprocessor and Intel386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Intel486 DX microprocessor from location 101 in the Intel386 microprocessor. Location 101 is now reserved and will never be generated by the Intel486 DX microprocessor.

Table 6.1. ADS# 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX2 microprocessor to begin execution at a known state. **For a power-up (cold start) reset, V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms** before the Intel486 DX2 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper Intel486 DX2 microprocessor operation. However, **for a warm boot-up case, RESET is required to remain active for a minimum of 15 clocks.** 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.)

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 Intel486 DX 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 Intel486 DX microprocessor asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the Intel486 DX microprocessor is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. 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. During a hold state, internal events may cause BREQ to be deasserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Intel486 DX microprocessor bus. The Intel486

DX 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. In addition, if the Intel486 DX CPU receives a HOLD request while performing a non-cacheable, non-burst-ed code prefetch and that cycle is backed off (BOFF#), the Intel486 DX CPU will recognize HOLD before restarting the cycle. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The Intel486 DX 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 Intel386 microprocessor, the **Intel486 DX 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 pull-down 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 Intel486 DX 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 bus clock cycle that the Intel486 DX microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the Intel486 DX microprocessor will resume driving the bus. The Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor completes the current bus cycle before floating its bus in response to HOLD. Second the Intel486 DX does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the Intel486 DX 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 Intel486 DX microprocessor will float its bus in the next bus 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the Intel486 DX 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 Intel486 DX microprocessor will not initiate another bus cycle during address hold. Since the Intel486 DX microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the Intel486 DX will redrive the same address (that it originally sent out) once AHOLD is negated.

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 pull-down 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 Intel486 DX 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 Intel486 DX 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 ad-

dress 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 3-state test mode of the Intel486 DX 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 correspond to bits 3 and 4 of the page table entry respectively. 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. When paging is disabled, the Intel486 DX CPU ignores the PCD and PWT bits and assumes they are zero for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD = 1 (cache line fills disabled) the Intel486 DX microprocessor forces PCD HIGH. When CD = 0, 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 Intel486 DX 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. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. 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 Intel486 DX microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The Intel486 DX microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the Intel387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in Intel486 DX microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

The Intel486 DX microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, but FERR# will still be activated. When deasserted, the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor emulates the 1 Mbyte address wrap-around that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should be sampled high 2 clocks before and 2 clocks after RESET goes low.

6.2.16 BOUNDARY SCAN TEST SIGNALS

Test Clock (TCK)

TCK is an input to the Intel486 DX2 CPU and provides the clocking function required by the JTAG boundary scan feature. TCK is used to clock state information and data into and out of the component. State select information and data are clocked into the component on the rising edge of TCK on TMS and TDI, respectively. Data is clocked out of the part on the falling edge of TCK on TDO.

In addition to using TCK as a free running clock, it may be stopped in a low, O, state, indefinitely as described in IEEE 1149.1. While TCK is stopped in the low state, the boundary scan latches retain their state.

When boundary scan is not used, TCK should be tied high or left as a NC (This is important during power up to avoid the possibility of glitches on the TCK which could prematurely initiate boundary scan operations). TCK is supplied with an internal pullup resistor.

TCK is a clock signal and is used as a reference for sampling other JTAG signals. On the rising edge of TCK, TMS and TDI are sampled. On the falling edge of TCK, TDO is driven.

Test Mode Select (TMS)

TMS is decoded by the JTAG TAP (Tap Access Port) to select the operation of the test logic, as described in Section 8.5.4.

To guarantee deterministic behavior of the TAP controller, TMS is provided with an internal pull-up resistor. If boundary scan is not used, TMS may be tied high or left unconnected. TMS is sampled on the rising edge of TCK. TMS is used to select the internal TAP states required to load boundary scan instructions to data on TDI. For proper initialization of the JTAG logic, TMS should be driven high, "1", for at least four TCK cycles following the rising edge of RESET.

Test Data Input (TDI)

TDI is the serial input used to shift JTAG instructions and data into the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4.

An internal pull-up resistor is provided on TDI to ensure a known logic state if an open circuit occurs on the TDI path. Note that when "1" is continuously shifted into the instruction register, the BYPASS instruction is selected. TDI is sampled on the rising edge of TCK, during the SHIFT-IR and the SHIFT-DR states. During all other TAP controller states, TDI is a "don't care".

Test Data Output (TDO)

TDO is the serial output used to shift JTAG instructions and data out of the component. The shifting of instructions and data occurs during the SHIFT-IR and SHIFT-DR TAP controller states, respectively. These states are selected using the TMS signal as described in Section 8.5.4. When not in SHIFT-IR or SHIFT-DR state, TDO is driven to a high impedance state to allow connecting TDO of different devices in parallel.

TDO is driven on the falling edge of TCK during the SHIFT-IR and SHIFT-DR TAP controller states. At all other times TDO is driven to the high impedance state.

6.3 Write Buffers

The Intel486 DX2 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 Intel486 DX 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. Reordering read once only maintains cache consistency. Consider the following example: The CPU writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 6.3(a)). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

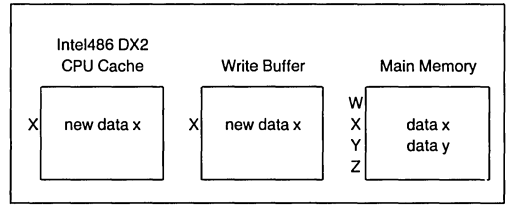


Figure 6.3(a)

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Since the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 6.3(b)).

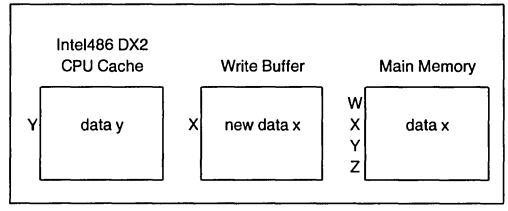


Figure 6.3(b)

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Since one of the conditions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propogate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

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 Intel486 DX microprocessor will update all memory locations before reading status from an I/O device.

The Intel486 DX 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 Intel486 DX microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the Intel486 DX microprocessor than with the Intel386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the Intel386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the Intel386 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 Intel486 DX 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 Intel486 DX microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the Intel486 DX microprocessor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the Intel486 DX microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 synchronizer like that used on the INTR line. Other than the 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 four 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 Intel486 DX microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted for 2 clocks before and 2 clocks after RESET is deasserted. RESET must be active for 15 clocks with or with no BIST being enabled. Refer to Section 8.0 for information on Intel486 DX microprocessor testability.

The Intel486 DX microprocessor registers have the values shown in Table 1.5 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain a stepping identifier (see Table 1.5). 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	0400 + Revision ID	0400 + 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	60000010h	60000010h
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

Table 6.3. Component and Revision ID

Intel486 DX2 CPU Stepping Name	Component ID	Revision ID
A	04	32
B	04	33

The Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The Intel486 DX2 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 Intel486 DX microprocessor bus is in the state shown in Figure 6.4 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. The floating point unit's status word register can be initialized by BIST or by executing FINIT/FNINIT instruction. Thus, after reset and before executing the first FINIT or FNINIT instruction, the values of the FERR# and the numeric status word register bits 0–7 depends on whether or not BIST is performed. Table 6-4 shows the state of FERR# signal after reset and before the execution of the FINIT/FNINIT instruction.

Table 6-4

BIST Performed	FERR# Pin	FPU Status Word Register Bits 0–7
YES	Inactive (High)	Inactive (Low)
NO	Undefined (Low or High)	Undefined (Low or High)

After the first FINIT or FNINIT instruction, FERR# pin and the FPU status word register bits (0–7) will be inactive irrespective of the Built-In Self-Test (BIST).

Power Down Mode (Upgrade Processor Support)

The Power Down Mode on the Intel486 DX2 microprocessor, when initiated by the upgrade processor, reduces the power consumption of the Intel486 DX2 CPU (see Table 14-2 D.C. Specifications), as well as forces all of its output signals to be 3-stated. The UP# pin on the Intel486 DX2 microprocessor is used for enabling the Power Down Mode.

Once the UP# pin is driven active by the upgrade processor upon power-up, the Intel486 DX2 microprocessor's bus is floated immediately. The Intel486 DX2 CPU enters the Power Down Mode when the UP# pin is sampled asserted in the clock before the falling edge of RESET. The UP# pin has no effect on the power down status, except during this edge. The Intel486 DX2 CPU then remains in the Power Down Mode until the next time the RESET signal is activated. For warm resets, with the upgrade processor in the system, the Intel486 DX2 CPU will remain 3-stated and re-enter the Power Down Mode once RESET is de-asserted. Similarly for power-up resets, if the upgrade processor is not taken out of the system, the Intel486 DX2 CPU will 3-state its outputs upon sensing the UP# pin active and enter the Power Down Mode after the falling edge of RESET.

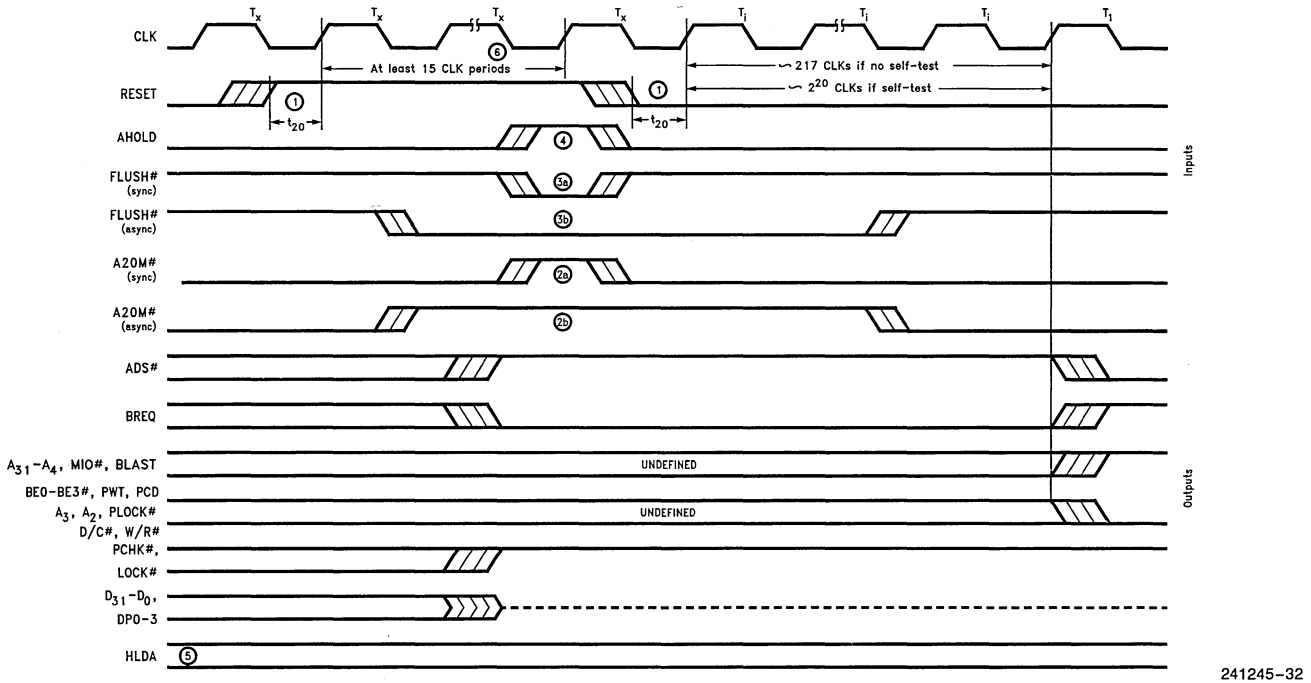


Figure 6.4. Pin States during RESET

NOTES:

1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.

2a. When A20M# is driven synchronously, it must be driven high (inactive) for the CLK edge prior to the falling edge of RESET to ensure proper operation. A20M# setup and hold times must be met.

2b. When A20M# is driven asynchronously, it must be driven high (inactive) for two CLKs prior to and two CLKs after the falling edge of RESET to ensure proper operation.

3a. When FLUSH# is driven synchronously, it should be driven low (active) for the CLK edge prior to the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESET being deasserted. FLUSH# setup and hold times must be met.

3b. When FLUSH# is driven asynchronously, it must be driven low (active) for two CLKs prior to and two CLKs after the falling edge of RESET to invoke the 3-state Output Test Mode. All outputs are guaranteed 3-stated within 10 CLKs of RESET being deasserted.

4. AHOLD should be driven high (active) for the CLK edge prior to the falling edge of RESET to invoke the Built-In-Self-Test (BIST). AHOLD setup and hold times must be met.

5. Hold is recognized normally during RESET.

6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

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 Intel486 DX 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

Intel486™ DX 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.

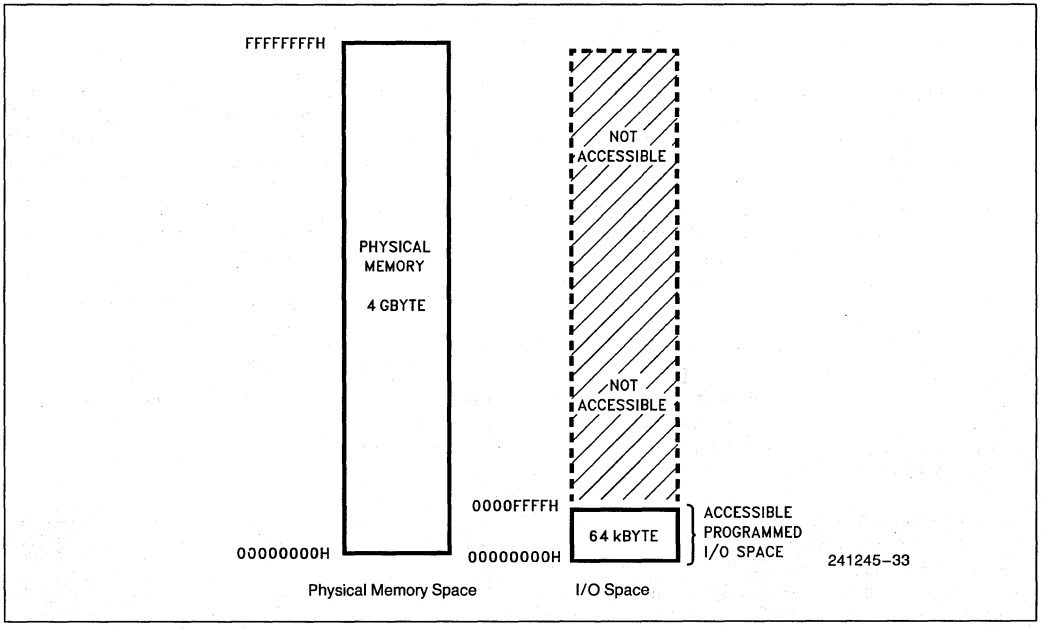


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The Intel486 DX microprocessor data path 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 Intel486 DX 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.

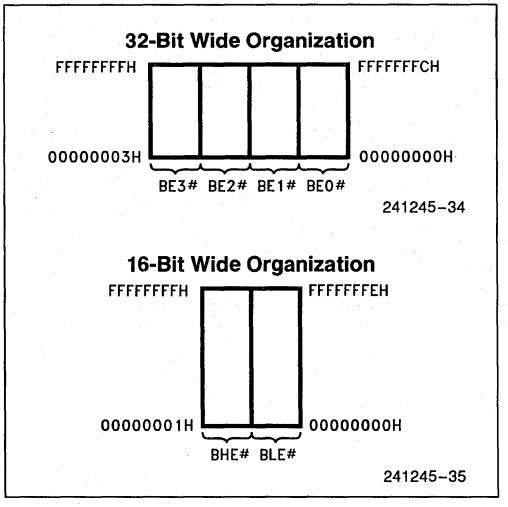


Figure 7.2. Physical Memory and I/O Space Organization

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 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor is significantly different than that of the Intel386 microprocessor. Unlike the Intel386 microprocessor, the Intel486 DX 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 Intel486 DX microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The Intel486 DX microprocessor expects the two low order bytes on D0–D15. The Intel386 microprocessor expects both the high and low order bytes on D0–D15. The Intel386 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 Intel486 DX microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the Intel486 DX 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 will be driven but they will not contain valid data. Unlike the Intel386 microprocessor, the Intel486 DX 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.

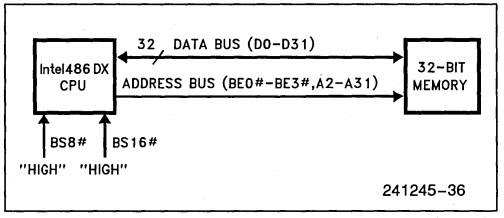


Figure 7.3. Intel486™ DX Microprocessor with 32-Bit Memory

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 Intel486 DX 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.

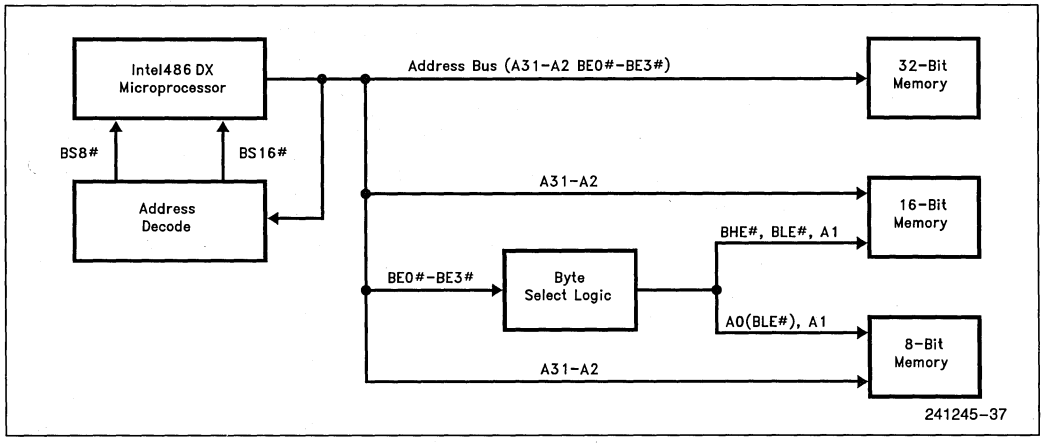


Figure 7.4. Addressing 16- and 8-Bit Memories

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

Intel486™ DX 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

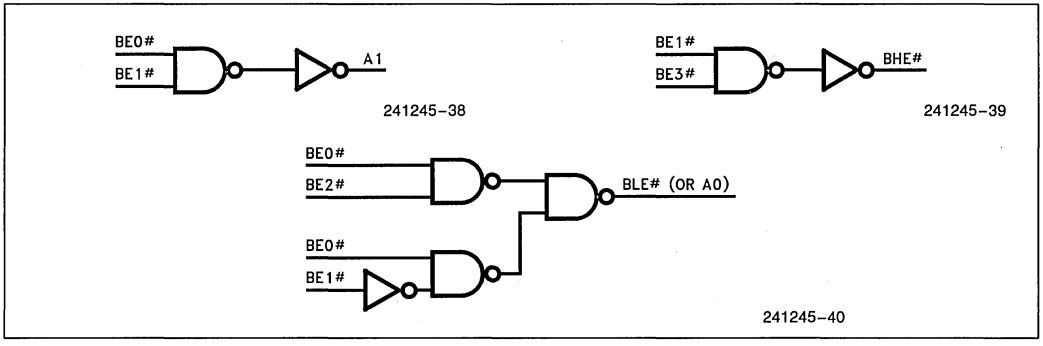


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

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 Intel486 DX 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 Intel486 DX microprocessor on the correct data pins (see Table 7.4).

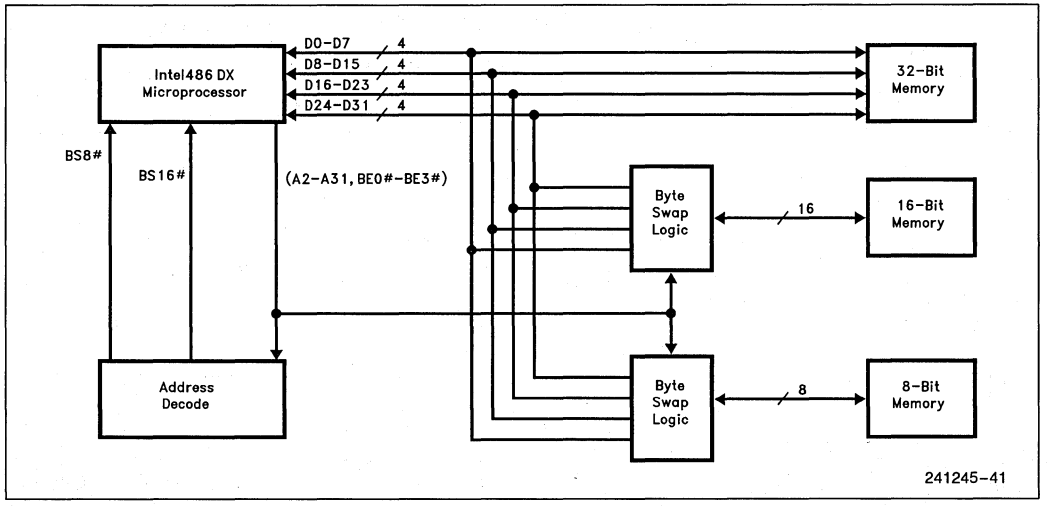


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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.

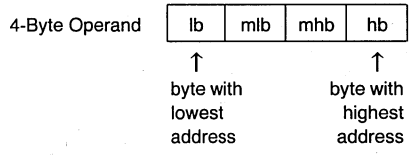
Table 7.6. Generating A0, A1 and BHE# from the Intel486™ DX Microprocessor Byte Enables

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor samples RDY# at the end of the second 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 an Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Intel486 DX microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Intel486 DX microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The Intel486 DX 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 Intel486 DX microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. 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 Intel486 DX microprocessor internal cache lines.

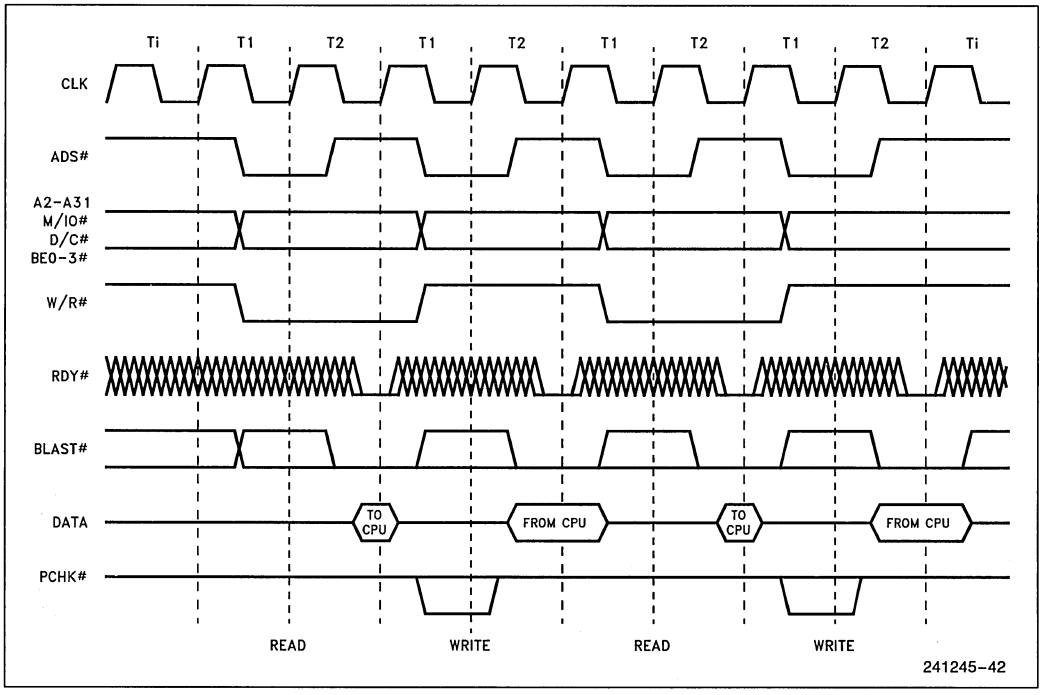


Figure 7.7. Basic 2-2 Bus Cycle

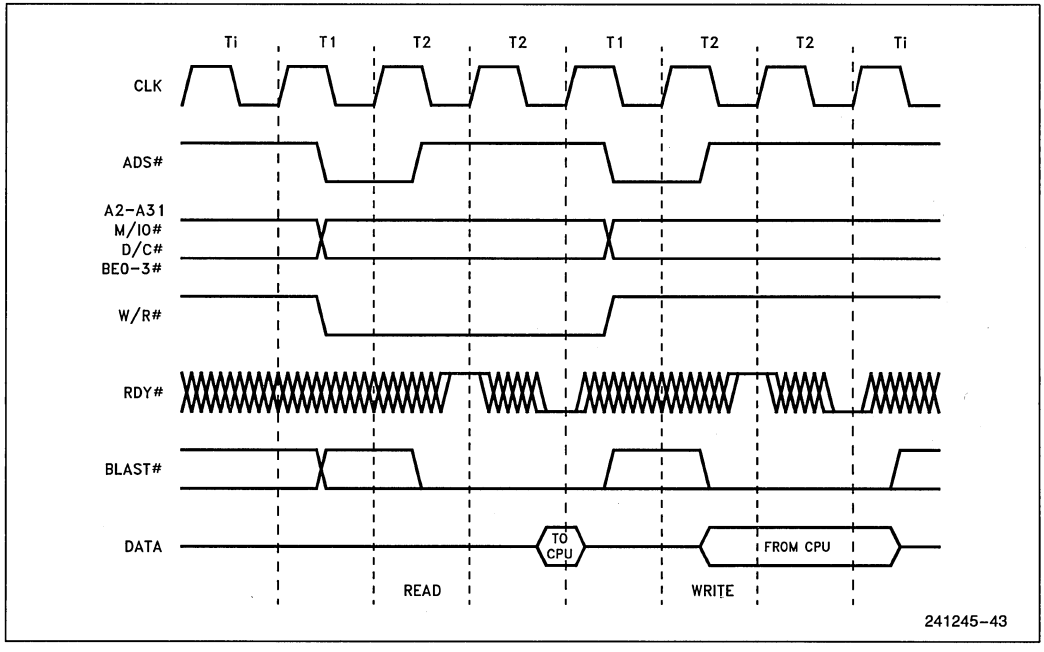


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Intel486 DX microprocessor can be converted into a burst cycle. The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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#.

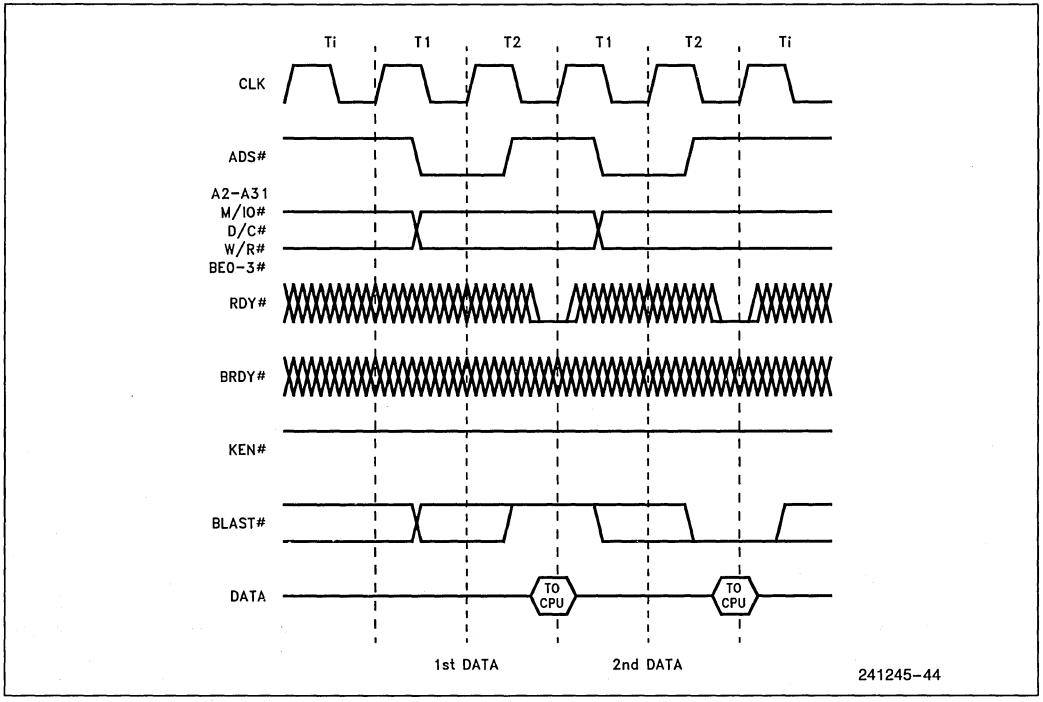


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

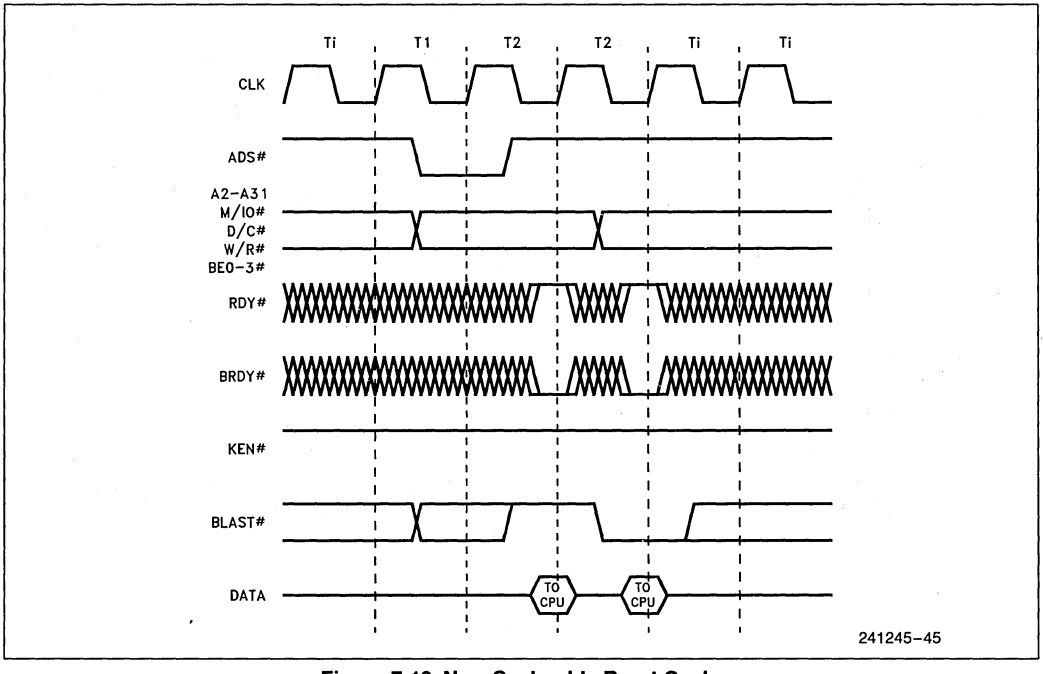


Figure 7.10. Non-Cacheable Burst Cycle

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 Intel486 DX 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 Intel486 DX 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 disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the Intel486 DX microprocessor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Intel486 DX drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor samples KEN# active at the end of the first clock. The Intel486 DX 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. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

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 Intel486 DX microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

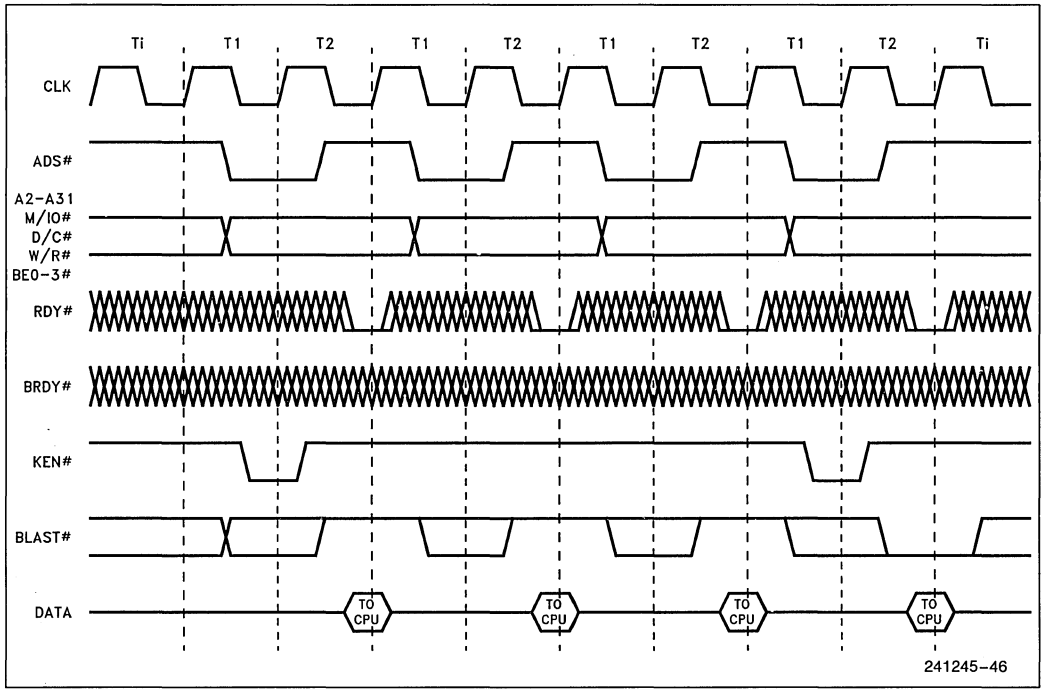


Figure 7.11. Non-Burst, Cacheable Cycles

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 Intel486 DX 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.

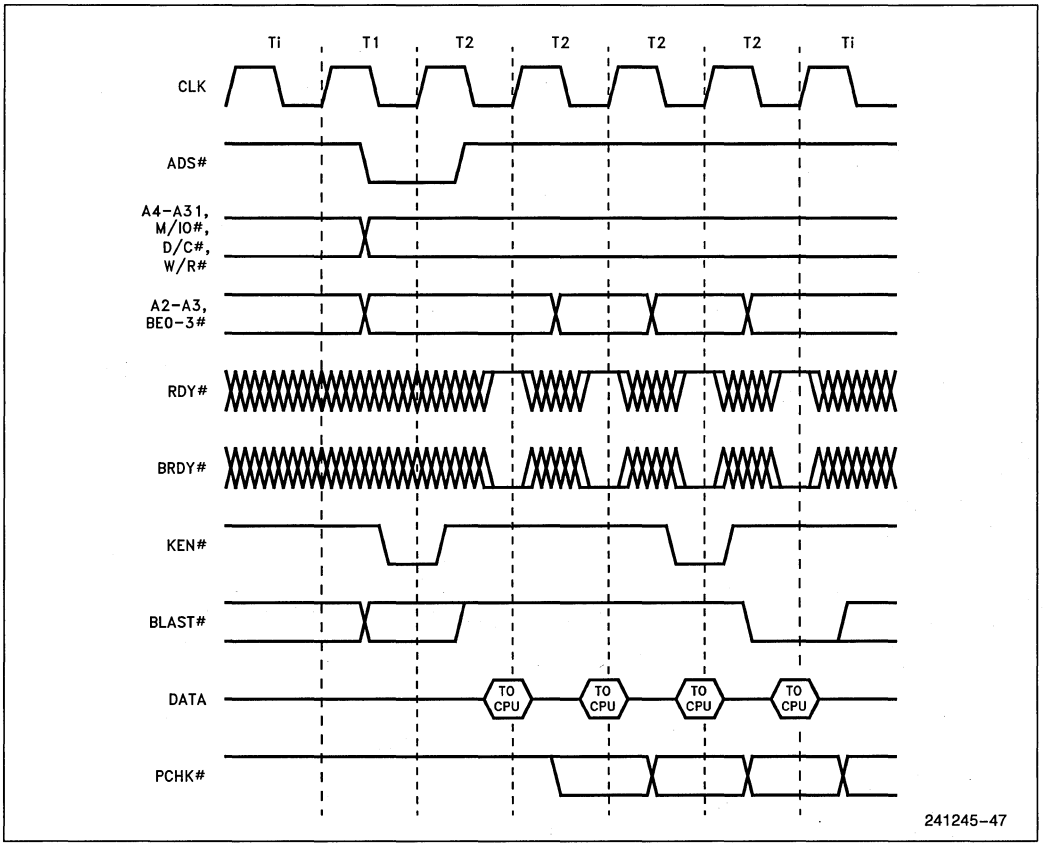


Figure 7.12. Burst Cacheable Cycle

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. The Intel486 DX samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle, before early RDY# to load the line just retrieved from the memory into the cache. KEN# is sampled every clock, it must satisfy setup and hold time.

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.

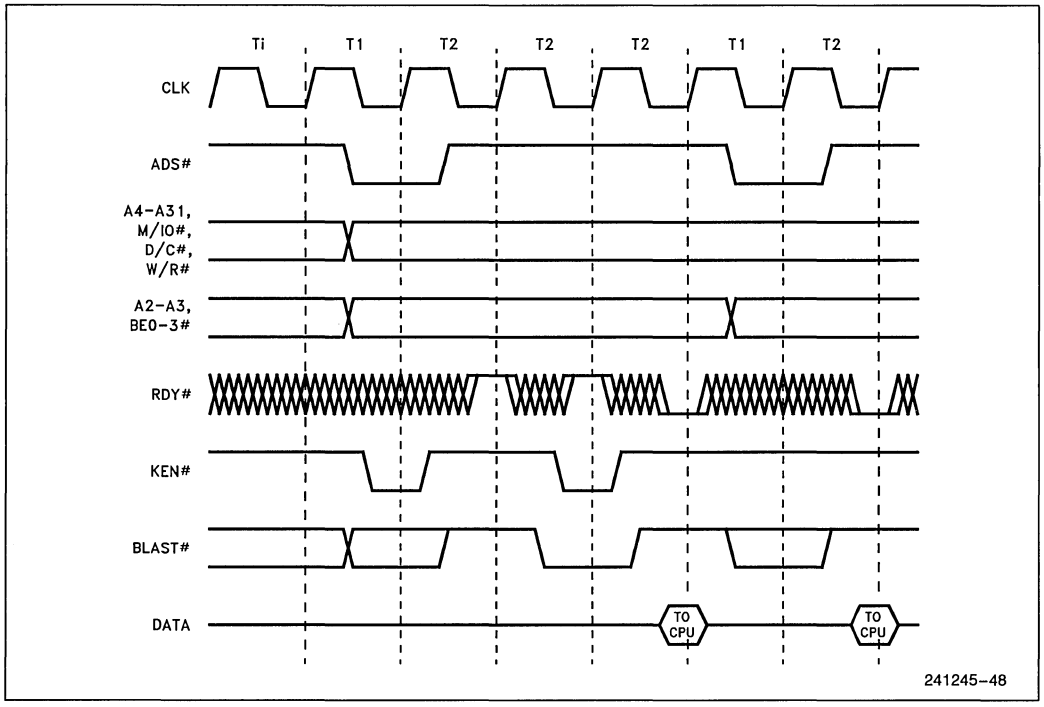


Figure 7.13. Effect of Changing KEN#

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 Intel486 DX 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.

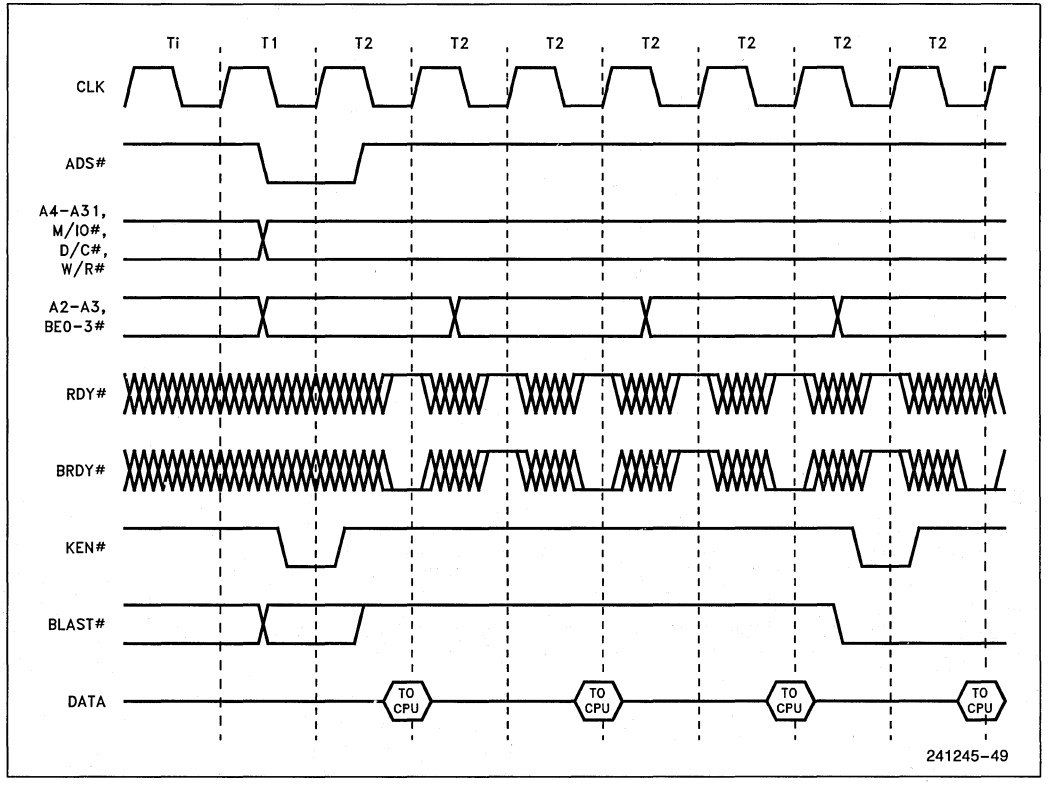


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Intel486 DX microprocessor is shown in Table 7.7. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

This burst order is optimized for a two-way interleaved memory architecture. This means that if the memory is built as 64-bit (versus 32-bit) words which are multiplexed into the 32-bit data bus, the Intel486 CPU will read all 64 bits before accessing the next location.

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.

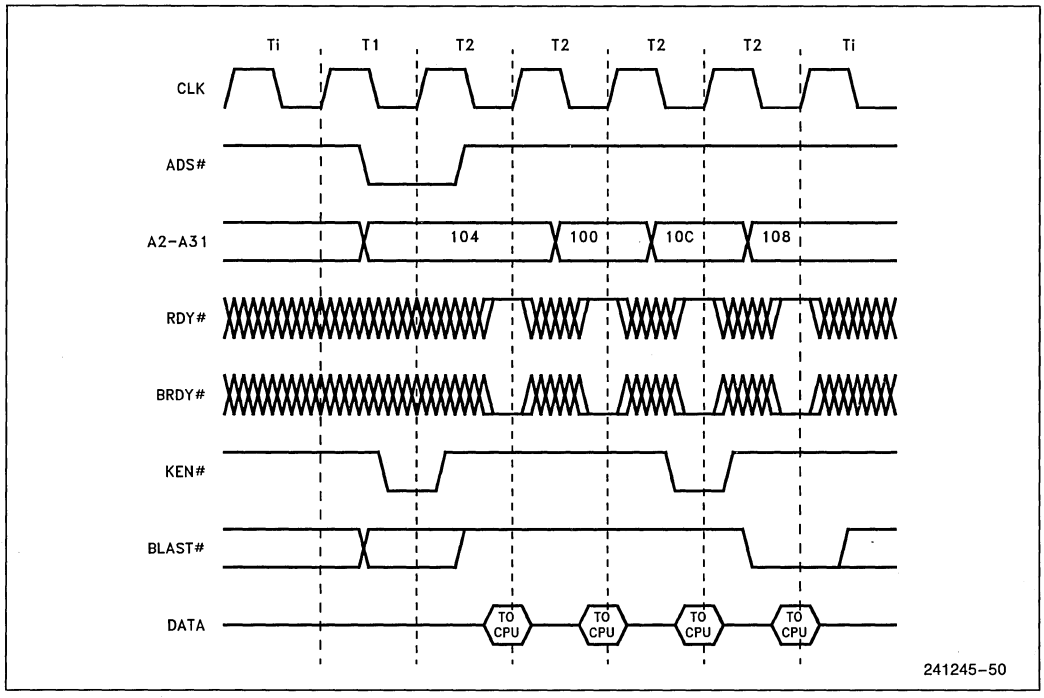


Figure 7.15. Burst Cycle Showing Order of Addresses

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 Intel486 DX 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 Intel486 DX microprocessor allows a burst cycle to be interrupted at any time. The Intel486 DX 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 Intel486 DX 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.

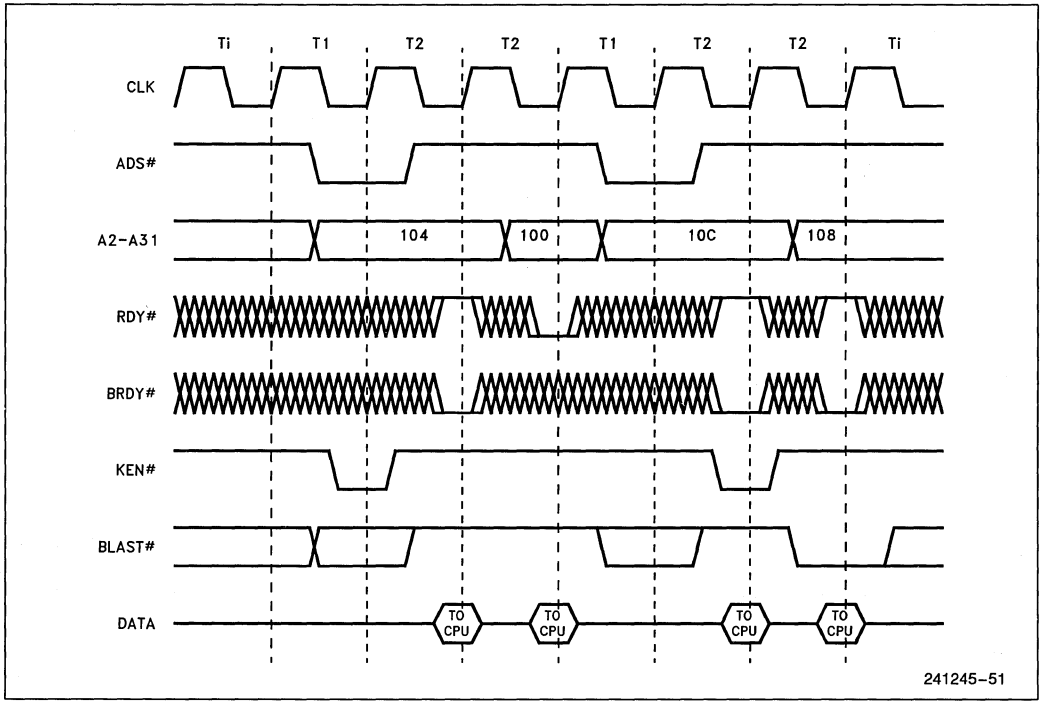


Figure 7.16. Interrupted Burst Cycle

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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor.

An example of the order in which the Intel486 DX microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The Intel486 DX 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 Intel486 DX microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the Intel486 DX 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#.

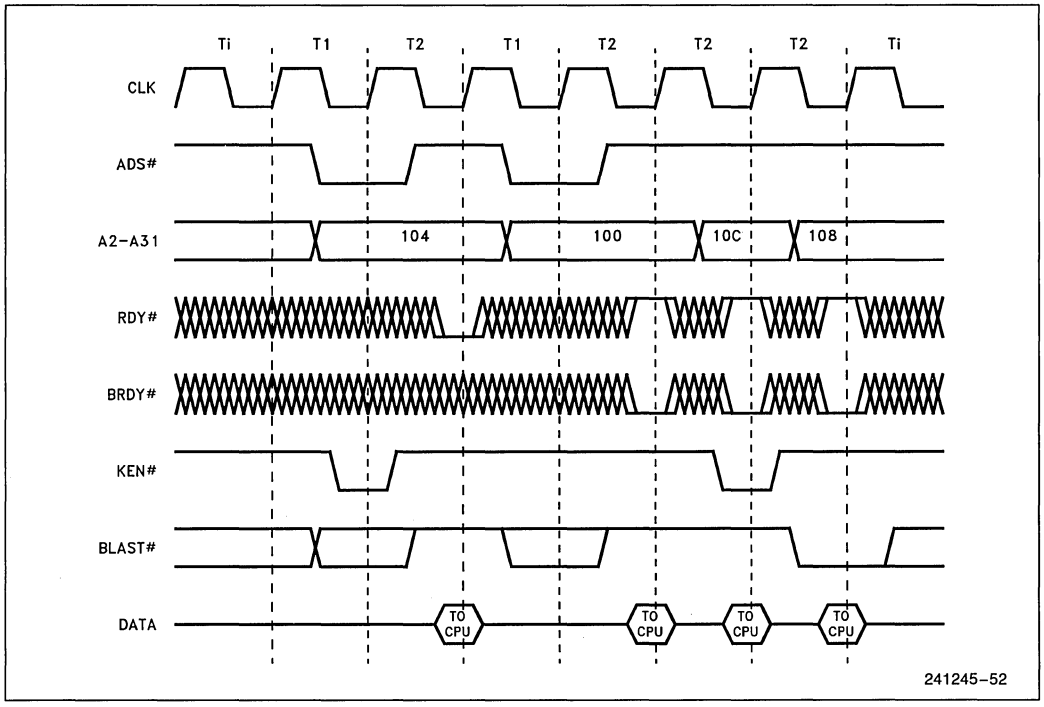


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor to run two extra cycles to complete a transfer. The Intel486 DX 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 Intel486 DX microprocessor issues two extra cycles to complete the transfer.

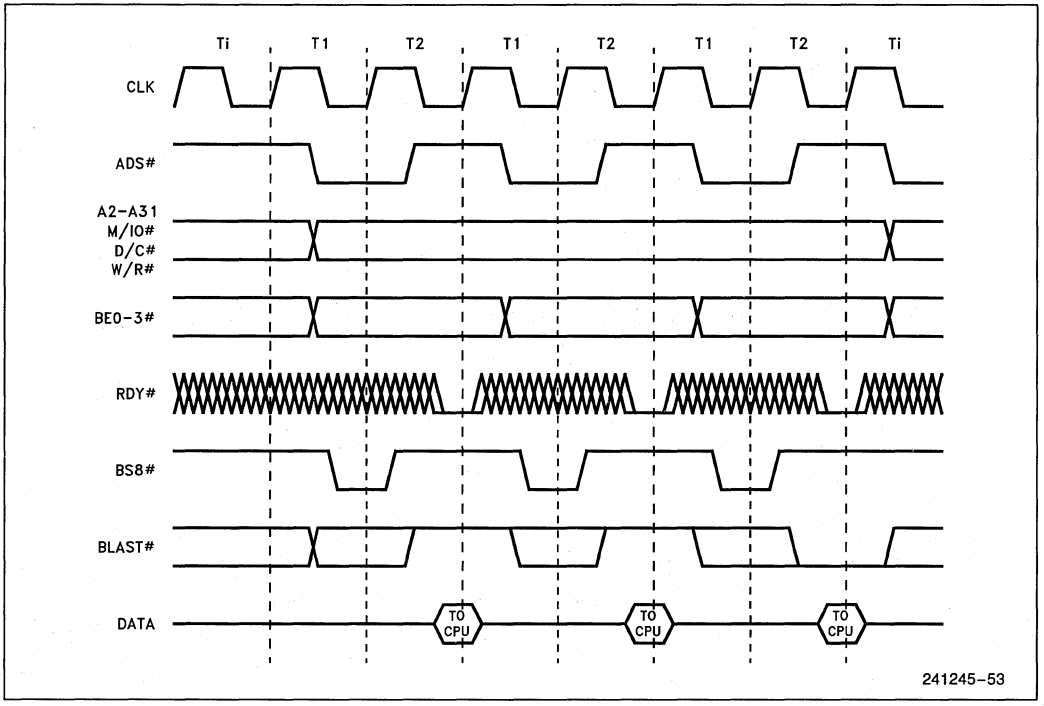


Figure 7.18. 8-Bit Bus Size Cycle

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 Intel486 DX 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 Intel486 DX 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.

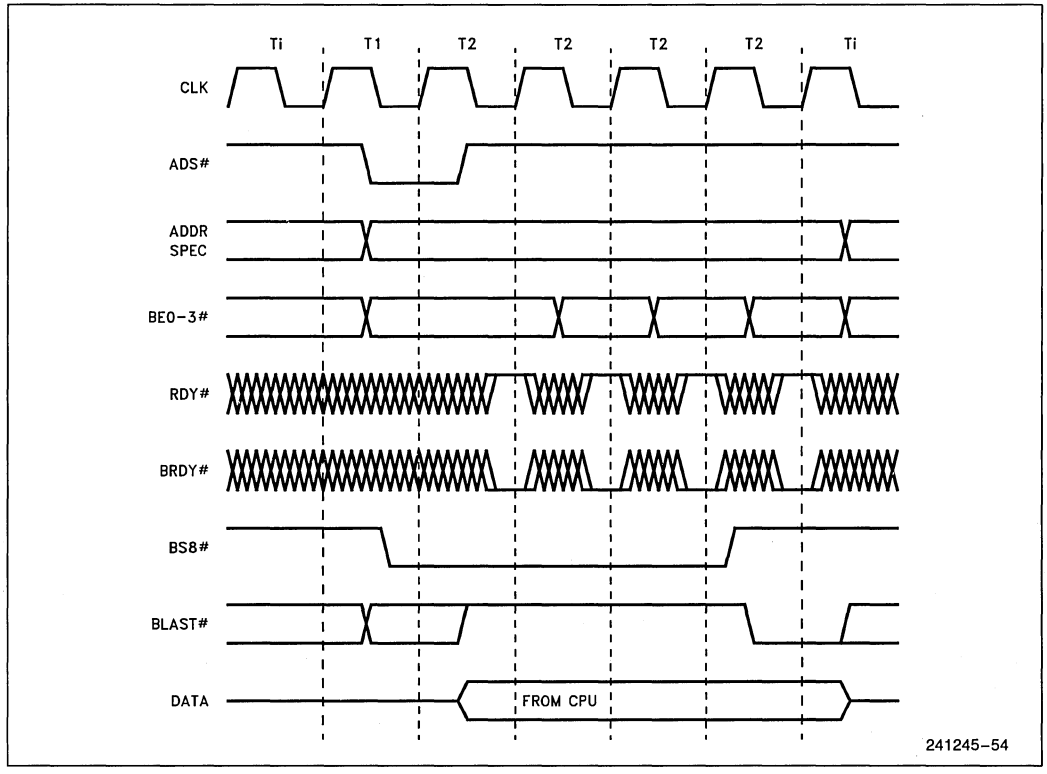


Figure 7.19. Burst Write as a Result of BS8# or BS16#

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. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

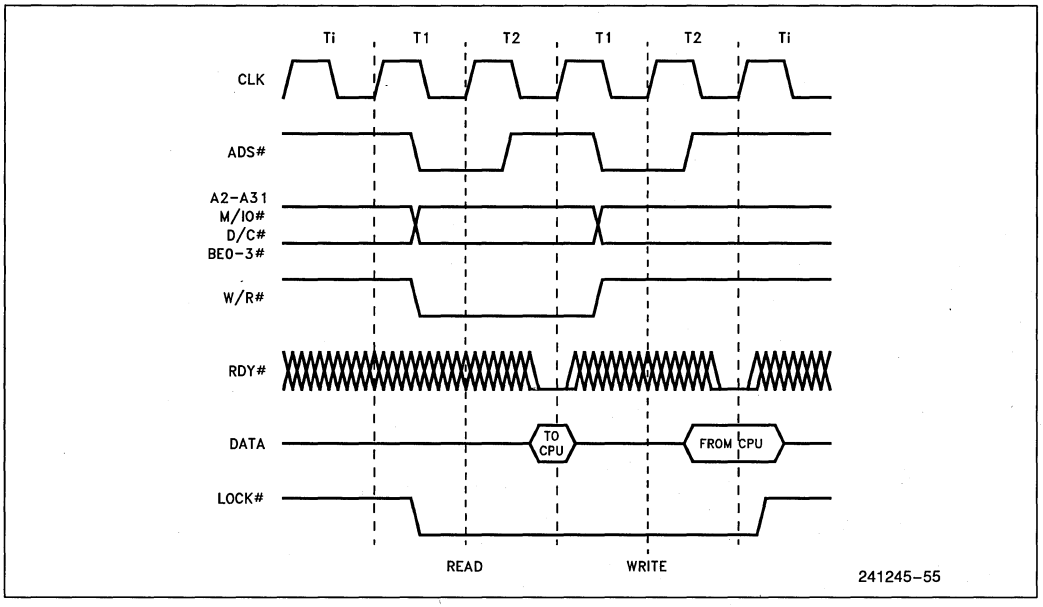


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be deasserted in response to BS8# or BS16#. 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.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable, non-bursted code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

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 Intel486 DX microprocessor's internal cache contents consistent with external memory. The Intel486 DX 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

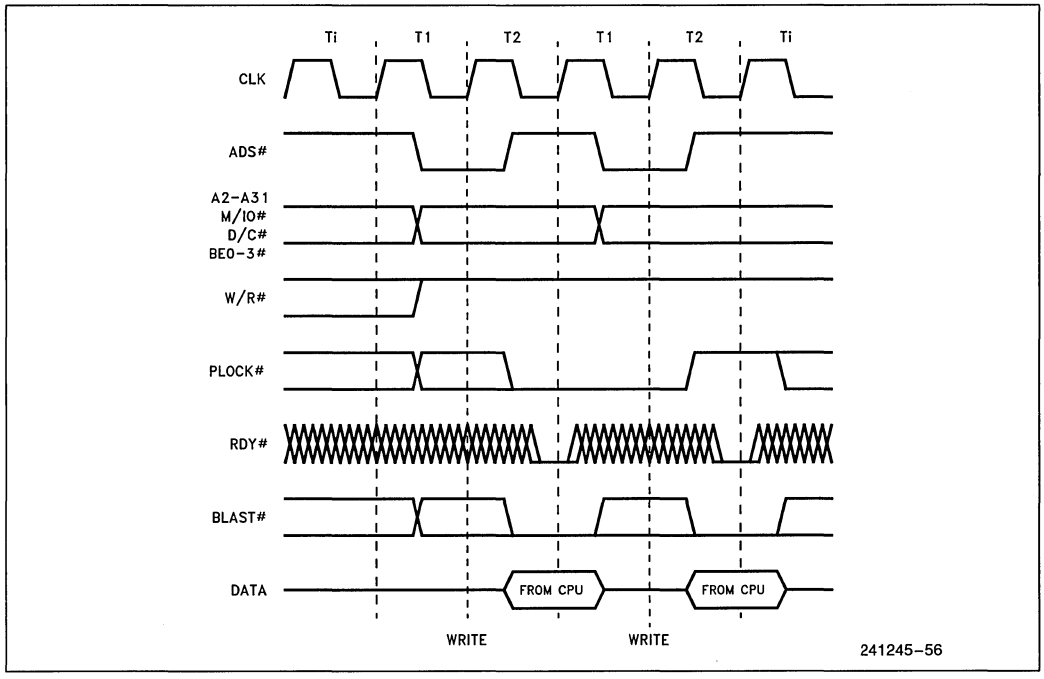


Figure 7.21. Pseudo Lock Timing

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 Intel486 DX microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the Intel486 DX microprocessor's address bus. EADS# and the invalidation address, Figure 7-22 shows the fastest possible invalidation cycle. The Intel486 DX CPU recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. 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 Intel486 DX microprocessor's address bus is input/output unlike the Intel386 microprocessor's bus, which is output only.

The Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor is driving the address bus, it is possible that the invalidation address will come from the Intel486 DX microprocessor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when HOLD or BOFF# is asserted.

Running an invalidate cycle prevents the Intel486 DX 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 Intel486 DX microprocessor.

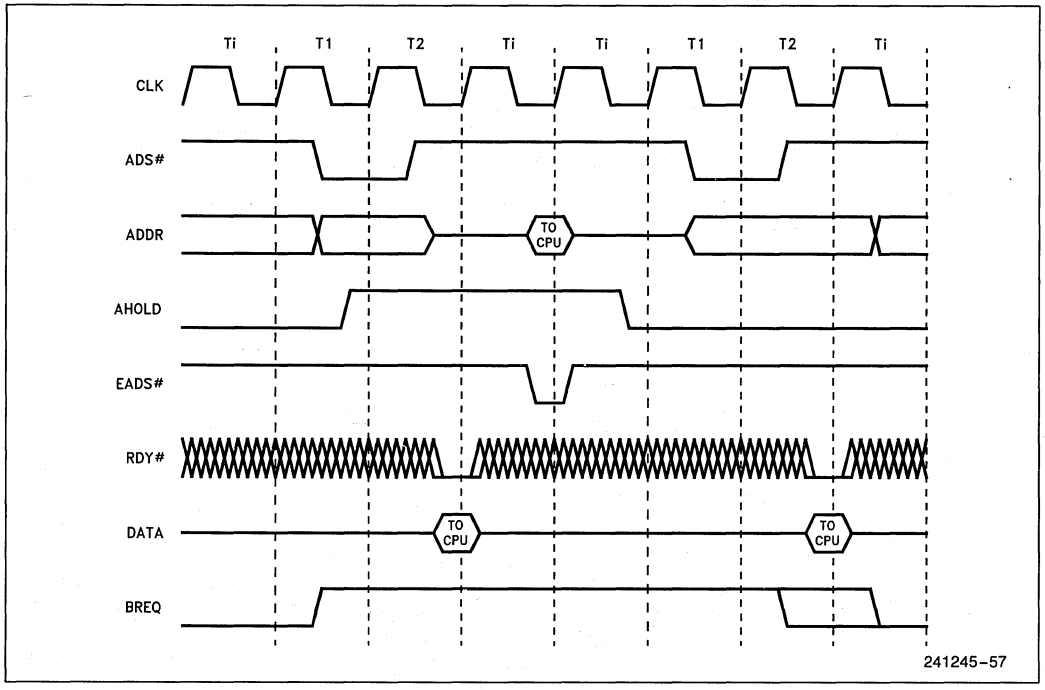


Figure 7.22. Fast Internal Cache Invalidation Cycle

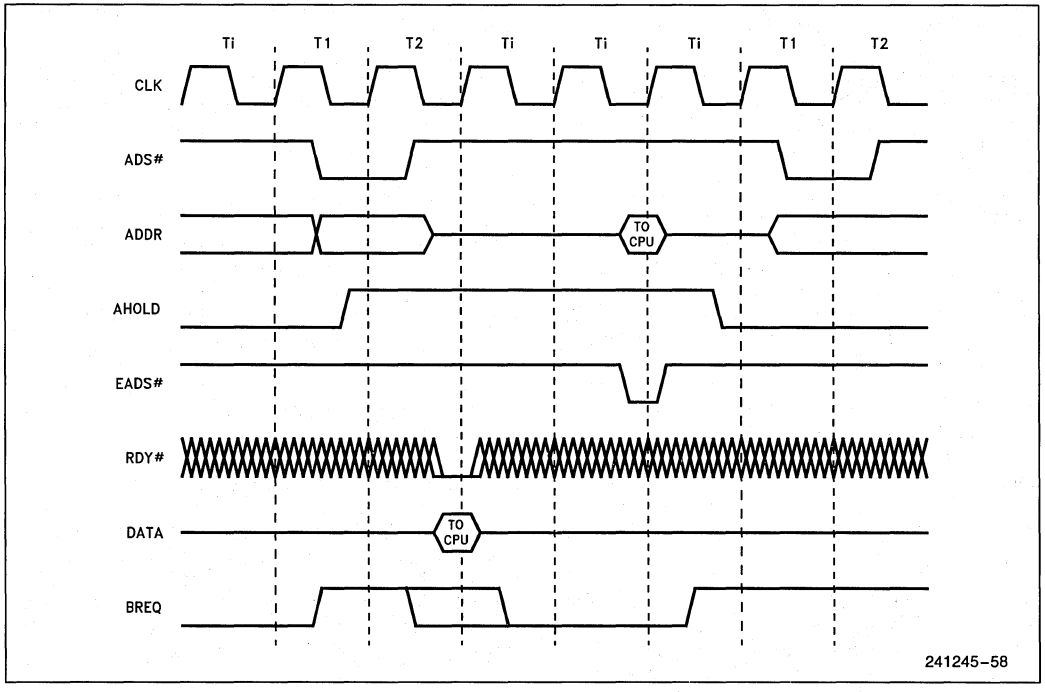


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor is retrieving data from the second level cache. The Intel486 DX 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 Intel486 DX microprocessor's cache.

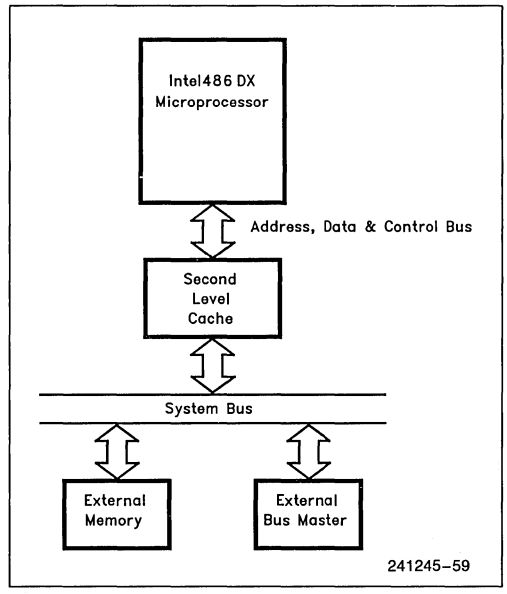
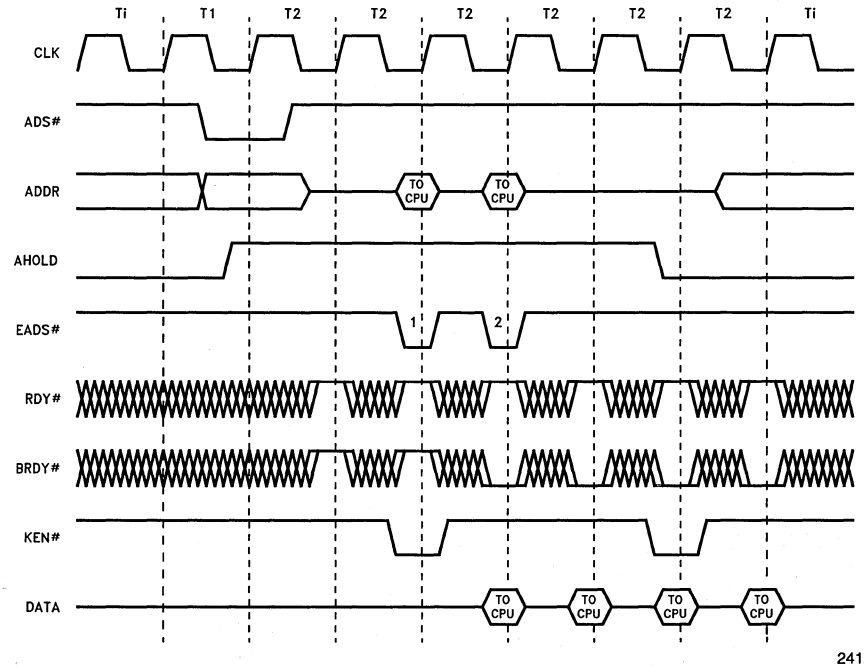


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Intel486 DX 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 Intel486 DX microprocessor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel386 microprocessor, the Intel486 DX microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted.

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 Intel486 DX 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.

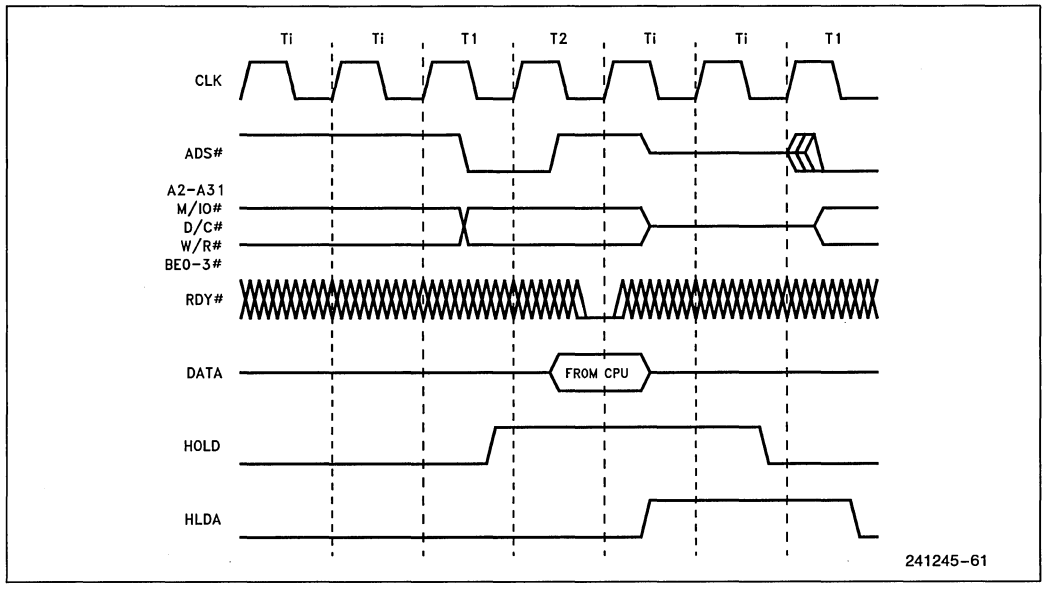


Figure 7.26. HOLD/HLDA Cycles

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 Intel486 DX microprocessor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

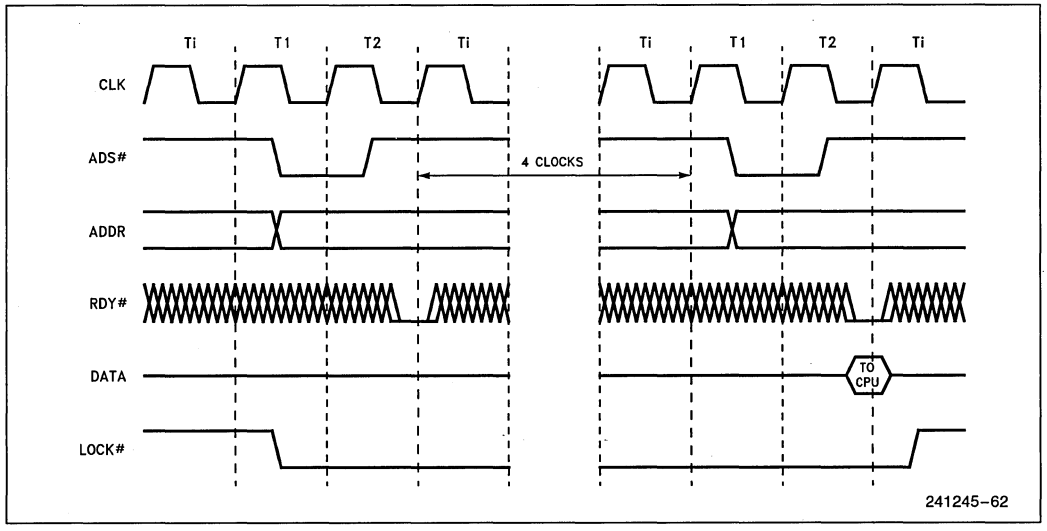


Figure 7.27. Interrupt Acknowledge Cycles

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 Intel486 DX 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 Intel486 DX 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. During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the Intel486 DX 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 Intel486 DX 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

7.2.11.1 Halt Indication Cycle

The Intel486 DX microprocessor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by RDY# or BRDY# asserted.

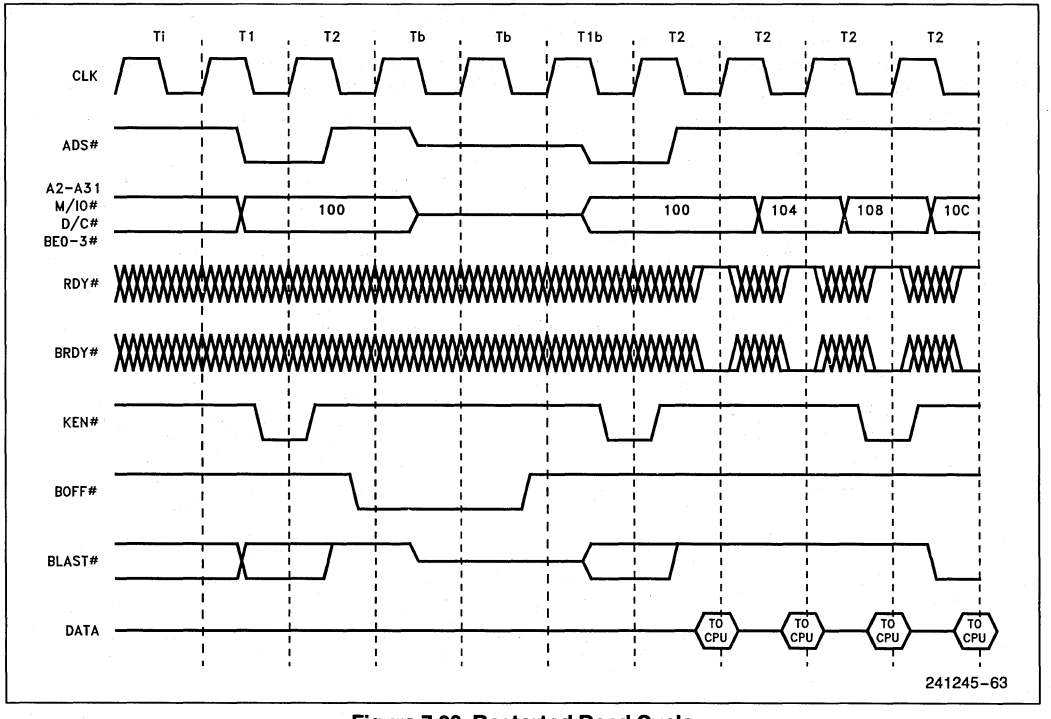


Figure 7.28. Restarted Read Cycle

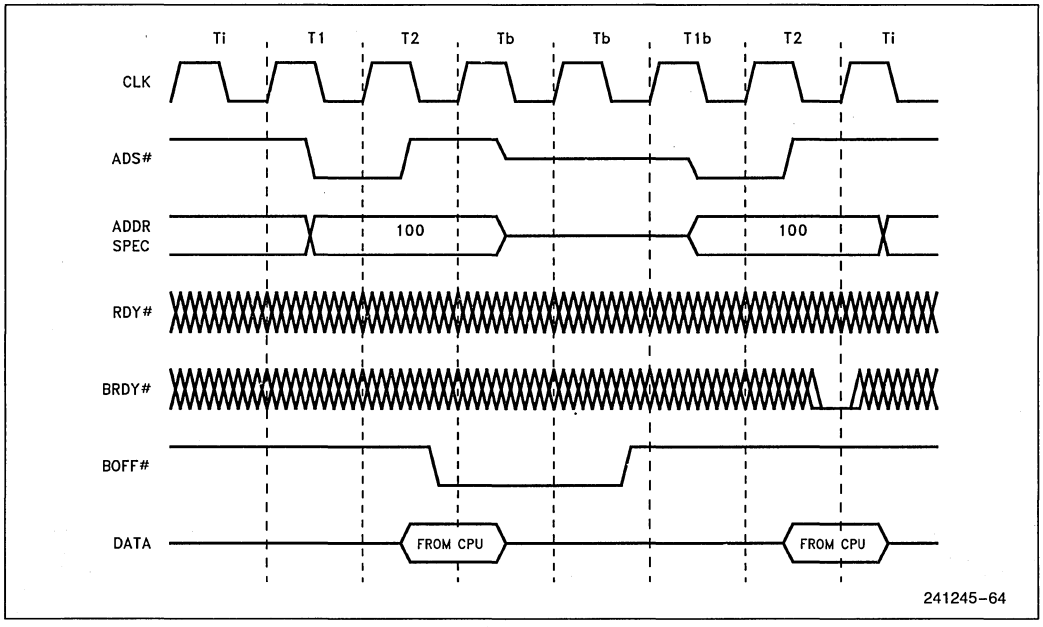


Figure 7.29. Restarted Write Cycle

A halted Intel486 DX microprocessor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

7.2.11.2 Shutdown Indication Cycle

The Intel486 DX microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the Intel486 DX microprocessor to complete its current bus request. In this situation the Intel486 DX 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 Intel486 DX microprocessor samples the BOFF# pin every clock. The Intel486 DX 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 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 Intel486 DX microprocessor bus is in its high impedance state. If backoff is requested after the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX 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. A description of the signals used in the diagram is given in Table 7.9.

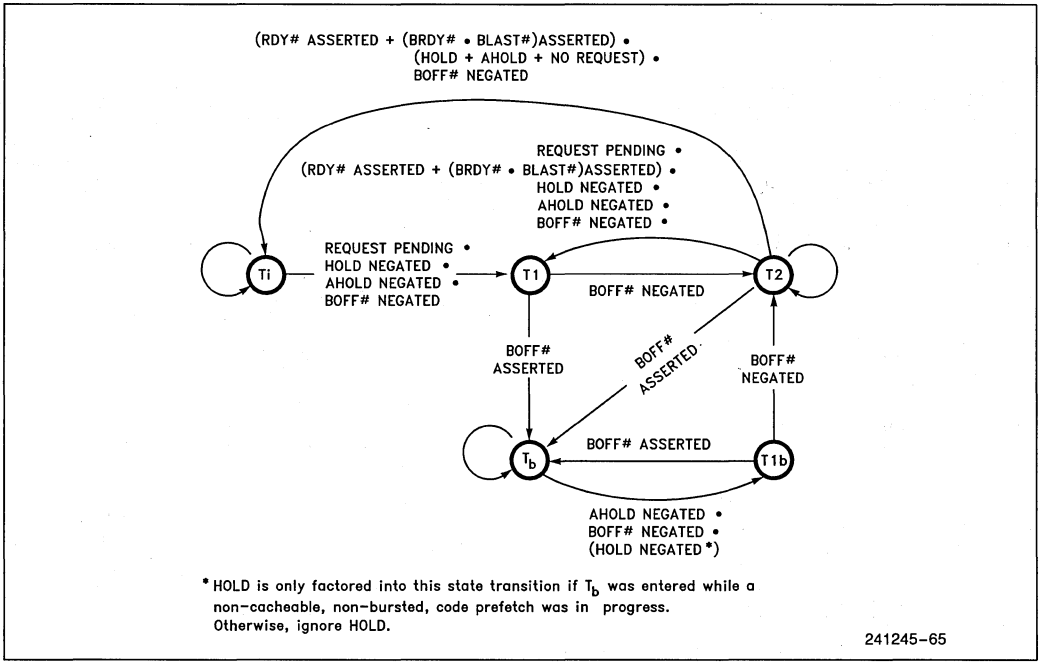


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

7.2.14 FLOATING POINT ERROR HANDLING

The Intel486 DX 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 Intel486 DX 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 Intel486 DX microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the Intel387 math coprocessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the Intel387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the Intel486 DX microprocessor.

When the NE bit in CR0 is cleared, and IGNNE# is asserted, the Intel486 DX microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the Intel486 DX 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 Intel486 DX 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 Intel486 DX 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 Intel486 DX2 CPU TESTABILITY

Testing the Intel486 DX2 microprocessor can be divided into three categories: Built-In Self Test (BIST), Boundary Scan, and external testing. BIST performs basic device testing on the Intel486 DX2 CPU, including the non-random logic, control ROM (CROM), translation lookaside buffer (TLB), and on-chip cache memory. Boundary Scan provides additional test hooks that conform to the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The Intel486 DX2 microprocessor also has a test mode in which all of its outputs are 3-stated. Additional testing can be performed by using the test registers within the Intel486 DX2 CPU.

8.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) pin HIGH for 2 CLKs before and 2 CLKs after RESET going from HIGH to LOW as shown in Figure 6.3. The BIST takes approximately 600 thousand clocks, or approximately 24 milliseconds with a 50 MHz Intel486 DX2 microprocessor. No bus cycles will be run by the Intel486 DX2 microprocessor until the BIST is concluded. Note that for the Intel486 DX2 microprocessor the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST is stored in the EAX register. The Intel486 DX2 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 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 Intel486 DX2 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.

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 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.

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 CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 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 8.2.

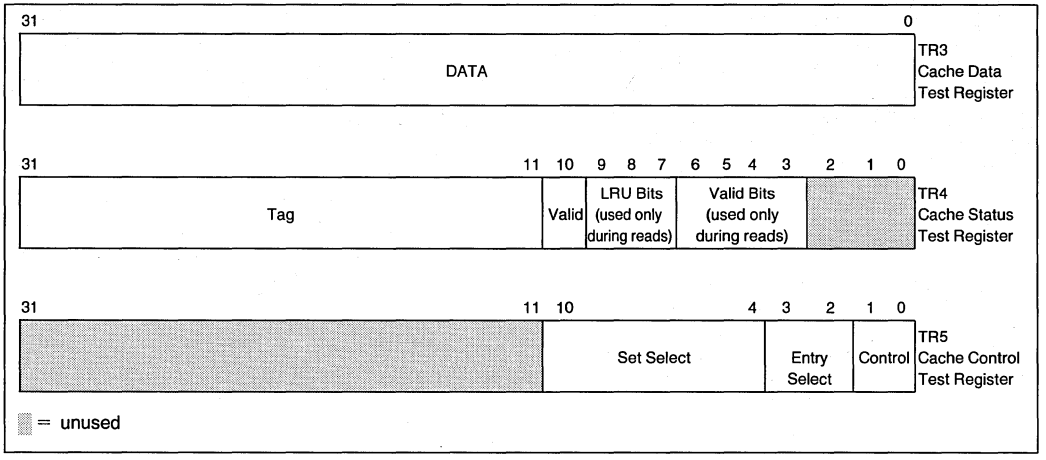


Figure 8.1. Cache Test Registers

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 8.2 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 CD bit is control register 0 is reset to 1). 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.

8.2.3 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 8.2.

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 Intel486 DX2 microprocessor whether the cache is enabled or not.

8.2.4 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 Intel486 DX2 microprocessor TLB testability hooks are similar to those in the Intel386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the Intel486 DX2 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 Intel486 DX2 microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

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.

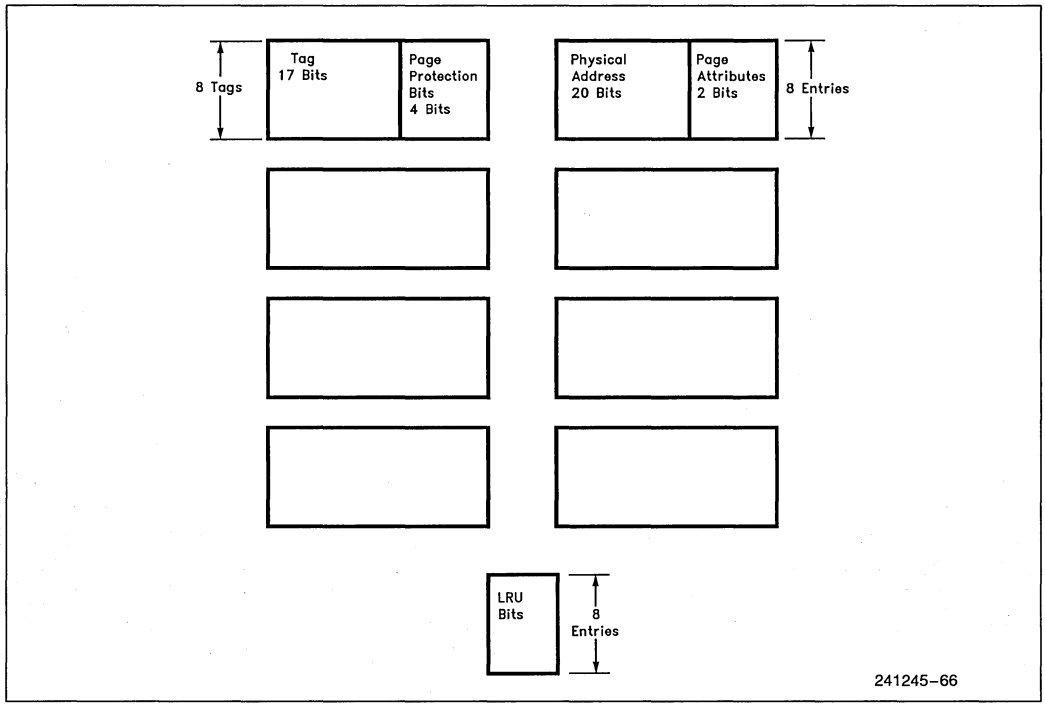


Figure 8.3. TLB Organization

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 8.4. 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.

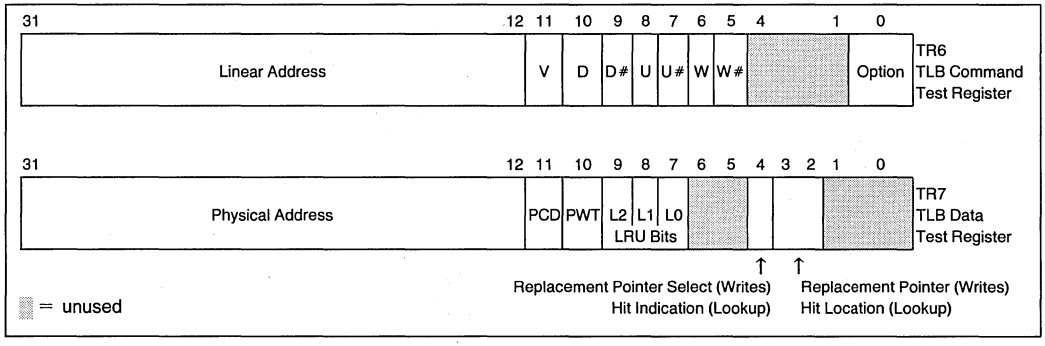


Figure 8.4. TLB Test Registers

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 Intel386 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 3-State Output Test Mode

The Intel486 DX2 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 Intel486 DX2 microprocessor is in the 3-state output test mode external testing can be used to test board connections.

The 3-state test mode is invoked by driving FLUSH# low for 2 clocks before and 2 clocks after RESET going low. The outputs are guaranteed to 3-state no later than 10 clocks after RESET goes low (see Figure 6.4). The Intel486 DX2 microprocessor remains in the 3-state test mode until the next RESET.

8.5 Intel486™ DX2 Microprocessor Boundary Scan (JTAG)

The Intel486 DX2 microprocessor provides testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std.1149.1). The test logic provided allows for testing to insure that components function correctly, that interconnections between various components are correct, and that various components interact correctly on the printed circuit board.

The boundary scan test logic consists of a boundary scan register and support logic that are accessed through a test access port (TAP). The TAP provides a simple serial interface that makes it possible to test all signal traces with only a few probes.

The TAP can be controlled via a bus master. The bus master can be either automatic test equipment or a component (PLD) that interfaces to the four-pin test bus.

8.5.1 BOUNDARY SCAN ARCHITECTURE

The boundary scan test logic contains the following elements:

— Test access port (TAP), consisting of input pins TMS, TCK, and TDI; and output pin TDO.

— TAP controller, which interprets the inputs on the test mode select (TMS) line and performs the corresponding operation. The operations performed by the TAP include controlling the instruction and data registers within the component.

— Instruction register (IR), which accepts instruction codes shifted into the test logic on the test data input (TDI) pin. The instruction codes are used to select the specific test operation to be performed or the test data register to be accessed.

— Test data registers: The Intel486 DX2 microprocessor contains three test data registers: Bypass register (BPR), Device Identification register (DID), and Boundary Scan register (BSR).

The instruction and test data registers are separate shift-register paths connected in parallel and have a common serial data input and a common serial data output connected to the TAP signals, TDI and TDO, respectively.

8.5.2 DATA REGISTERS

The Intel486 DX2 CPU contains the two required test data registers; bypass register and boundary scan register. In addition, they also have a device identification register.

Each test data register is serially connected to TDI and TDO, with TDI connected to the most significant bit and TDO connected to the least significant bit of the test data register. Data is shifted one stage (bit position within the register) on each rising edge of the test clock (TCK).

In addition the Intel486 DX2 CPU contains a runbist register to support the RUNBIST boundary scan instruction.

8.5.2.1 Bypass Register

The Bypass Register is a one-bit shift register that provides the minimal length path between TDI and TDO. This path can be selected when no test operation is being performed by the component to allow rapid movement of test data to and from other components on the board. While the bypass register is selected, data is transferred from TDI to TDO without inversion.

8.5.2.2 Boundary Scan Register

The Boundary Scan Register is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Intel486 DX2 CPU. Figure 8.1 shows the logical structure of the boundary scan register. While output cells determine the value of the signal driven on the corresponding pin, input cells only capture data; they do not affect the normal operation of the device. Data is transferred without inversion from TDI to TDO through the boundary scan register during scanning. The boundary scan register can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in Section 8.5.5.

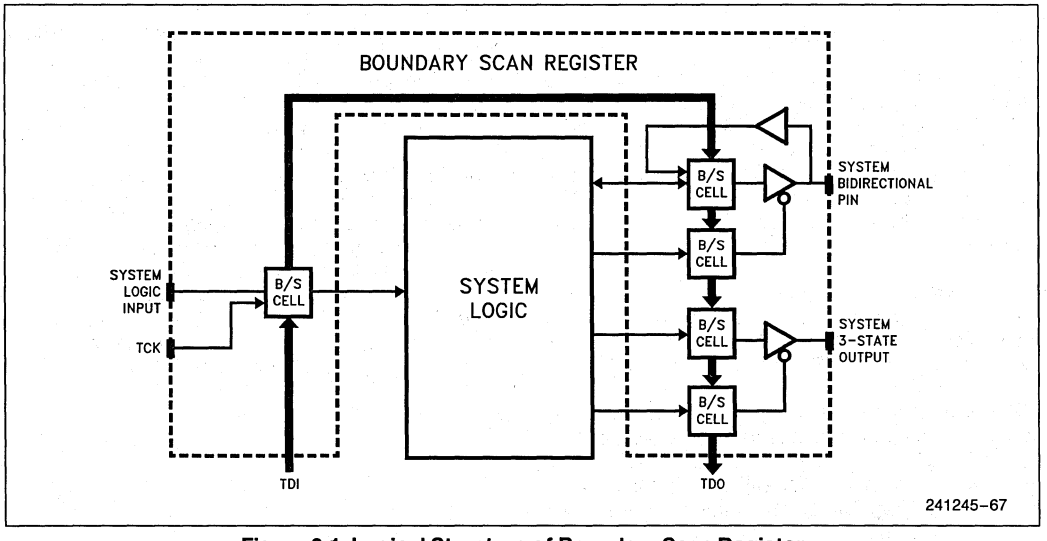


Figure 8.1. Logical Structure of Boundary Scan Register

8.5.2.3 Device Identification Register

The Device Identification Register contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 8.2. Table 8.1 lists the codes corresponding to the Intel486 DX2 CPU.

8.5.2.4 Runbist Register

The Runbist Register is a one bit register used to report the results of the Intel486 DX2 CPU BIST when it is initiated by the RUNBIST instruction. This register is loaded with a "1" prior to invoking the BIST and is loaded with "0" upon successful completion.

8.5.3 INSTRUCTION REGISTER

The Instruction Register (IR) allows instructions to be serially shifted into the device. The instruction selects the particular test to be performed, the test data register to be accessed, or both. The instruction register is four (4) bits wide. The most significant bit is connected to TDI and the least significant bit is connected to TDO. There are no parity bits associated with the Instruction register. Upon entering the Capture-IR TAP controller state, the Instruction register is loaded with the default instruction "0001", SAMPLE/PRELOAD. Instructions are shifted into the instruction register on the rising edge of TCK while the TAP controller is in the Shift-IR state.

8.5.3.1 Intel486 DX2 CPU Boundary Scan Instruction Set

The Intel486 DX2 CPU supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with two optional instructions (IDCODE and RUNBIST). Table 8.2 lists the Intel486 DX2 CPU boundary scan instruction codes. The instructions listed as PRIVATE cause TDO to become enabled in the Shift-DR state and cause "0" to be shifted out of TDO on the rising edge of TCK. Execution of the PRIVATE instructions will not cause hazardous operation of the Intel486 DX2 CPU.

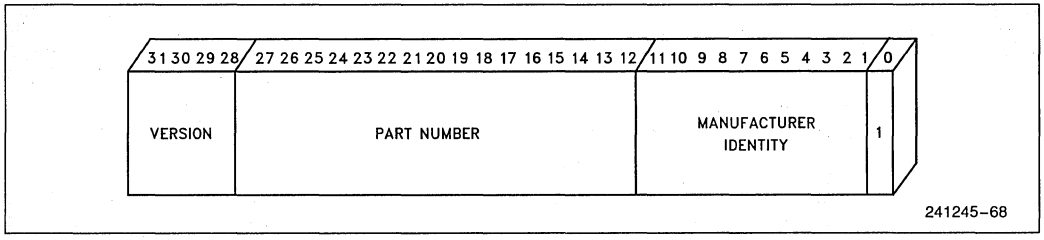


Figure 8.2. Format of Device Identification Register

Table 8.1

Component Code	Version Code	Part Number Code	Manufacturer Identity
Intel486 DX2 CPU (Ax)	00h	0432h	09h
Intel486 DX2 CPU (Bx)	00h	0433h	09h

Table 8.2

Instruction Code	Instruction Name
0000	EXTEST
0001	SAMPLE
0010	IDCODE
0011	PRIVATE
0100	PRIVATE
0101	PRIVATE
0110	PRIVATE
0111	PRIVATE
1000	RUNBIST
1001	PRIVATE
1010	PRIVATE
1011	PRIVATE
1100	PRIVATE
1101	PRIVATE
1110	PRIVATE
1111	BYPASS

EXTEST The instruction code is "0000". The EXTEST instruction allows testing of circuitry external to the component package, typically board interconnects. It does so by driving the values loaded into the Intel486 DX2 CPU's boundary scan register out on the output pins corresponding to each boundary scan cell and capturing the values on Intel486 DX2 CPU input pins to be loaded into their corresponding boundary scan register locations. I/O pins are selected as input or output, depending on the value loaded into their control setting locations in the boundary scan register. Values shifted into input latches in the boundary scan register are never used by the internal logic of the Intel486 DX2 CPU.

NOTE:

After using the EXTEST instruction, the Intel486 DX2 CPU must be reset before normal (non-boundary scan) use.

SAMPLE/PRELOAD The instruction code is "0001". The SAMPLE/PRELOAD has two functions that it performs. When the TAP controller is in the Capture-DR state, the SAMPLE/PRELOAD instruction allows a "snap-shot" of the normal operation of the component without interfering with that normal operation. The instruction causes boundary scan register cells associated with outputs to sample the value being driven by the Intel486 DX2 CPU. It causes the cells associated with inputs to sample the value being driven into the Intel486 DX2 CPU. On both outputs and inputs the sampling occurs on the rising edge of TCK. When the TAP controller is in the Update-DR state, the SAMPLE/PRELOAD instruction preloads data to the device pins to be driven to the board by executing the EXTEST instruction. Data is preloaded to the pins from the boundary scan register on the falling edge of TCK.

IDCODE The instruction code is "0010". The IDCODE instruction selects the device identification register to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the device identification register is not altered by data being shifted in on TDI.

BYPASS The instruction code is "1111". The BYPASS instruction selects the bypass register to be connected to TDI or TDO, effectively bypassing the test logic on the Intel486 DX2 microprocessor by reducing the shift length of the device to one bit. Note than an open circuit fault in the board level test data path will cause the bypass register to be selected following an instruction scan cycle due to the pull-up resistor on the TDI input. This has been done to prevent any unwanted interference with the proper operation of the system logic.

RUNBIST The instruction code is "1000". The RUNBIST instruction selects the one (1) bit runbist register, loads a value of "1" into the runbist register, and connects it to TDO. It also initiates the built-in self test (BIST) feature of the Intel486 DX2 CPU, which is able to detect approximately 60% of the stuck-at faults on the Intel486 DX2 CPU. The Intel486 DX2 CPU AC/DC Specifications for V_{CC} and CLK must be met and reset must have been asserted at least once prior to executing the RUNBIST boundary scan instruction. After loading the RUNBIST instruction code in the instruction register, the TAP controller must be placed in the Run-Test/Idle state. BIST begins on the first rising edge of TCK after entering the Run-Test/Idle state. The TAP

controller must remain in the Run-Test/Idle state until BIST is completed. It requires 1.2 million clock (CLK) cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million clock (CLK) cycles, the value in the runbist register should be shifted out on TDO during the Shift-DR state. A value of "0" being shifted out on TDO indicates BIST successfully completed. A value of "1" indicates a failure occurred. After executing the RUNBIST instruction, the Intel486 DX2 CPU must be reset prior to normal operation.

8.5.4 TEST ACCESS PORT (TAP) CONTROLLER

The TAP controller is a synchronous, finite state machine. It controls the sequence of operations of the test logic. The TAP controller changes state only in response to the following events:

1. a rising edge of TCK
2. power-up.

The value of the test mode state (TMS) input signal at a rising edge of TCK controls the sequence of the state changes. The state diagram for the TAP controller is shown in Figure 8.3. Test designers must consider the operation of the state machine in order to design the correct sequence of values to drive on TMS.

8.5.4.1 Test-Logic-Reset State

In this state, the test logic is disabled so that normal operation of the device can continue unhindered. This is achieved by initializing the instruction register such that the IDCODE instruction is loaded. No matter what the original state of the controller, the controller enters Test-Logic-Reset state when the TMS input is held high (1) for at least five rising edges of TCK. The controller remains in this state while TMS is high. The TAP controller is also forced to enter this state at power-up.

8.5.4.2 Run-Test/Idle State

A controller state between scan operations. Once in this state, the controller remains in this state as long

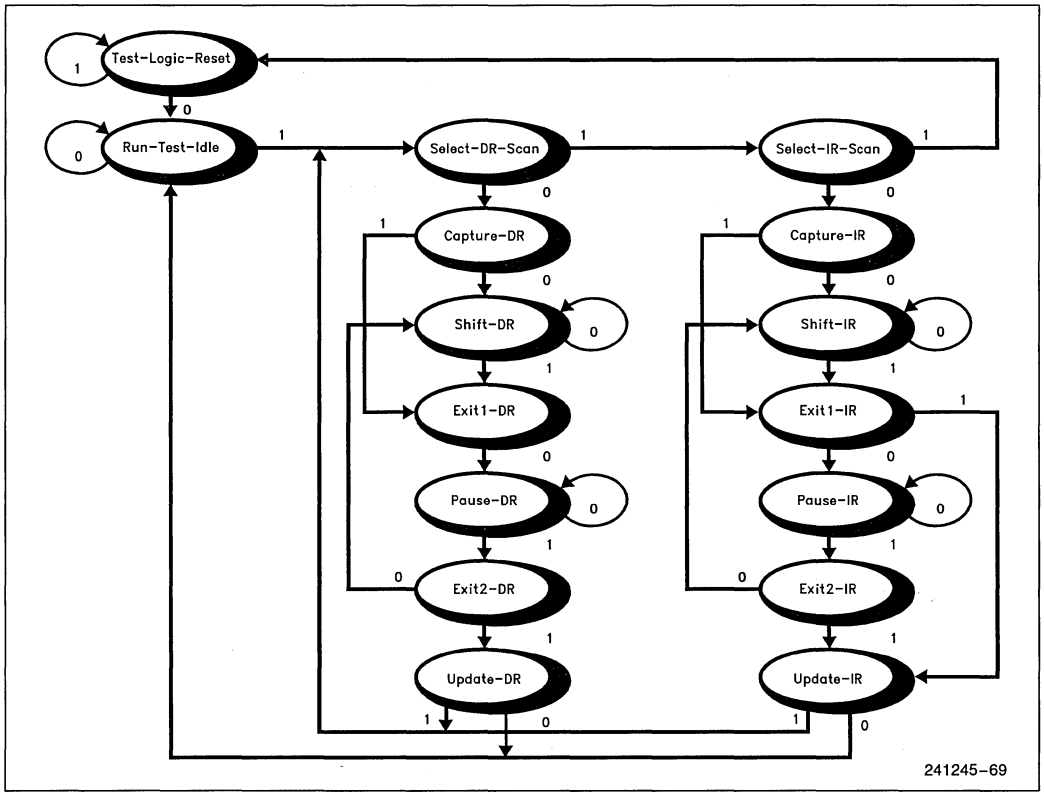


Figure 8.3. TAP Controller State Diagram

as TMS is held low. In devices supporting the RUNBIST instruction, the BIST is performed during this state and the result is reported in the runbist register. For instruction not causing functions to execute during this state, no activity occurs in the test logic. The instruction register and all test data registers retain their previous state. When TMS is high and a rising edge is applied to TCK, the controller moves to the Select-DR state.

8.5.4.3 Select-DR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-DR state, and a scan sequence for the selected test data register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Select-IR-Scan state.

The instruction does not change in this state.

8.5.4.4 Capture-DR State

In this state, the boundary scan register captures input pin data if the current instruction is EXTEST or SAMPLE/PRELOAD. The other test data registers, which do not have parallel input, are not changed.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or the Shift-DR state if TMS is low.

8.5.4.5 Shift-DR State

In this controller state, the test data register connected between TDI and TDO as a result of the current instruction, shifts data one stage toward its serial output on each rising edge of TCK.

The instruction does not change in this state.

When the TAP controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-DR state if TMS is high or remains in the Shift-DR state if TMS is low.

8.5.4.6 Exit1-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.7 Pause-Dr State

The pause state allows the test controller to temporarily halt the shifting of data through the test data register in the serial path between TDI and TDO. An example of using this state could be to allow a tester to reload its pin memory from disk during application of a long test sequence.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-DR state.

8.5.4.8 Exit2-DR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-DR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-DR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.9 Update-DR State

The boundary scan register is provided with a latched parallel output to prevent changes at the parallel output while data is shifted in response to the EXTEST and SAMPLE/PRELOAD instructions. When the TAP controller is in this state and the boundary scan register is selected, data is latched onto the parallel output of this register from the shift-register path on the falling edge of TCK. The data held at the latched parallel output does not change other than in this state.

All shift-register stages in test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.10 Select-IR-Scan State

This is a temporary controller state. The test data register selected by the current instruction retains its previous state. If TMS is held low and a rising edge is applied to TCK when in this state, the controller moves into the Capture-IR state, and a scan sequence for the instruction register is initiated. If TMS is held high and a rising edge is applied to TCK, the controller moves to the Test-Logic-Reset state.

The instruction does not change in this state.

8.5.4.11 Capture-IR State

In this controller state the shift register contained in the instruction register loads the fixed value "0001" on the rising edge of TCK.

The test data register selected by the current instruction retains it previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or the Shift-IR state if TMS is held low.

8.5.4.12 Shift-IR State

In this state the shift register contained in the instruction register is connected between TDI and TDO and shifts data one stage towards its serial output on each rising edge of TCK.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

When the controller is in this state and a rising edge is applied to TCK, the controller enters the Exit1-IR state if TMS is held high, or remains in the Shift-IR state if TMS is held low.

8.5.4.13 Exit1-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Pause-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.14 Pause-IR State

The pause state allows the test controller to temporarily halt the shifting of data through the instruction register.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

The controller remains in this state as long as TMS is low. When TMS goes high and a rising edge is applied to TCK, the controller moves to the Exit2-IR state.

8.5.4.15 Exit2-IR State

This is a temporary state. While in this state, if TMS is held high, a rising edge applied to TCK causes the controller to enter the Update-IR state, which terminates the scanning process. If TMS is held low and a rising edge is applied to TCK, the controller enters the Shift-IR state.

The test data register selected by the current instruction retains its previous value during this state. The instruction does not change in this state.

8.5.4.16 Update-IR State

The instruction shifted into the instruction register is latched onto the parallel output from the shift-register path on the falling edge of TCK. Once the new instruction has been latched, it becomes the current instruction.

Test data registers selected by the current instruction retain the previous value.

8.5.5 BOUNDARY SCAN REGISTER CELL

The boundary scan register contains a cell for each pin, as well as cells for control of I/O and 3-state pins.

The following is the bit order of the Intel486 DX2 CPU boundary scan register: (from left to right and top to bottom).

TDI → WRCTL ABUSCTL BUSCTL MISCCTL ADS# BLAST# PLOCK# LOCK# PCHK# BRDY# BOFF# BS16# BS8# RDY# KEN# HOLD AHOLD CLK HLDA WR# BREQ BE0# BE1# BE2# BE3# MIO# DC# PWT PCD EADS# A20M# RESET FLUSH# INTR NMI UP# FERR# IGNNE# D31 D30 D29 D28 D27 D26 D25 D24 DP3 D23 D22 D21 D20 D19 D18 D17 D16 DP2 D15 D14 D13 D12 D11 D10 D9 D8 DP1 D7 D6 D5 D4 D3 D2 D1 D0 DP0 A31 A30 A29 A28 A27 A26 A25 A24 A23 A22 A21 A20 A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 RESERVED A5 A4 A3 A2 → TDO

"RESERVED" corresponds to no connect "NC" signals on the Intel486 DX2 CPU.

All the *CTL cells are control cells that are used to select the direction of bidirectional pins or 3-state output pins. If "1" is loaded into the control cell (*CTL), the associated pin(s) are 3-stated or selected as input. The following lists the control cells and their corresponding pins.

1. WRCTL controls the D31–0 and DP3–0 pins.
2. ABUSCTL controls the A31–A2 pins.
3. BUSCTL controls the ADS#, BLAST#, PLOCK#, LOCK#, WR#, BE0#, BE1#, BE2#, BE3#, MIO#, DC#, PWT, and PCD pins.
4. MISCCTL controls the PCHK#, HLDA, BREQ, and FERR# pins.

8.5.6 TAP CONTROLLER INITIALIZATION

The TAP controller is automatically initialized when a device is powered up. In addition, the TAP controller can be initialized by applying a high signal level on the TMS input for five TCK periods.

8.5.7 BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL)

```
entity i486DX2 is
   generic(PHYSICAL_PIN_MAP : string := "PGA_18x18");

   port (A20M_     : in      bit;
         A         : inout   bit_vector(2 to 31); -- Address bus (words)
         ADS_      : out     bit;
         AHOLD     : in      bit;
         BE_       : out     bit_vector(0 to 3);
         BLAST_    : out     bit;
         BOFF_     : in      bit;
         BRDY_     : in      bit;
         BREQ      : out     bit;
         BS8_      : in      bit;
         BS16_     : in      bit;
         CLK       : in      bit;
         D         : inout   bit_vector(0 to 31);  -- Data bus
         DC_       : out     bit;
         DP        : inout   bit_vector(0 to 3);
         EADS_     : in      bit;
         FERR_     : out     bit;
         FLUSH_    : in      bit;
         HLDA      : out     bit;
         HOLD      : in      bit;
         IGNNE_    : in      bit;
         INTR      : in      bit;
         KEN_      : in      bit;
         LOCK_     : out     bit;
         MIO_      : out     bit;
         NC        : linkage bit_vector(1 to 12); -- No Connects
         NC1       : in      bit;
         NMI       : in      bit;
         PCD       : out     bit;
         PCHK_     : out     bit;
         PLOCK_    : out     bit;
         PWT       : out     bit;
         RDY_      : in      bit;
         RESET     : in      bit;
         RES_A     : in      bit;
         RES_B     : out     bit;
         TCK       : in      bit;                -- Scan Port inputs
         TMS       : in      bit;                -- Scan Port inputs
         TDI       : in      bit;                -- Scan Port inputs
         TDO       : out     bit;                -- Scan Port output
         UP_       : in      bit;
         VCC       : linkage bit_vector(1 to 21);   -- VCC
         GND       : linkage bit_vector(1 to 27);   -- VSS or GND
         VCCANA    : linkage bit;                   -- VCCANA
         VSSANA    : linkage bit;                   -- VSSANA
         WR_       : out   bit);

   use STD_1149_1_1990.all;

   attribute PIN_MAP of i486DX2: entity is PHYSICAL_PIN_MAP;

   constant PGA_18x18 : PIN_MAP_STRING :=      -- Define Pin Out of PGA
        "A20M_     : D15,  "   &
        "A         :(Q14, R15, S16, Q12, S15, Q13, R13, Q11, S13, R12," &
        "            S7,  Q10, S5,  R7,  Q9,  Q3,  R5,  Q4,  Q8,  Q5,"  &
```

241245-87

Intel486™ DX2 CPU BSDL Model for Boundary Scan

```
      "            Q7,  S3,  Q6,  R2,  S2,  S1,  R1,  P2,  P3,  Q1)," &
      "ADS_      : S17, "  &
      "AHOLD     : A17, "  &
      "BE_       : (K15, J16, J15, F17), "  &
      "BLAST_    : R16, "  &
      "BOFF_     : D17, "  &
      "BRDY_     : H15, "  &
      "BREQ      : Q15, "  &
      "BS8_      : D16, "  &
      "BS16_     : C17, "  &
      "CLK       : C3,  "  &
      "D         : (P1, N2, N1, H2, M3, J2, L2, L3, F2, D1, E3, "  &
      "             C1, G3, D2, K3, F3, J3, D3, C2, B1, A1, B2, "  &
      "             A2, A4, A6, B6, C7, C6, C8, A8, C9, B8)," &
      "DC_       : M15, "  &
      "DP        : (N3,  F1,  H3,  A5), "  &
      "EADS_     : B17, "  &
      "FERR_     : C14, "  &
      "FLUSH_    : C15, "  &
      "HLDA      : P15, "  &
      "HOLD      : E15, "  &
      "IGNNE_    : A15, "  &
      "INTR      : A16, "  &
      "KEN_      : F15, "  &
      "LOCK_     : N15, "  &
      "MIO_      : N16, "  &
      "NC        : (R17, G15, C10, C13, B10, B12, B13," &
      "             A10, A12, A13)," &
      "NC1       : S4,  "  &
      "NMI       : B15, "  &
      "PCD       : J17, "  &
      "PCHK_     : Q17, "  &
      "PLOCK_    : Q16, "  &
      "PWT       : L15, "  &
      "RDY_      : F16, "  &
      "RESET     : C16, "  &
      "RES_A     : B11, "  &
      "RES_B     : C12, "  &
      "TCK       : A3,  "  &
      "TDI       : A14, "  &
      "TDO       : B16, "  &
      "TMS       : B14, "  &
      "UP_       : C11, "  &
      "VCC       : (R8, R9, R10, R11, R14, P16, M2, M16, L16, K2,  "  &
      "             K16, J1, H16, G2,  G16, E2,  E16, C5, B7,  B9), "  &
      "GND       : (S6, S8, S9, S10, S11, S12, S14, R4, Q2, P17, M1, "  &
      "            M17, L1, L17, K1, K17, H1, H17, G1, G17, E1, E17, "  &
      "               B3,  B5,  A7,  A9,  A11), "  &
      "VCCANA    : C4, "  &
      "VSSANA    : B4, "  &
      "WR_       : N17 ";

attribute Tap_Scan_In    of  TDI   : signal is true;
attribute Tap_Scan_Mode  of  TMS   : signal is true;
attribute Tap_Scan_Out   of  TDO   : signal is true;
attribute Tap_Scan_Clock of  TCK   : signal is (25.0e6, BOTH);

attribute Instruction_Length of i486: entity is 4;

attribute Instruction_Opcode of i486: entity is
   "BYPASS  (1111),"    &
   "EXTEST  (0000),"    &
   "SAMPLE  (0001),"    &
   "IDCODE  (0010),"    &
```


241245-88

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

```
      "RUNBIST (1000),"    &
      "PRIVATE
(0011,0100,0101,0110,0111,1001,1010,1011,1100,1101,1110)";

   attribute Instruction_Capture of i486: entity is "0001";
   -- there is no Instruction_Disable attribute for i486

   attribute Instruction_Private of i486: entity is "private";

   attribute Instruction_Usage of i486: entity is
      "RUNBIST (registers BIST; "  &
      "result 0;"                  &
      "clock CLK in Run_Test_Idle;"&
      "length 1200000)";

   attribute Idcode_Register of i486: entity is
      "0000"                    &  --version, A-step
      "0000010000010000"        &  --part number
      "00000001001"             &  --manufacturers identity
      "1";                         --required by the standard

   attribute Register_Access of i486: entity is
      "BIST[1] (RUNBIST)";

--{*****************************************************************}
--{  The first cell is closest to TDO                                }
--{                                                                  }
--{  Note on cell type BC_3:                                         }
--{  In our case, bscan output is not routed to the output MUX, only }
--{  to next scan cell. Thus, i486(tm)DX does not support INTEST.    }
--{  The correct cell type is BC_4.                                  }
--{  This is a work-around since HP BSDL parser and package don't    }
--{  BC_4 cell type to support RUNBIST.                              }
--{*****************************************************************}

   attribute Boundary_Cells of i486: entity is "BC_1, BC_2, BC_3, BC_6";
   attribute Boundary_Length of i486: entity is 107;
   attribute Boundary_Register of i486: entity is
      "0   (BC_2,  A(2),    output3, X,  102,  1,  Z)," &
      "1   (BC_2,  A(3),    output3, X,  102,  1,  Z)," &
      "2   (BC_6,  A(4),    bidir,   X,  102,  1,  Z)," &
      "3   (BC_6,  A(5),    bidir,   X,  102,  1,  Z)," &
      "4   (BC_3,  NC1,     input,   X)," &
      "5   (BC_6,  A(6),    bidir,   X,  102,  1,  Z)," &
      "6   (BC_6,  A(7),    bidir,   X,  102,  1,  Z)," &
      "7   (BC_6,  A(8),    bidir,   X,  102,  1,  Z)," &
      "8   (BC_6,  A(9),    bidir,   X,  102,  1,  Z)," &
      "9   (BC_6,  A(10),   bidir,   X,  102,  1,  Z)," &
      "10  (BC_6,  A(11),   bidir,   X,  102,  1,  Z)," &
      "11  (BC_6,  A(12),   bidir,   X,  102,  1,  Z)," &
      "12  (BC_6,  A(13),   bidir,   X,  102,  1,  Z)," &
      "13  (BC_6,  A(14),   bidir,   X,  102,  1,  Z)," &
      "14  (BC_6,  A(15),   bidir,   X,  102,  1,  Z)," &
      "15  (BC_6,  A(16),   bidir,   X,  102,  1,  Z)," &
      "16  (BC_6,  A(17),   bidir,   X,  102,  1,  Z)," &
      "17  (BC_6,  A(18),   bidir,   X,  102,  1,  Z)," &
      "18  (BC_6,  A(19),   bidir,   X,  102,  1,  Z)," &
      "19  (BC_6,  A(20),   bidir,   X,  102,  1,  Z)," &
      "20  (BC_6,  A(21),   bidir,   X,  102,  1,  Z)," &
      "21  (BC_6,  A(22),   bidir,   X,  102,  1,  Z)," &
      "22  (BC_6,  A(23),   bidir,   X,  102,  1,  Z)," &
      "23  (BC_6,  A(24),   bidir,   X,  102,  1,  Z)," &
      "24  (BC_6,  A(25),   bidir,   X,  102,  1,  Z)," &
      "25  (BC_6,  A(26),   bidir,   X,  102,  1,  Z)," &
```


241245-89

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

```
"26  (BC_6,  A(27),   bidir,   X,  102,  1,  Z)," &
"27  (BC_6,  A(28),   bidir,   X,  102,  1,  Z)," &
"28  (BC_6,  A(29),   bidir,   X,  102,  1,  Z)," &
"29  (BC_6,  A(30),   bidir,   X,  102,  1,  Z)," &
"30  (BC_6,  A(31),   bidir,   X,  102,  1,  Z)," &
"31  (BC_6,  DP(0),   bidir,   X,  103,  1,  Z)," &
"32  (BC_6,  D(0),    bidir,   X,  103,  1,  Z)," &
"33  (BC_6,  D(1),    bidir,   X,  103,  1,  Z)," &
"34  (BC_6,  D(2),    bidir,   X,  103,  1,  Z)," &
"35  (BC_6,  D(3),    bidir,   X,  103,  1,  Z)," &
"36  (BC_6,  D(4),    bidir,   X,  103,  1,  Z)," &
"37  (BC_6,  D(5),    bidir,   X,  103,  1,  Z)," &
"38  (BC_6,  D(6),    bidir,   X,  103,  1,  Z)," &
"39  (BC_6,  D(7),    bidir,   X,  103,  1,  Z)," &
"40  (BC_6,  DP(1),   bidir,   X,  103,  1,  Z)," &
"41  (BC_6,  D(8),    bidir,   X,  103,  1,  Z)," &
"42  (BC_6,  D(9),    bidir,   X,  103,  1,  Z)," &
"43  (BC_6,  D(10),   bidir,   X,  103,  1,  Z)," &
"44  (BC_6,  D(11),   bidir,   X,  103,  1,  Z)," &
"45  (BC_6,  D(12),   bidir,   X,  103,  1,  Z)," &
"46  (BC_6,  D(13),   bidir,   X,  103,  1,  Z)," &
"47  (BC_6,  D(14),   bidir,   X,  103,  1,  Z)," &
"48  (BC_6,  D(15),   bidir,   X,  103,  1,  Z)," &
"49  (BC_6,  DP(2),   bidir,   X,  103,  1,  Z)," &
"50  (BC_6,  D(16),   bidir,   X,  103,  1,  Z)," &
"51  (BC_6,  D(17),   bidir,   X,  103,  1,  Z)," &
"52  (BC_6,  D(18),   bidir,   X,  103,  1,  Z)," &
"53  (BC_6,  D(19),   bidir,   X,  103,  1,  Z)," &
"54  (BC_6,  D(20),   bidir,   X,  103,  1,  Z)," &
"55  (BC_6,  D(21),   bidir,   X,  103,  1,  Z)," &
"56  (BC_6,  D(22),   bidir,   X,  103,  1,  Z)," &
"57  (BC_6,  D(23),   bidir,   X,  103,  1,  Z)," &
"58  (BC_6,  DP(3),   bidir,   X,  103,  1,  Z)," &
"59  (BC_6,  D(24),   bidir,   X,  103,  1,  Z)," &
"60  (BC_6,  D(25),   bidir,   X,  103,  1,  Z)," &
"61  (BC_6,  D(26),   bidir,   X,  103,  1,  Z)," &
"62  (BC_6,  D(27),   bidir,   X,  103,  1,  Z)," &
"63  (BC_6,  D(28),   bidir,   X,  103,  1,  Z)," &
"64  (BC_6,  D(29),   bidir,   X,  103,  1,  Z)," &
"65  (BC_6,  D(30),   bidir,   X,  103,  1,  Z)," &
"66  (BC_6,  D(31),   bidir,   X,  103,  1,  Z)," &
"67  (BC_3,  IGNNE_,  input,   X)," &
"68  (BC_2,  FERR_,   output3, X,  100,  1,  Z)," &
"69  (BC_3,  UP_,     input,   X)," &
"70  (BC_2,  RES_B,   output,  X,  100,  1,  Z)," &
"71  (BC_3,  RES_A,   input,   X)," &
"72  (BC_3,  NMI,     input,   X)," &
"73  (BC_3,  INTR,    input,   X)," &
"74  (BC_3,  FLUSH_,  input,   X)," &
"75  (BC_3,  RESET,   input,   X)," &
"76  (BC_3,  A20M_,   input,   X)," &
"77  (BC_3,  EADS_,   input,   X)," &
"78  (BC_2,  PCD,     output3, X,  101,  1,  Z)," &
"79  (BC_2,  PWT,     output3, X,  101,  1,  Z)," &
"80  (BC_2,  DC_,     output3, X,  101,  1,  Z)," &
"81  (BC_2,  MIO_,    output3, X,  101,  1,  Z)," &
"82  (BC_2,  BE_(3),  output3, X,  101,  1,  Z)," &
"83  (BC_2,  BE_(2),  output3, X,  101,  1,  Z)," &
"84  (BC_2,  BE_(1),  output3, X,  101,  1,  Z)," &
"85  (BC_2,  BE_(0),  output3, X,  101,  1,  Z)," &
"86  (BC_2,  BREQ,    output3, X,  100,  1,  Z)," &
"87  (BC_2,  WR_,     output3, X,  101,  1,  Z)," &
"88  (BC_2,  HLDA,    output3, X,  100,  1,  Z)," &
"89  (BC_3,  CLK,     input,   X)," &
```


241245-90

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

```
        "90  (BC_3,  AHOLD,   input,   X)," &
        "91  (BC_3,  HOLD,    input,   X)," &
        "92  (BC_3,  KEN_,    input,   X)," &
        "93  (BC_3,  RDY_,    input,   X)," &
        "94  (BC_3,  BS8_,    input,   X)," &
        "95  (BC_3,  BS16_,   input,   X)," &
        "96  (BC_3,  BOFF_,   input,   X)," &
        "97  (BC_3,  BRDY_,   input,   X)," &
        "98  (BC_2,  PCHK_,   output3, X, 101, 1,  Z)," &
        "99  (BC_2,  LOCK_,   output3, X, 101, 1,  Z)," &
        "100 (BC_2,  PLOCK_,  output3, X, 101, 1,  Z)," &
        "101 (BC_2,  BLAST_,  output3, X, 101, 1,  Z)," &
        "102 (BC_2,  ADS_,    output3, X, 101, 1,  Z)," &
        "103 (BC_1,  *,       control, 1)," &          -- DISMISC
        "104 (BC_1,  *,       control, 1)," &          -- DISBUS
        "105 (BC_2,  *,       control, 1)," &          -- DISA(BUS)
        "106 (BC_2,  *,       control, 1)";            -- DISWR

end i486;
```

241245-91

Intel486™ DX2 CPU BSDL Model for Boundary Scan (Continued)

9.0 DEBUGGING SUPPORT

The Intel486 microprocessor family 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 Intel486 microprocessor family. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution break-point 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 Intel486 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)

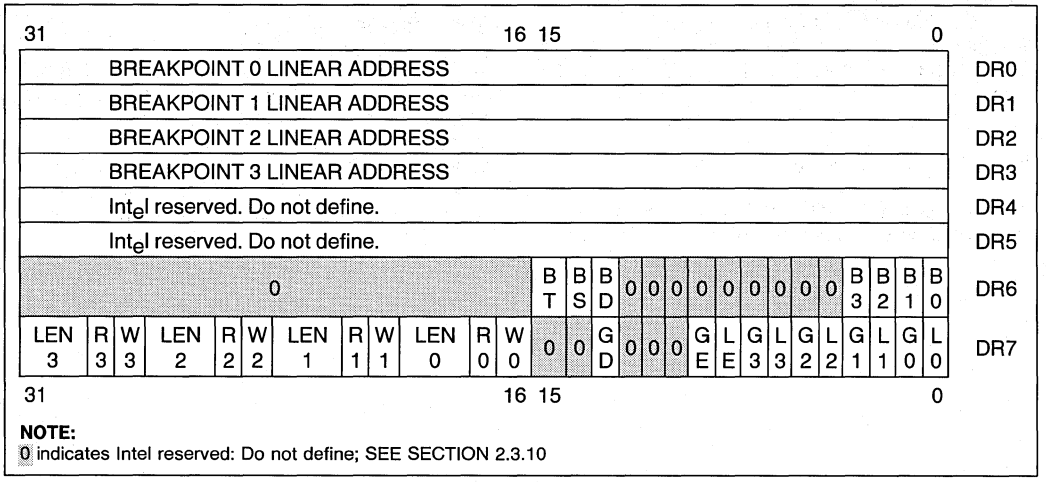
Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. Intel486 microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated 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:



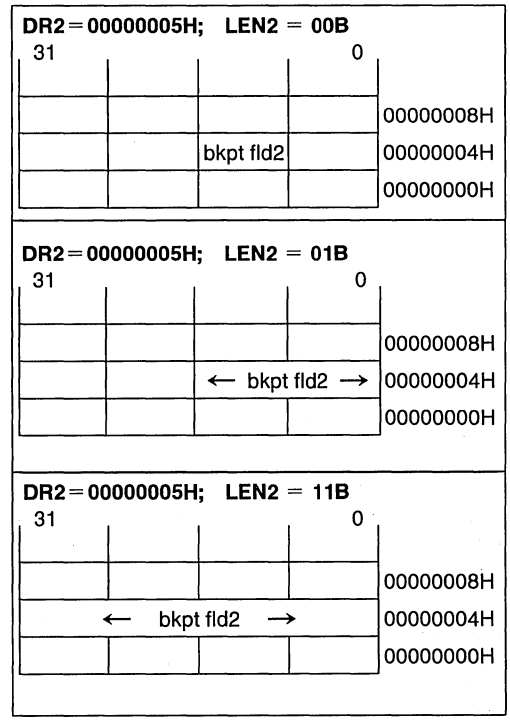
NOTE:
0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

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)

The breakpoint mechanism of the Intel486 microprocessor family differs from that of the Intel386. The Intel486 microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. 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 Intel486 microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Intel486 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 Intel486 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.

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 Intel486 microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Intel486 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 Intel486 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 Intel486 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 Intel486 DX2 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 Intel486 DX2 microprocessor instructions.

10.1 Intel486™ DX2 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 core clock period (e.g., 20 ns for a 50 MHz Intel486 DX2 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 Intel486 DX2 microprocessor instruction core 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 Intel486 DX2 microprocessor to run an external bus cycle. The Intel486 DX2 microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of bus 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 bus clocks for the second and subsequent cycles in a burst read.

w = The number of bus clocks for a write.

The fastest bus the Intel486 DX2 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 bus clocks to reads until the bus is available.
2. Accesses are aligned. Add three core 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 prefetch, 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 core clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
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, 1 core clock **may** be added 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 bus clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b bus clocks.
7. If no write buffer delay, w bus clocks are added only in the case in which all write buffers are full.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 core clock **may** be added to the core clock count shown.
9. No invalidate cycles. Add a delay of 1 bus clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Intel486 DX2 CPU needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 bus clocks + 1 possible core clock 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 core 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) bus clocks to the cache miss penalty.

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS			
MOV = Move:			
reg1 to reg2	1000100W | 11 reg1 reg2	1	
reg2 to reg1	1000101w | 11 reg1 reg2	1	
memory to reg	1000101w | mod reg r/m	1	
reg to memory	1000100w | mod reg r/m	1	
Immediate to reg	1100011w | 11000 reg | immediate data	1	
or	1011w reg | immediate data	1	
Immediate to Memory	1100011w | mod 000 r/m | displacement immediate	1	
Memory to Accumulator	1010000w | full displacement	1	
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 reg	00001111 | 1011z11w | mod reg r/m	3	
PUSH = Push			
reg	11111111 | 11 110 reg	4	
or	01010 reg	1	
memory	11111111 | mod 110 r/m	4	
immediate	011010s0 | immediate data	1	
PUSHA = Push All	01100000	11	
POP = Pop			
reg	10001111 | 11 000 reg	4	
or	01011 reg	1	
memory	10001111 | mod 000 r/m	5	
POPA = Pop All	01100001	9	
XCHG = Exchange			
reg1 with reg2	1000011w | 11 reg1 reg2	3	
Accumulator with reg	10010 reg	3	
Memory with reg	1000011w | mod reg r/m	5	
NOP = No Operation	10010000	1	
LEA = Load EA to Register	10001101 | mod reg r/m		
no index register		1	
with index register		2	

z	instruction
0	MOVZX
1	MOVSX

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS (Continued)			
	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		
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	
register to memory	00TTT00w | mod reg r/m	3	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	U/L
	Instruction **TTT** INC = Increment 000 DEC = Decrement 001		
reg	1111111w | 11 TTT reg	1	
or	01TTT reg	1	
memory	1111111w | mod TTT r/m	3	U/L
	Instruction **TTT** NOT = Logical Complement 010 NEG = Negate 011		
reg	1111011w | 11 TTT reg	1	
memory	1111011w | mod TTT r/m	3	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	
register with memory	0011101w | mod reg r/m	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	
TEST = Logical Compare			
reg1 and reg2	1000010w | 11 reg1 reg2	1	
memory and register	1000010w | mod reg r/m	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	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS (Continued)			
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	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	MN/MX, 3
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	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	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	MN/MX, 3
Word		13/26	MN/MX, 3
Dword		13/42	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	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
INTEGER OPERATIONS (Continued)			
acc. by memory	1111011w | mod111 r/m		
Divisor-Byte		20	
Word		28	
Dword		44	
CBW/CWDE = Convert Byte to Word/ Convert Word to Dword	10011000	3	
CWD/CDQ = Convert Word to Dword/ Convert Dword to Quadword	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	
reg by CL	1101001w | 11 TTT reg	3	
memory by CL	1101001w | mod TTT r/m	4	
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	
Through Carry (RCL and RCR)			
reg by 1	1101000w | 11 TTT reg	3	
memory by 1	1101000w | mod TTT r/m	4	
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 reg2 reg1 | imm 8-bit data	2	
memory by immediate	00001111 | 10TTT100 | mod reg r/m | imm 8-bit data	3	
register by CL	00001111 | 10TTT101 | 11 reg2 reg1	3	
memory by CL	00001111 | 10TTT101 | mod reg r/m	4	
BSWAP = Byte Swap	00001111 | 11001 reg	1	
XADD = Exchange and Add			
reg1, reg2	00001111 | 1100000w | 11 reg2 reg1	3	
memory, reg	00001111 | 1100000w | mod reg r/m	4	U/L
CMPXCHG = Compare and Exchange			
reg1, reg2	00001111 | 1011000w | 11 reg2 reg1	6	
memory, reg	00001111 | 1011000w | mod reg r/m	7/10	6

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	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)			
reg	00001111 | 1001tttn | 11 000 reg	4/3	
memory	00001111 | 1001tttn | mod 000 r/m	3/4	

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

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	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	7
RET = Return from CALL (within segment)			
	11000011	5	
Adding Immediate to SP	11000010 | 16-bit disp.	5	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	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	
MULTIPLE-SEGMENT INSTRUCTIONS			
MOV = Move			
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	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)	000sreg2110	3	
segment reg. (FS or GS)	00001111 | 10 sreg3000	3	
POP = Pop			
segment reg. (ES, SS, or DS)	000sreg2111	3/9	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3001	3/9	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	RV/P, 9
CALL = Call			
Direct intersegment	10011010 | unsigned full offset, selector	18	R, 7, 22
to same level		20	P, 9
thru Gate to same level		35	P, 9
to inner level, no parameters		69	P, 9
to inner level, x parameter (d) words		77+4X	P, 11, 9
to TSS		37+TS	P, 10, 9
thru Task Gate		38+TS	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	R, 7
to same level		20	P, 9
thru Gate to same level		35	P, 9
to inner level, no parameters		69	P, 9
to inner level, x parameter (d) words		77+4X+n	P, 11, 9
to TSS		37+TS	P, 10, 9
thru Task Gate		38+TS	P, 10, 9
RET = Return from CALL			
intersegment	11001011	13	R, 7
to same level		17	P, 9
to outer level		35	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.	14	R, 7
to same level		18	P, 9
to outer level		36	P, 9

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)			
JMP = Unconditional Jump			
Direct intersegment	11101010 | unsigned full offset, selector	17	R, 7, 22
to same level		19	P, 9
thru Call Gate to same level		32	P, 9
thru TSS		42+TS	P, 10, 9
thru Task Gate		43+TS	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	R, 7, 9
to same level		18	P, 9
thru Call Gate to same level		31	P, 9
thru TSS		41+TS	P, 10, 9
thru Task Gate		42+TS	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	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3	
memory, reg	00001111 | 10100011 | mod reg r/m	8	
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	U/L
reg1, reg2	00001111 | 10TTT011 | 11 reg2 reg1	6	
memory, reg	00001111 | 10TTT011 | mod reg r/m	13	U/L
BSF = Scan Bit Forward			
reg1, reg2	00001111 | 10111100 | 11 reg2 reg1	6/42	MN/MX, 12
memory, reg	00001111 | 10111100 | mod reg r/m	7/43	MN/MX, 13
BSR = Scan Bit Reverse			
reg1, reg2	00001111 | 10111101 | 11 reg2 reg1	6/103	MN/MX, 14
memory, reg	00001111 | 10111101 | mod reg r/m	7/104	MN/MX, 15
STRING INSTRUCTIONS			
CMPS = Compare Byte Word	1010011w	8	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	
MOVS = Move Byte/Word	1010010w	7	16
SCAS = Scan Byte/Word	1010111w	6	
STOS = Store Byte/Word from AL/AX/EX	1010101w	5	
XLAT = Translate String	11010111	4	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	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	11110011 1010110w		
C = 0		5	
C > 0		7+4c	16, 18
REP MOVS = Move String	11110011 1010010w		
C = 0		5	
C = 1		13	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	11110011 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. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	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	17	
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	10	
DR6–7 from register	00001111 00100011 11 eee reg	10	
Register from DR6–7	00001111 00100001 11 eee reg	9	
Register from DR0–3	00001111 00100001 11 eee reg	9	
TR3 from register	00001111 00100110 11 011 reg	4	
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	
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 111 r/m	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. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Notes
PROTECTION CONTROL			
ARPL = Adjust Requested Privilege Level			
From register	01100011 | 11 reg1 reg2	9	
From memory	01100011 | mod reg r/m	9	
LAR = Load Access Rights			
From register	00001111 | 00000010 | 11 reg1 reg2	11	
From memory	00001111 | 00000010 | mod reg r/m	11	
LGDT = Load Global Descriptor			
Table register	00001111 | 00000001 | mod 010 r/m	12	
LIDT = Load Interrupt Descriptor			
Table register	00001111 | 00000001 | mod 011 r/m	12	
LLDT = Load Local Descriptor			
Table register from reg.	00001111 | 00000000 | 11 010 reg	11	
Table register from mem.	00001111 | 00000000 | mod 010 r/m	11	
LMSW = Load Machine Status Word			
From register	00001111 | 00000001 | 11 110 reg	13	
From memory	00001111 | 00000001 | mod 110 r/m	13	
LSL = Load Segment Limit			
From register	00001111 | 00000011 | 11 reg1 reg2	10	
From memory	00001111 | 00000011 | mod reg r/m	10	
LTR = Load Task Register			
From Register	00001111 | 00000000 | 11 011 reg	20	
From Memory	00001111 | 00000000 | mod 011 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 | 00000001 | 11 100 reg	2	
To memory	00001111 | 00000001 | 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	
Memory	00001111 | 00000000 | mod 100 r/m	11	
VERW = Verify Write Access			
To register	00001111 | 00000000 | 11 101 reg	11	
To memory	00001111 | 00000000 | mod 101 r/m	11	

Table 10.1. Intel486™ DX2 Microprocessor Integer Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	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	21
If out of range		INT+24	21
IRET = Interrupt Return	11001111		
Real Mode/Virtual Mode		15	
Protected Mode			
To same level		20	9
To outer level		36	9
To nested task (EFLAGS.NT = 1)		TS+32	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**
VM/Intel486 DX2 CPU/286 TSS To Intel486 DX2 CPU TSS	162
VM/Intel486 DX2 CPU/286 TSS To 286 TSS	143
VM/Intel486 DX2 CPU/286 TSS To VM TSS	140

Interrupt Clock Counts Table		
Method	**Value for INT**	
	Cache Hit	**Notes**
Real Mode	26	
Protected Mode		
Interrupt/Trap gate, same level	44	9
Interrupt/Trap gate, different level	71	9
Task Gate	37 + TS	9, 10
Virtual Mode		
Interrupt/Trap gate, different level	82	
Task gate	37 + TS	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. Intel486™ DX2 Microprocessor I/O Instructions Core 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	17	12	32	30	
Variable Port	1 1 1 0 1 1 0 w	17	11	31	30	
OUT = Output to:						
Fixed Port	1 1 1 0 0 1 1 w | port number	19	14	34	32	
Variable Port	1 1 1 0 1 1 1 w	19	13	33	32	
INS = Input Byte/Word from DX Port	0 1 1 0 1 1 0 w	20	13	35	33	
OUTS = Output Byte/Word to DX Port	0 1 1 0 1 1 1 w	20	13	35	33	1
REP INS = Input String	1 1 1 1 0 0 1 1 | 0 1 1 0 1 1 0 w	19+11c	13+11c	33+11c	32+11c	2
REP OUTS = Output String	1 1 1 1 0 0 1 1 | 0 1 1 0 1 1 1 w	20+8c	14+8c	34+8c	33+8c	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. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit 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	
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	
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)	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	
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 001 | 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	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	
ST(i)	11011 000 | 11010 ST(i)	4	
FCOMP = Compare ST(0) with Real and Pop			
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	
ST(i)	11011 000 | 11011 ST(i)	4	

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)			
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5	
FICOM = Compare ST(0) with Integer			
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	
FICOMP = Compare ST(0) with Integer			
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	
FTST = Compare ST(0) with 0.0	11011 001 | 1110 0100	4	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 010 | 1110 1001	5	
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	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8	
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)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	
ST(d) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)	
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)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 1110d ST(i)	10(8–20)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)	

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit 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)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 1110d ST(i)	10(8–20)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)	
FMUL = Multiply real with ST(0)			
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16	
FDIV = Divide ST(0) by Real			
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 110 r/m | s-i-b/disp.	73	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 1111d ST(i)	73	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73	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	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 1111d ST(i)	73	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73	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)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	
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)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	
FISUBR = Integer Subtract Reversed			
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	
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)	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	
FIDIV = Integer Divide			
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	3

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit 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)	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84–86)	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83–87)	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30–32)	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16–20)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70–138)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72–167)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21–30)	
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	241(193–279)	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200–273)	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218–303)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	241(193–279)	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	291(243–329)	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140–279)	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196–329)	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171–326)	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	
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	
Real and Virtual modes 32-bit Address		44	
Protected mode 16-bit Address		34	
Protected mode 32-bit Address		34	

Table 10.3. Intel486™ DX2 Microprocessor Floating Point Core Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit 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	
Real and Virtual modes 32-bit Address		131	
Protected mode 16-bit Address		120	
Protected mode 32-bit Address		120	
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 Intel486 DX2 microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

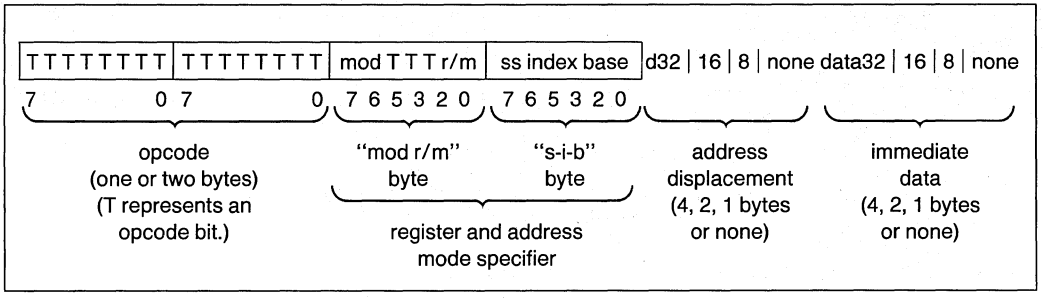


Figure 10.1. General Instruction Format

Table 10.4. Fields within Intel486™ DX2 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

The Intel486 DX2 supports all Intel486 extensions to the 8086/80186/80286 instruction set.

With the Intel486 microprocessor, the 8086/80186/80286 instruction set was 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 Intel486 DX2 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 Intel486 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
110	SI	ESI
111	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 Intel486 DX2 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 tt 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 INTEL486™ DX2 MICROPROCESSOR AND THE INTEL386™ MICROPROCESSOR PLUS THE INTEL387™ MATH COPROCESSOR EXTENSION

The differences between the Intel486 DX2 microprocessor and the Intel386 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 Intel486 DX2 microprocessor bus is significantly faster than the Intel386 microprocessor bus. Differences include an internally doubled 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 Intel486 DX2 microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (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 clearing the CD and NW bit in CR0.
4. The complete Intel387 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 Intel486 DX2 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. In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered, depending upon the execution state the instruction causing exception (see Sections 6.2.13 and 7.2.14). For both of these cases, the Intel387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.
7. 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)
8. 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).
9. 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).
10. 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).
11. The replacement algorithm for the translation lookaside buffer has been changed from a random algorithm 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.
12. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
13. 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.
14. 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 UPGRADE SOCKET

This chapter contains the specifications for the Upgrade Socket for systems based on the Intel486 DX2 Microprocessor. All of the specifications described herein are based on the specifications of the Intel486 DX2 Microprocessor.

One of the most important features of the Intel486 family architecture, compared with previous X86 architectures, is its "end user easy" upgradability via the Upgrade Socket. Inclusion of the Upgrade Socket in systems based on the Intel486 family of microprocessors provides the end user with an easy and cost-effective way to increase system performance. The paradigm of simply installing an additional component into an empty Upgrade Socket to achieve enhanced system performance is familiar to the millions of end users and dealers who have purchased Intel Math CoProcessor upgrades to boost system floating point performance. The first Upgrade Processor for Intel486 DX2 CPU based systems will provide up to 50% integer performance improvement and up to 150% floating point performance improvement over the base system performance. This Upgrade Processor will take advantage of Intel's next generation processor technology to provide this performance improvement.

The Upgrade Processor will implement a superset of the Intel486 DX2 Microprocessor signals. The new signals for the Upgrade Socket, in addition to the Intel486 DX2 CPU signals, support a writeback protocol for the on-chip cache in Intel's next generation processors. Implementation of the cache writeback capability for the Upgrade Socket is optional. Implementation of the Level 1 writeback protocol enables maximum performance gain for the Upgrade Processor over the base Intel486 DX2 system. The signals required to implement this writeback are detailed in a separate document and are marked reserved in this databook.

As a new system architecture feature, the provision of the Upgrade Socket as a means for PC users to take advantage of the ever more rapid advances in software and hardware technology will help to maintain the competitiveness of X86 PC-compatible systems over other architectures into the future.

The majority of upgrade installations which take advantage of the Upgrade Socket will be performed by end users and resellers. Therefore, it is important that the design be "end user easy", and that the amount of training and technical expertise required to install the Upgrade Processors be minimized. Upgrade installation instructions should be clearly described in the system user's manual. In addition, by making installation simple and foolproof, PC manufacturers can reduce the risk of system damage, warranty claims and service calls. Feedback from Intel's Math CoProcessor upgrade customers highlights three main characteristics of end user easy designs: accessible Upgrade Socket location, clear indication of upgrade component orientation, and minimization of insertion force.

Upgrade Socket Location: The Upgrade Socket for Intel486 DX2 Microprocessor based systems is an empty socket which can be located on either the motherboard or modular CPU card. The Upgrade Socket should be easily accessible for installation and readily visible when the PC case is removed. The Upgrade Socket should not be located in a position that requires removal of any other hardware (such as hard disk drives) in order to install the Upgrade Processor. Since Math CoProcessor sockets are typically found near CPU socket on the motherboard, similarly locating the Upgrade Socket near the CPU further adds to the ease of installation.

Component Orientation: The most common mistake made by end users and resellers when installing Math CoProcessor upgrades is incorrect orientation of the chip. This can result in irreversible damage to the chip and/or the PC. To solve this problem, Intel has designed the Upgrade Socket and the Upgrade Processor with a keying mechanism to ensure proper orientation of the upgrade component by the PC user. The keying mechanism for the Upgrade Processor is three missing pins on one corner of the device. To be effective as a keying mechanism the corresponding locations in the socket must be plugged. The Upgrade Socket for Intel486 DX2 CPU-based systems is designed to be backward compatible with the 169-pin Upgrade Socket of Intel486 SX and Intel486 DX systems. In order to maintain compatibility, the Upgrade Socket for Intel486 DX2 Microprocessor systems should include the Key Pin at location E5. In addition, the location of the key corner should be clearly marked on the motherboard or CPU card, for example by silk screening.

Insertion Force: The third major concern voiced by end users refers to how much pressure should be exerted on the upgrade chip and PC board for prop-

er installation without damage. This becomes even more of a concern with the larger components which require up to 200 pounds of pressure for insertion into a standard screw machine socket. This level of pressure can easily result in cracked traces and stress to solder joints. To minimize the risk of system damage, it is recommended that a Zero Insertion Force (ZIF) socket be used for the Upgrade Socket. Designing with a ZIF socket eliminates the need to design in additional structural support to prevent flexing of the PC board during installation, and results in improved end user and reseller product satisfaction due to easy "drop-in" installation.

12.0.1 UPGRADE SOCKET OVERVIEW

The Upgrade Socket for Intel486 DX2 CPU-based systems is designed such that when an Upgrade Processor is installed in the Upgrade Socket, the original CPU relinquishes control of the system to the Upgrade Processor by backing off the bus. The circuit design requirements for the Upgrade Socket are discussed in Section 12.1. In addition to the Upgrade Socket circuits, there are layout considerations for the Upgrade Socket and Upgrade Processor spatial requirements. These issues are discussed in Section 12.2. Because future high-performance Upgrade Processors must function in the Upgrade Socket, the Upgrade Socket heat dissipation specifications must be implemented. Section 12.3 shows the Upgrade Processor heat dissipation requirements for a hypothetical system design at 25 MHz and 33 MHz. Because the system must operate correctly with any Upgrade Processor without a BIOS change, BIOS and software restrictions and recommendations are provided in Section 12.4. Section 12.5 discusses Upgrade Socket test requirements. Finally, Sections 12.6 and 12.7 specify the pinout and electrical characteristics of the Upgrade Processor, respectively.

12.1 Upgrade Circuit Design

The Upgrade Socket for Intel486 DX2 Microprocessor-based systems is designed to reside on the same processor bus as the Intel486 DX2 CPU. This Upgrade Socket specifies a UP# output (Upgrade Present) pin which should be connected directly to the UP# input pin of the Intel486 DX2 Microprocessor. When the Upgrade Processor occupies the Upgrade Socket, the UP# signal (active low) forces the Intel486 DX2 Microprocessor to 3-state all outputs and reduce power consumption. When the Upgrade Processor is not in the Upgrade Socket, a pullup resistor, internal to the Intel486 DX2 Microprocessor, drives UP# inactive and allows the Intel486 DX2 Microprocessor to control the processor bus.

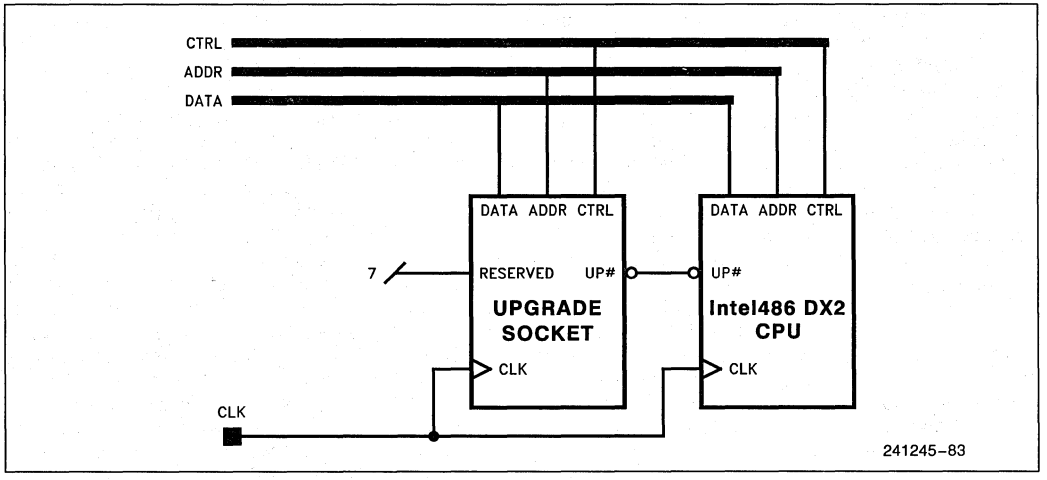


Figure 12-1. Upgrade Socket Circuit Diagram

12.2 Socket Layout

This section discusses four aspects for the Upgrade Socket: compatibility, size, upgradability, and vendors.

12.2.1 BACKWARD COMPATIBILITY

The Upgrade Socket for Intel486 DX2 Microprocessor-based systems is designed to be compatible with the Upgrade Processor for Intel486 DX2 CPU-based systems as well as the Upgrade Processor for Intel486 SX CPU- and Intel486 DX CPU-based systems.

The Upgrade Socket for Intel486 DX2 microprocessor-based systems has a fourth row of contacts around the outside of the 169 contacts defined for the Intel486 SX CPU- and Intel486 DX CPU-based Upgrade Processor sockets. The three inner rows, with inner key pin, are 100% compatible with the 169-pin PGA Upgrade Processor, for Intel486-based systems. For backward compatibility, the inner row key pin location (E5) must be included in the Upgrade Socket for Intel486 DX2 CPU-based systems.

12.2.2 PHYSICAL DIMENSIONS

The Upgrade Socket for Intel486 DX2 microprocessor-based systems is equivalent to a standard 240-lead PGA package with three corner pins removed for orientation purposes. The Upgrade Processor will be provided with a heat sink attached (see Section 12.3), to dissipate heat.

The maximum and minimum dimensions of the Upgrade Processor package with the heat sink are shown in Table 12-1.

Table 12-1. Upgrade Processor, 237-Pin, PGA Package Dimensions with Heat Sink Attached

Dimension (Inches)	Minimum	Maximum
A. Heat Sink Width	1.772	1.790
B. PGA Package Width	1.950	1.975
C. Heat Sink Edge Gap	0.037	0.138
D. Heat Sink Height	0.312	0.360
E. Adhesive Thickness	0.008	0.012
F. Package Height from Stand-Offs	0.140	0.180
G. Total Height from Stand-Offs to Top of Heat Sink	0.460	0.552

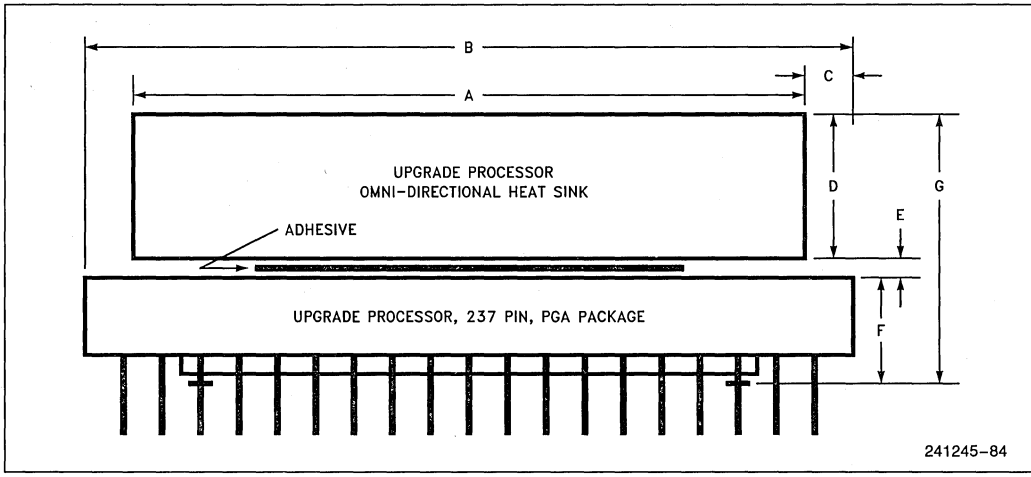


Figure 12-2. Upgrade Processor, 237-Pin, PGA Package with Heat Sink Attached

12.2.3 "END USER EASY"

PC buyers value easy and safe upgrade installation. PC manufacturers can make upgrade component installation in the Upgrade Socket simple and foolproof for the end user and reseller by implementing the suggestions listed in Table 12-2.

Table 12-2. Upgrade Socket and Layout Considerations

"End User Easy" Feature	Implementation
Visible Upgrade Socket	The Upgrade Socket should be easily visible when the PC's cover is removed. Label the Upgrade Socket and the location of Pin 1 by silk screening this information on the PC board.
Accessible Upgrade Socket	Make the Upgrade Socket easily accessible to the end user (i.e., do not place the Upgrade Socket under a disk drive). If a Low Insertion Force (LIF) or screw machine socket is used, position the Upgrade Socket on the PC board such that there is enough clearance around the socket to use a chip removal tool.
Foolproof Chip Orientation	This Upgrade Socket must insure proper orientation of the Upgrade Processor for Intel486 DX2 CPU-based systems and also the Upgrade Processor for Intel486 SX/Intel486 DX CPU-based systems. The 237-pin, PGA package of the Upgrade Processor for Intel486 DX2 CPU-based systems is oriented by the three corner pins that have been removed from the "Pin 1" corner. These three contacts (A1, A2, and B1) should be plugged, such that PGA pins cannot be inserted, in order to assure correct orientation. The 169-pin PGA package of the Upgrade Processor for Intel486 SX/Intel486 DX CPU-based systems is oriented by the "key" pin located in the inside corner of the "Pin 1" corner. All inside contacts (11 innermost rows) should be plugged, except the "key" pin (E5), to ensure correct orientation and alignment. The total number of contacts for the Upgrade Socket is therefore 238; a standard 240-pin socket plus the inside "key" pin and less the three outside corner pins. Supplying a 238-pin socket as the Upgrade Socket eliminates the possibility of end users or resellers damaging the PC board or Upgrade Processor by powering up the system with the Upgrade Processor in an incorrect orientation.
Zero Insertion Force Upgrade Socket	The high pin count of the Upgrade Processor makes the insertion force required for installation in a screw machine PGA socket excessive by end users or resellers. Even most Low Insertion Force (LIF) sockets often require more than 60 lbs. of insertion force. A Zero Insertion Force (ZIF) socket ensures that the chip insertion force does not damage the PC board. If the ZIF socket has a handle, be sure to allow enough clearance for the socket handle. If a LIF or screw machine socket is used, additional PC board support is recommended.
"Plug and Play"	Jumper or switch changes should not be needed to electrically configure the system for the Upgrade Processor.
Thorough Documentation	Describe the Upgrade Socket and the installation procedure for the Upgrade Processor in the PC's User Manual.

12.2.4 LIF AND ZIF SOCKET VENDORS

The following lists provide socket vendor information based on products offered for the Upgrade Socket for Intel486 DX and Intel486 SX CPU Systems

NOTE:
This is not a comprehensive list. Intel has not tested the sockets listed below and cannot guarantee that these sockets will meet every PC manufacturer's specific requirements.

Zero Insertion Force Upgrade Sockets and Vendors:

1. AMP Inc.
   P.O. Box 3608
   Harrisburg, PA 17105-3608
   Tel: (800) 522-6752
   Part Number: TBD
   Contact: Rick Simonic, New Product Manager (717) 561-6143
2. Aries Electronics
   P.O. Box 130
   Frenchtown, NJ 08825
   Tel: (908) 996-6841
   Part Number: TBD
   Contact: Frank Folmsbee, Marketing Manager (908) 996-6841
3. JAE
   599 N. Mathilda Ave., Suite 8
   Sunnyvale, CA 94086
   Tel: (408) 733-0493
   Part Number: TBD
   Contact: Bob Gerleman, Western Sales Manager (408) 733-0493
4. Thomas and Betts
   200 Executive Center Drive
   P.O. Box 24901
   Greenville, SC 29616-2401
   Tel: (803) 676-2900
   Part Number: TBD
   Contact: Scott Roland, Product Marketing Manager (803) 676-2910
5. Yamaichi Electronics
   1420 Koll Circle, Suite B
   San José, CA 95112
   Tel: (408) 452-0797
   Part Number: TBD
   Contact: Jim Bennett, Sales Manager (408) 452-0797

12.3 Thermal Management

The Upgrade Socket must be designed to dissipate the heat generated by the Upgrade Processor. In the following Sections the airflow required over the Upgrade Socket is calculated for a hypothetical system design where there is a maximum ambient temperature of 45°C inside the system box.

12.3.1 THERMAL CALCULATIONS FOR HYPOTHETICAL SYSTEM

The maximum temperature specification for the Upgrade Processor is 80°C (with heat sink attached). Therefore, the temperature of the heat sink surface (T_S) cannot exceed 80°C under the worst case specified operating conditions for the system. The variables which affect the heat sink temperature include ambient temperature inside the system box (T_A), V_{CC}, and I_{CC}. An equation for the approximate Upgrade Processor temperature (T_S) is:

$T_S = T_A + \text{Power} * \theta_{SA}$ where $\text{Power} = V_{CC} * I_{CC}$

In the above equation, the variables under worst case conditions are specified as follows:

T_S: Specified as 80°C for the Upgrade Processor.

T_A: Specified by the PC manufacturer for the worst case system operating conditions.

V_{CC}: Specified for the Upgrade Processor as 5V.

I_{CC}: Specified for the Upgrade Processor and related to clock frequency.

θ_{SA}: $\theta_{SA} = \theta_{JA} - \theta_{JS}$.
θ_{JA} and θ_{JS} are specified in Table 12-4.

I_{CC} is dependent upon the system's V_{CC}, bus loading, software code sequences, and silicon process variations. For the Upgrade Socket specifications, the typical I_{CC} value will be derived by testing a sample of components under the following worst case conditions: V_{CC} = 5.3V, full D.C. current loads on all output pins, and running a file with the predicted worst case software code sequences at the specified frequency. I_{CC} typical is not a guaranteed specification.

The I_{CC} maximum values in Table 12-3 is the best known design estimate and should be used as a guaranteed maximum specification.

Table 12-3. Upgrade Processor Typical and Maximum I_{CC} Values

System Frequency (MHz)	Upgrade Processor Typical I_{CC} (mA)	Upgrade Processor Maximum I_{CC} (mA)
25	TBD	1600
33	TBD	1900

The Upgrade Processor for Intel486 DX2 CPU-based systems will be provided with a heat sink. The θ_{JS} and θ_{JA} values for the Upgrade Processor with a heat sink are shown in Table 12-4. The maximum T_A values for the 25 MHz and 33 MHz Upgrade Processor are shown in Table 12-5. The maximum T_A values shown in Table 12-5 were calculated using T_S = 80°C, V_{CC} = 5V, the maximum I_{CC} value shown in Table 12-3, and the θ_{JC} and θ_{JS} values shown in Table 2-4.

Table 12-4. Thermal Resistance (°C/W) θ_{JS} and θ_{JA}

Upgrade CPU with Heat Sink	θ_{JS}	Airflow (ft/min, LFM)				
	2.5°C/W	0	200	400	600	800
θ_{JA} (°C/W)		11.5	8.3	6.0	5.0	4.7

Table 12-5. Maximum T_A for 25 MHz and 33 MHz Upgrade Processor

Upgrade Processor with Heat Sink		Linear Airflow (ft/min)				
	f_{CLK} (MHz)	0	200	400	600	800
T_A (°C)	25	8	33	52	60	62
	33	−5	24	46	56	59

12.3.2 HEAT SINKS

The Upgrade Processor will be shipped with a heat sink attached. Because of the heat sink attached to the Upgrade Processor, it is vital that vertical clearance is provided for the Upgrade Socket. The height of the package and the heat sink is shown in Table 12-1 in Section 12.2.2.

12.3.3 AIRFLOW

The Upgrade Socket for Intel486 DX2 Microprocessor-based systems must be positioned such that the airflow over the socket has the velocity, volume, and ambient temperature to satisfy the thermal calculations in Section 12.3.1. Because the heatsink attached to the Upgrade Processor is omni-directional, the specified airflow may come from any direction. However, because the Upgrade Processor will generate more power than the Intel486 DX2 Microprocessor, it is recommended that the airflow reach the Upgrade Socket before or at the same time that it reaches the Intel486 DX2 CPU.

NOTE:
An airflow of at least 400 linear feet per minute (LFM) will be required to cool the Upgrade Processor at 33 MHz. It is essential that care be taken to ensure sufficient airflow over the Upgrade Socket.

12.4 BIOS and Software

The following should be considered when designing the Upgrade Socket for a Intel486 DX2 microprocessor-based system.

12.4.1 UPGRADE PROCESSOR DETECTION

The component identifier and stepping/revision identifier for the Upgrade Processor for Intel486 DX2 CPU-based systems is readable in the DH and DL registers respectively, immediately after RESET, where

DH = 14h

DL = 30h–3Fh

As with the Intel486 DX2 microprocessor specification, it is recommended that the BIOS save the contents of the DX register, immediately after RESET, so that this information can be used later, if required.

12.4.2. TIMING DEPENDENT LOOPS

The Upgrade Processor for Intel486 DX2 microprocessor-based systems executes instructions at twice the frequency of the input clock. This Upgrade Processor also will use advanced design techniques to decrease the number of clocks per instruction (cpi) from that of the Intel486 DX2 microprocessor. Thus software, such as instruction-based timing loops, will execute faster on this Upgrade Processor than on either the Intel486 DX CPU or the Intel486 DX2 microprocessor at the same input clock frequency. Instructions such as NOP, LOOP, and JMP $+2 are frequently used by the BIOS to implement timing loops that are required, for example, to enforce recovery time between consecutive accesses for I/O devices. These instruction-based, timing-loop implementations may require modification to be compatible with this Upgrade Socket.

In order to avoid any incompatibilities, timing loops must be implemented in hardware rather than in software. This provides transparency and also does not require any change in BIOS or I/O device drivers in the future when moving to higher processor clock speeds.

As an example, a timing loop may be implemented as follows: The software performs a dummy I/O instruction to an unused I/O port. The hardware for the bus controller logic recognizes this I/O instruction and delays the termination of the I/O cycle by keeping RDY# or BRDY# deasserted for the appropriate amount of time.

12.5 Test Requirements

The Upgrade Socket's electrical functionality can be verified by fully testing the PC with a populated Upgrade Socket. We recommend that the system is tested with all **available** Upgrade Processors to ensure that there are no BIOS issues. This Upgrade Socket can be electrically tested with the Upgrade Processor for Intel486 SX/Intel486 DX CPU-based systems. The Upgrade Processor for Intel486 DX2 CPU-based systems should also be used to test the hardware and software when it is available. The BIOS requirements to maintain compatibility with all Upgrade Processors are discussed in Section 12.4 of this document. All Upgrade Processors undergo thorough application software compatibility testing prior to their introduction.

12.6 Upgrade Socket Pinout

The Upgrade Socket for Intel486 DX2 Microprocessor-based systems specifies 238 contacts. The 238 contacts correspond to a standard 240-pin socket with one "KEY" contact inside and three "orientation" contacts plugged on the outside corner. The "KEY" contact provides backward compatibility for the Intel487 SX Math CoProcessor and the Upgrade Processor for Intel486 SX/Intel486 DX CPU-based systems. The three contacts plugged on the outside corner ensure proper orientation for the Upgrade Processor for Intel486 DX2 CPU-based systems. Additionally, all 120 inner contacts of a standard 240-pin socket (the inner 11 rows minus the "KEY" pin) should be plugged to ensure proper alignment of the 169-pin PGA Intel487 SX Math CoProcessor and Upgrade Processor.

12.6.1 PINOUT

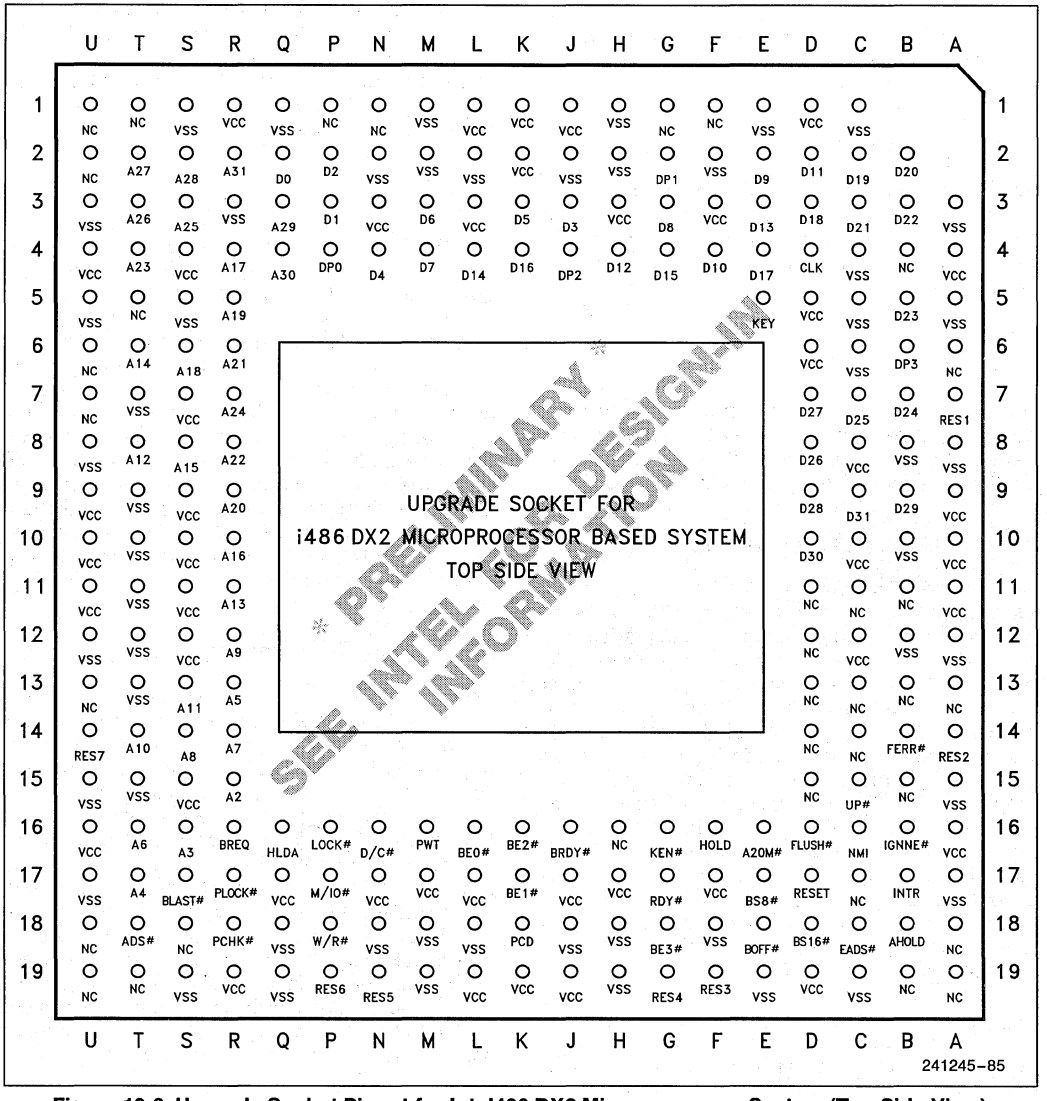


Figure 12-3. Upgrade Socket Pinout for Intel486 DX2 Microprocessor System (Top Side View)

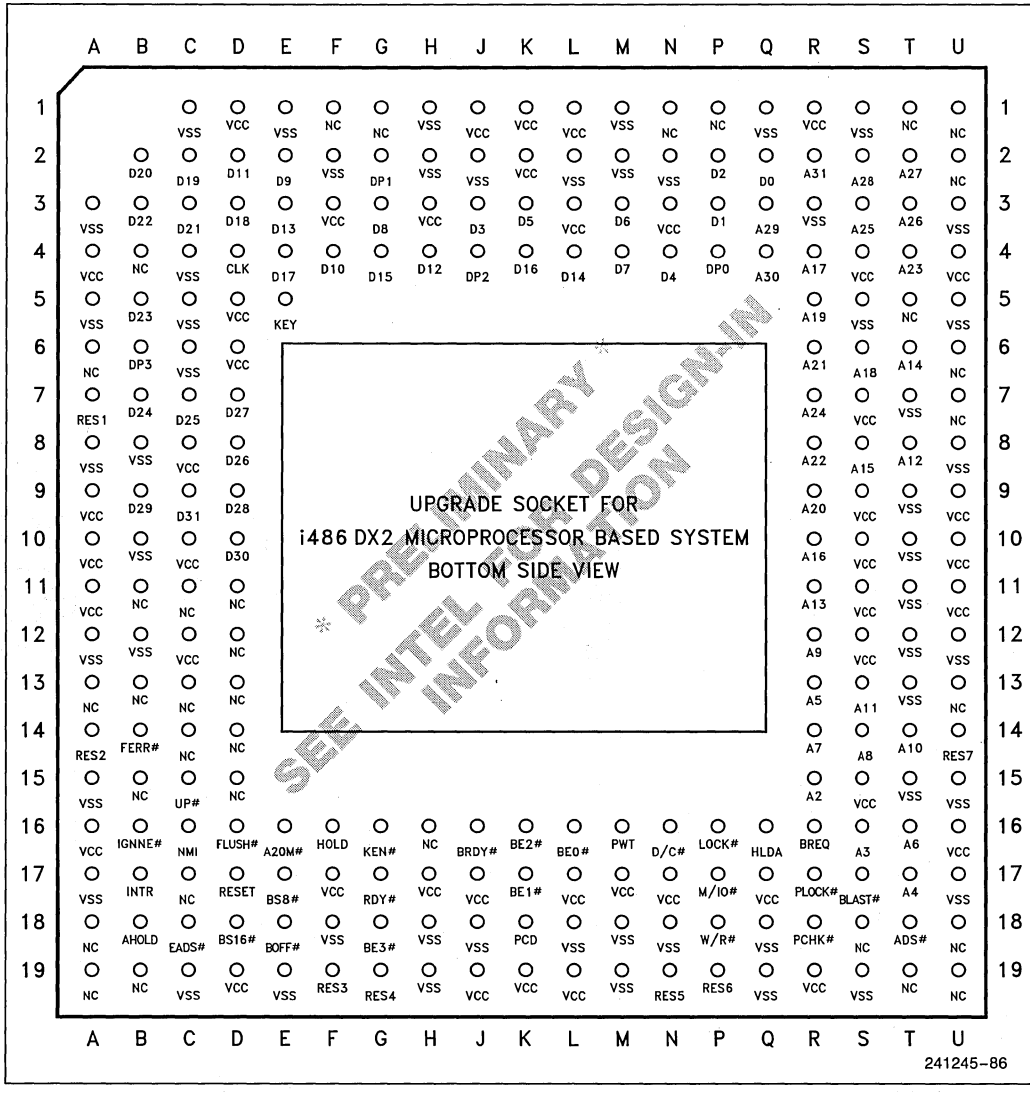


Figure 12-4. Upgrade Socket Pinout for Intel486 DX2 Microprocessor System (Bottom Side View)

Table 12-6. Upgrade Socket Pin Cross Reference

Address		Data		Control		Control		N/C	V_{CC}			V_{SS}	
A_2	R15	D_0	Q2	A20M#	E16	PLOCK#	R17	A6	A4	C8		A3	B8
A_3	S16	D_1	P3	ADS#	T18	RDY#	G17	A18	A9	C10		A5	B10
A_4	T17	D_2	P2	AHOLD	B18	RESET	D17	A19	A10	C12		A8	B12
A_5	R13	D_3	J3	BE0#	L16	W/R#	P18	B4	A11	D5		A12	C4
A_6	T16	D_4	N4	BE1#	K17	UP#	C15	B11	A16	D6		A15	C5
A_7	R14	D_5	K3	BE2#	K16	**Reserved**		B13	D1	F3		A17	C6
A_8	S14	D_6	M3	BE3#	G18			B15	D19	F17		C1	F2
A_9	R12	D_7	M4	BLAST#	S17	RES1	A7	B19	J1	H3		C19	F18
A_{10}	T14	D_8	G3	BOFF#	E18	RES2	A14	C11	J19	H17		E1	H2
A_{11}	S13	D_9	E2	BRDY#	J16	RES3	F19	C13	K1	J17		E19	H18
A_{12}	T8	D_{10}	F4	BREQ#	R16	RES4	G19	C14	K19	K2		H1	J2
A_{13}	R11	D_{11}	D2	BS8#	E17	RES5	N19	C17	L1	L3		H19	J18
A_{14}	T6	D_{12}	H4	BS16#	D18	RES6	P19	D11	L19	L17		M1	L2
A_{15}	S8	D_{13}	E3	CLK	D4	RES7	U14	D12	R1	M17		M19	L18
A_{16}	R10	D_{14}	L4	D/C#	N16	**Position**		D13	R19	N3		Q1	M2
A_{17}	R4	D_{15}	G4	DP0	P4	KEY	E5	D14	U4	N17		Q19	M18
A_{18}	S6	D_{16}	K4	DP1	G2	PLUG	A1	D15	U9	Q17		S1	N2
A_{19}	R5	D_{17}	E4	DP2	J4	PLUG	A2	F1	U10	S4		S19	N18
A_{20}	R9	D_{18}	D3	DP3	B6	PLUG	B1	G1	U11	S7		U3	Q18
A_{21}	R6	D_{19}	C2	EADS#	C18			H16	U16	S9		U5	R3
A_{22}	R8	D_{20}	B2	FERR#	B14			N1		S10		U8	S5
A_{23}	T4	D_{21}	C3	FLUSH#	D16			P1		S11		U12	T7
A_{24}	R7	D_{22}	B3	HLDA	Q16			S18		S12		U15	T9
A_{25}	S3	D_{23}	B5	HOLD	F16			T1		S15		U17	T10
A_{26}	T3	D_{24}	B7	IGNNE#	B16			T5					T11
A_{27}	T2	D_{25}	C7	INTR	B17			T19					T12
A_{28}	S2	D_{26}	D8	KEN#	G16			U1					T13
A_{29}	Q3	D_{27}	D7	LOCK#	P16			U2					T15
A_{30}	Q4	D_{28}	D9	M/IO#	P17			U6					
A_{31}	R2	D_{29}	B9	NMI	C16			U7					
		D_{30}	D10	PCD	K18			U13					
		D_{31}	C9	PCHK#	R18			U18					
				PWT	M16			U19					

NOTE:
The Upgrade Socket for Intel486 DX2 Microprocessor-based systems provides orientation guides for Upgrade Processors via one "KEY" pin and three corner plugs.

12.6.2 PIN DESCRIPTION

The signal pin descriptions for the Upgrade Processor are identical to the pin descriptions for the Intel486 DX2 Microprocessor except for the Upgrade Present pin (UP#) and KEY pin. The pin descriptions for these two signals are shown in Table 12-7.

12.6.3 RESERVED PIN SPECIFICATION

Seven pins in the Upgrade Socket are defined as reserved. The function of these pins is documented separately. These signals will be used to implement a Write Back level 1 (on-chip) cache protocol. These signals must not be connected unless they are used to implement a level 1 Write Back solution using the information available separately.

Table 12-7. Upgrade Socket Pin Description

Symbol	Type	Name and Function
Intel486 DX2 CPU INTERFACE		
UP#	O	The *Upgrade Present* pin is used to signal the Intel486 DX2 microprocessor to float its outputs and get-off the bus. It is active low and is never floated. UP# is driven low at power-up and remains active for the entire duration of the Upgrade Processor operation.
KEY PIN		
KEY		The Key pin is an electrically non-functional pin which provides backward compatibility to the Upgrade Processor for Intel486 SX/Intel486 DX CPU-based systems and is used to ensure correct orientation for 169-pin upgrade products. Proper orientation of the Upgrade Processor for Intel486 DX2 Microprocessor systems (237-pin PGA) is insured by the three socket contacts which must be plugged (A1, A2, and B1); the Upgrade Processor will not have pins in these locations.

12.7 D.C./A.C. Specifications

The electrical specifications in this section represent the electrical interface of the Upgrade Processor for a Intel486 DX2 microprocessor-based system. The Upgrade Processor will be compatible to the maximum ratings and A.C. Specifications of the Intel486 DX2 Microprocessor. Table 12-8 provides the D.C. Operating Conditions for the Upgrade Processor.

Table 12-8. Upgrade Socket D.C. Parametric Values(1)

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 2)
V_{OH}	Output High Voltage	2.4		V	(Note 3)
I_{CC}	Power Supply Current CLK = 25 MHz CLK = 33 MHz		 1600 1900	mA	
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		13	pF	F_C = 1 MHz(7)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz(7)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz(7)

NOTES:
1. Functional operating range: V_{CC} = 5V; T_S = 0°C to +80°C.
2. This parameter is measured at:
   — Address, Data, BEn 4.0 mA
   — Definition, Control 5.0 mA
3. This parameter is measured at:
   — Address, Data, BEn −1.0 mA
   — Definition, Control −0.9 mA
4. This parameter is for inputs without pullups or pulldowns and $0 \leq V_{IN} \leq V_{CC}$.
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.
7. Not 100% tested.

13.0 CONVERTING AN EXISTING Intel486™ DX CPU DESIGN

Converting an Intel486 DX CPU system design to an Intel486 DX2 CPU design provides more performance for a small difference in cost. Three conversion possibilities are available as shown in Table 13.1. Migrating from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU could increase performance by 35%, and migrating from a 25 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU could increase performance by an average of 70%. See the *Intel486™ DX2 Microprocessor Performance Brief* (Order #241254) for more details on performance. Conversion can be as easy as replacing one or two devices.

Table 13.1. Converting Intel486™ DX CPU Designs to Intel486™ DX2 CPU Designs

Initial Design (Intel486 DX CPU)	Converted Design (Intel486 DX2 CPU)	Typical Performance Gain
25 MHz	50 MHz	70%
33 MHz	50 MHz	35%
33 MHz	66 MHz	70%

A few system details should be checked first to be sure the design is ready for the Intel486 DX2 CPU. Check with your BIOS vendor to be sure any BIOS issues have been resolved. The BIOS for the Intel486 DX CPU may have timing loops. Since the Intel486 DX2 CPU runs instructions twice as fast as the Intel486 DX CPU, timing loops may no longer return the required results. Most of the timing loops have been removed from a standard BIOS, but there may be some versions that need updating. Another BIOS issue that may not be critical, is the processor identification code. There are different ID codes in the Intel486 DX CPU and the Intel486 DX2 CPU. The BIOS may need to be modified to identify the Intel486 DX2 CPU properly. Refer to Table 6.3 for the component ID code.

Other system parameters to watch out for are the thermal and power supply specifications. Table 14.2 details the Power Supply Current information, and Table 15.2 outlines the Thermal Resistance. Since the processor core runs twice as fast for the same input clock, the Intel486 DX2 CPU uses more power and generates more heat than the Intel486 DX CPU. Be sure that there is adequate cooling and adequate power built into the design. A heat sink is a recommended method to help provide cooling for the Intel486 DX2 CPU.

The system checks mentioned above are common to all conversions from an Intel486 DX CPU to an Intel486 DX2 CPU regardless of the speed of the processor or system.

A few system implications exist for converting from one frequency to another as shown in Figure 13.1. The first case is migrating from a 25 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU (the bus runs at the same speed for both parts). System hardware modifications need not be made to plug in the 50 MHz Intel486 DX2 CPU and achieve the desired performance. When all instructions are running out of the on-chip cache, performance increases by a maximum of 100%.

The second case is migrating from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. This conversion is a two step process. The first step is to change the frequency source for the CPU from 33 MHz to 25 MHz. The Intel486 DX2 CPU can then be inserted into the system. Without any tuning of the memory and depending on the application, only a modest performance improvement may be observed. For programs running entirely out of the on-chip cache, however, performance can increase up to 50%. There are many factors which contribute to the performance of an application, including whether there is a second-level (L2) cache, the cache size if present, the memory subsystem design, and many other factors beyond the scope of this introduction. A comprehensive memory subsystem design guide, *AP469: Cache and Memory Design Considerations for Intel486™ DX2 Microprocessor,* is available which includes more detailed information on how each of these many factors affects Intel486 DX2 CPU-based system performance.

Because the Intel486 DX2 core runs twice as fast as its external bus, it is more sensitive to wait states. The Intel486 DX2 CPU needs to be fed instructions and data quickly. Either a high performance memory subsystem is needed or an external cache should be added. An external cache benefits the Intel486 DX2 CPU even more than it benefits the Intel486 DX CPU, and helps to hide the effects of a slower memory subsystem. The Intel486 DX CPU gains an average of 3%–9% performance by the addition of a second-level cache, but the Intel486 DX2 CPU gains an average of 20-30% performance by adding a second level cache. It should be noted however, that an external cache does not preclude the benefits of tuning the memory subsystem.

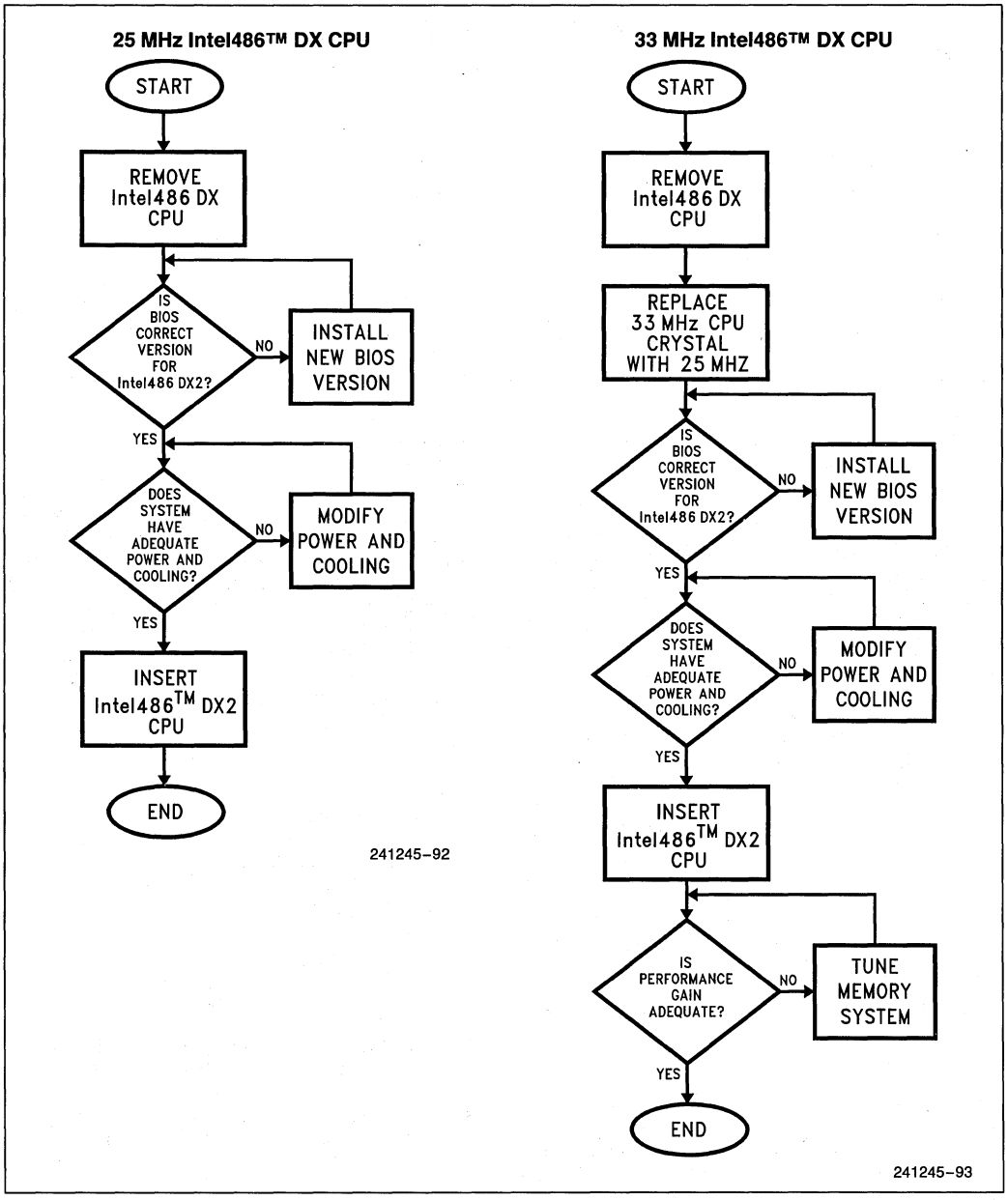


Figure 13.1. Flowchart for Intel486™ DX CPU to Intel486™ DX2 CPU Conversion

The graph shown in Figure 13.2 shows a set of benchmarks known to have a poor cache hit rate. This is shown for purposes of memory tuning and not to be taken as absolute performance. Please refer to the *Intel486™ DX2 Microprocessor Performance Brief* (Order # 241254) for performance details.

With the absence of a second-level cache, the memory subsystem becomes critical to gaining performance when converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. For slow memory systems without tuning, the 50 MHz Intel486 DX2 CPU can possibly run slower than the 33 MHz Intel486 DX CPU (see Figure 13.2). By tuning the memory design, the 50 MHz Intel486 DX2 CPU can reach equivalent performance to the 33 MHz Intel486 DX CPU running applications with low cache hit rates, and increase performance for applications with higher hit rates. Tuning the memory design can be done easily by either removing a wait state from the memory design (if timing permits), and/or adding faster DRAM and removing wait state(s) from the memory design.

Changing the wait state configuration for the system is often done by programming the DRAM controller in the chip set on the motherboard. Each chip set is programmed differently at the BIOS level, requiring a BIOS modification. For testing purposes, the chip set may be programmed on the fly from a DOS program if the register locations are known.

A typical ISA chip set with an L2 cache, for example, allows 6-4-4-4 bus cycles at 33 MHz with 80 ns DRAMs for the Intel486 DX CPU. Without modifying the memory subsystem, the 50 MHz Intel486 DX2 CPU achieved an average of 7%–12% improvement over the 33 MHz Intel486 DX CPU. By reducing the bus cycles at 25 MHz to 5-2-2-2 (still with 80 ns DRAMs), the 50 MHz Intel486 DX2 CPU improved to achieve an average of 15%–20% more performance than the 33 MHz Intel486 DX CPU. By replacing the DRAMs with faster devices (70 ns) bus cycles could be reduced to 4-2-2-2 at 25 MHz, improving the performance of the 50 MHz Intel486 DX2 CPU even greater.

A typical EISA solution is shown in Figure 13.3, using the Intel 82350DT Chip Set, which was specifically designed to permit variation in CPU type and frequency.

In an 82350DT based design the memory subsystem is controlled by the combination of a flexible Programmable State Tracker (PST) and a highly configurable 82359 DRAM Controller. The PST, which is typically implemented as a 3 to 5 PLD solution, is responsible for converting the CPU's clock-dependent handshake protocol into a clock-less memory interface protocol. The 82359 in turn uses the clock-less memory interface protocol to control main memory as well as to forward host CPU cycles to the EISA bus if needed. As a result of this clock-

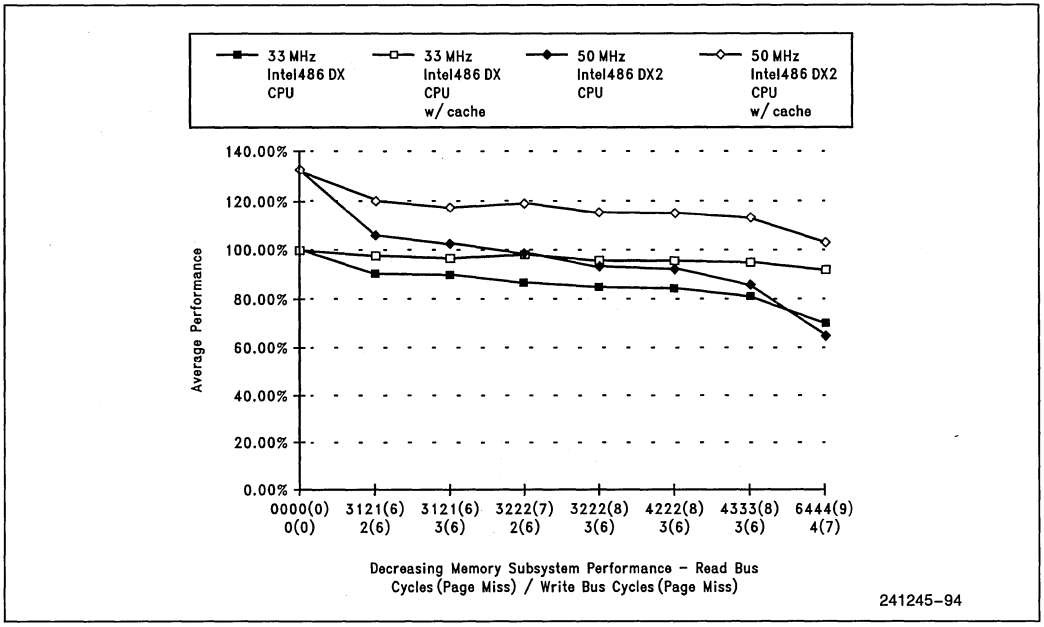


Figure 13.2. Performance of 50 MHz Intel486™ DX2 CPU vs. 33 MHz Intel486™ DX CPU

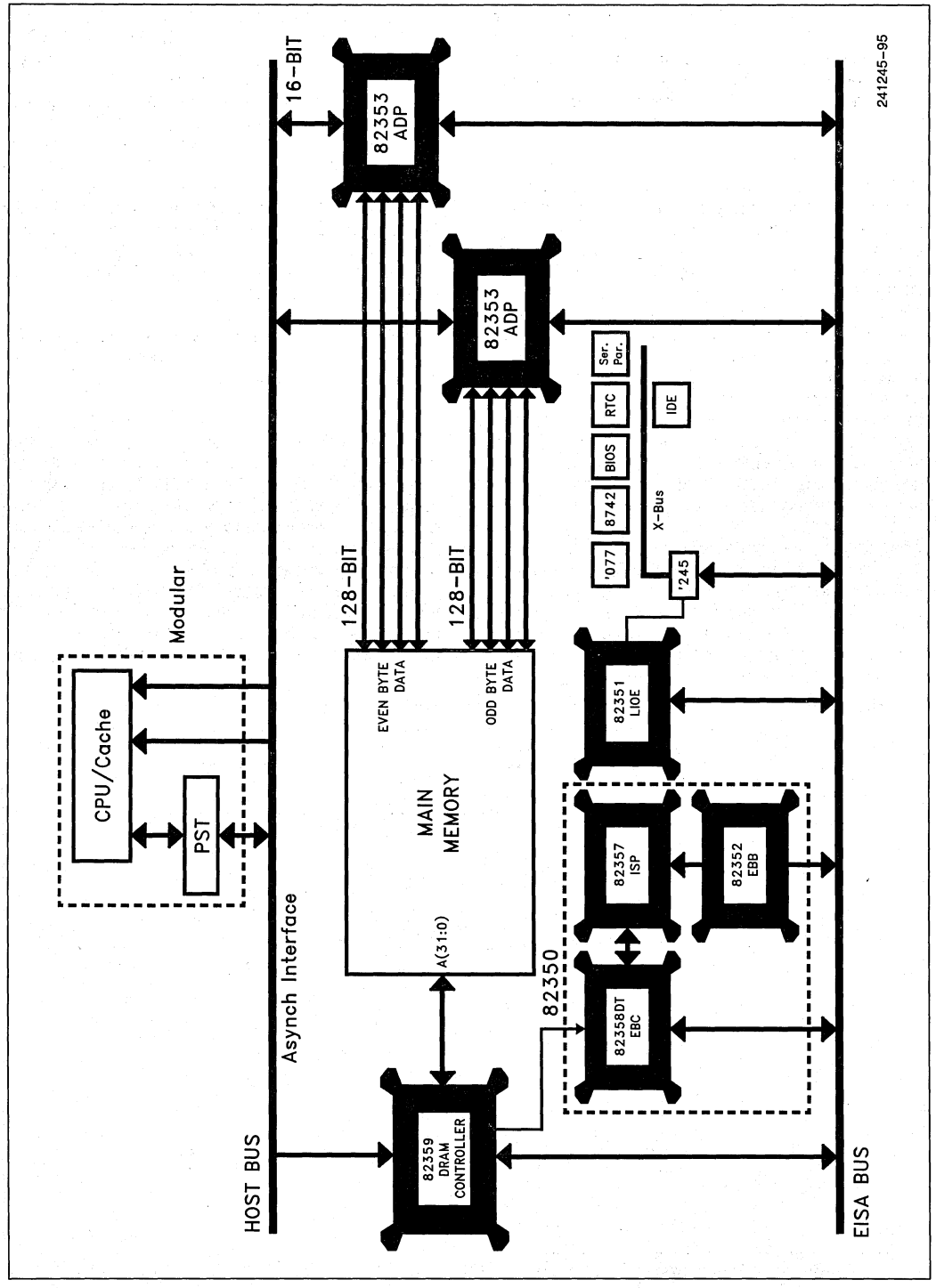


Figure 13.3. Typical 82350DT System Architecture

less protocol, the system design becomes independent of the CPU and cache combination being used and the speed of the CPU clock. Therefore, whenever a new CPU and cache combination is to be used with an 82350DT based system, the only re-design that is necessary is to the CPU subsection, leaving the main memory and EISA subsections unchanged. Typically, this CPU subsection re-design entails modifying only the PST functionality and the programmable registers of the 82359.

There are five steps to determine whether wait states can be removed from the main memory design of an 82350DT system when converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. An overview of these five steps is covered here; the system designer is referred to the *82350DT EISA Chip Set Design Guide* (Order #296911) for detailed design information.

1. Calculate the 82359 delay line tap values for optimal 25 MHz operation
2. Determine all memory cycle lengths for operation at 25 MHz
3. Re-evaluate the PST design (deterministic & snoop cycle trackers)
4. Modify the PLD equations for the PST
5. Update system BIOS to reflect new 82359 programmable register values

Step one is to perform a timing analysis of the main memory subsystem to determine the minimum number of CPU clocks required for each memory cycle. Once the memory cycle lengths are known, the PST design can be re-evaluated with the goal of removing unnecessary wait states. Before the system designer can determine the memory cycle lengths, the delay line timings of the 82359 must be analyzed.

The timing for the DRAM control and address signals of the 82359 is based on four integrated asynchronous delay line elements which can be controlled by the 82359 programmable registers. Once the delay line tap values have been verified or modified, the system designer should determine the minimum number of CPU clocks required for the different memory cycles (i.e., read page hit, page miss write, burst read, etc.). Armed with the delay line tap programming values and the number of CPU clocks required for each type of memory cycle, the system designer can now evaluate the PST design to determine if any unneeded wait states can be removed.

The PST for an 82350DT based system can be separated into four primary functions: bus cycle control (including arbitration and posted write control), cycle length tracking, CPU "Ready" generation, and cache invalidation control. Although the PST contains a number of state machines only a few of the state machines need to be re-evaluated to determine whether any wait states can be removed for converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU. The state machines that need to be re-evaluated are the deterministic cycle tracker and the snoop cycle tracker.

The deterministic cycle tracker is responsible for generating RDY or BRDY to the CPU and cache for cycles that are deterministic in length. All main memory cycles, except locked cycles which require EISA arbitration, are deterministic cycles. Once the deterministic cycle tracker knows that a deterministic cycle is occurring, it uses the 82359 CYCLN(2:0) and PAGEHIT# outputs to determine when to generate RDY or BRDY to the CPU. For burst cycles the deterministic cycle tracker also uses the IF(1:0) and SPEED(1:0) outputs of the 82359 for generating BRDY. After this analysis has been completed the system designer can then determine if the deterministic cycle tracker can be optimized to take advantage of converting from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU.

The other state machine that must be re-evaluated for correct system functionality is the snoop cycle tracker. The snoop cycle tracker state machine design must meet two goals; respond to a SNUPRQ with a SNUPACK# within 180 ns (based on an EISA burst write cycle), and maintain a snoop cycle frequency capability that is equal to or faster than the fastest system bus master write cycle frequency. Due to the change of CPU clock frequency from 33 MHz to 25 MHz, the system designer must re-evaluate the snoop cycle tracker state machine to determine if the two design goals are still being met in the 50 MHz Intel486 DX2 CPU implementation.

In conclusion, when converting an 82350DT based design from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU the system designer must re-evaluate the main memory cycle timings to determine whether the PST and 82359 programmable registers need to be modified to take advantage of the increased performance benefits of the 50 MHz Intel486 DX2 CPU. Once the system designer has decided to modify the PST and the 82359 programmable registers, the conversion from a 33 MHz Intel486 DX CPU to a 50 MHz Intel486 DX2 CPU is usually just as simple as modifying the PLD equations, re-programming the PST PLDs, and upgrading the system BIOS to reflect the new 82359 programmable register values.

14.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the Intel486 DX2 microprocessor, and its electrical specifications.

14.1 Power and Grounding

14.1.1 POWER CONNECTIONS

The Intel486 DX2 microprocessor is implemented in CHMOS V 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 Intel486 DX2 microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the Intel486 DX2 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.

14.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the Intel486 DX2 microprocessor. The Intel486 DX2 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 Intel486 DX2 microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

14.1.3 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.

14.2 Maximum Ratings

Table 14.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 14.3 D.C. Specifications and 14.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the Intel486 DX2 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 . . . −65°C to +110°C

Storage Temperature . . . . . −65°C to +150°C

Voltage on Any Pin with Respect to Ground . . . . . −0.5 to V_{CC} + 0.5V

Supply Voltage with Respect to V_{SS} . . . . . −0.5V to +6.5V

14.3 Intel486 DX2 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 14-2. 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 (66 MHz) (50 MHz)		 1200 950	mA	(Note 3)
I_{CCF}	Power Supply Current in Power Down Mode		50	mA	(Note 8)
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		13	pF	F_C = 1 MHz (Note 7)
C_O	I/O or Output Capacitance		17	pF	F_C = 1 MHz (Note 7)
C_{CLK}	CLK Capacitance		15	pF	F_C = 1 MHz (Note 7)

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:
   775 mA @ 50 MHz
   975 mA @ 66 MHz
4. This parameter is for inputs without internal pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with internal pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with internal pullups and V_{IL} = 0.45V.
7. Not 100% tested.
8. The I_{CCF} specification in the above table is a target value. It has not been tested.

14.4 A.C. Specifications

The A.C. specifications, given in Table 14.3, 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 14.1–14.7. All timings are referenced to 1.5V unless otherwise specified. Inputs must be driven to the voltage levels indicated by Figure 14.3 when A.C. specifications are measured. Intel486 DX2 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum Intel486 DX2 microprocessor delay times are hold times provided to external circuitry. Intel486 DX2 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 Intel486 DX2 microprocessor operation.

Table 14.4.1. 50 MHz Intel486 DX2 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 Intel486 DX2
t_1	CLK Period	40	125	ns	14.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	14.1	at 2V
t_3	CLK Low Time	14		ns	14.1	at 0.8V
t_4	CLK Fall Time		4	ns	14.1	2V to 0.8V
t_5	CLK Rise Time		4	ns	14.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	19	ns	14.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		28	ns	14.6	(Note 1)
t_8	PCHK# Valid Delay	3	24	ns	14.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	24	ns	14.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		28	ns	14.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	20	ns	14.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		28	ns	14.6	(Note 1)
t_{12}	EADS# Setup Time	8		ns	14.2	
t_{13}	EADS# Hold Time	3		ns	14.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	14.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	14.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	14.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	14.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	8		ns	14.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	14.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	8		ns	14.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	14.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	14.2, 14.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	14.2, 14.3	

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 14.4.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel486 DX2 output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 14.4.2 66 MHz Intel486 DX2 Microprocessor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified (Note 2)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to Intel486 DX2
t_1	CLK Period	30	125	ns	14.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	14.1	at 2V
t_3	CLK Low Time	11		ns	14.1	at 0.8V
t_4	CLK Fall Time		3	ns	14.1	2V to 0.8V
t_5	CLK Rise Time		3	ns	14.1	0.8V to 2V
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	14	ns	14.5	(Note 3)
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	14.6	(Note 1)
t_8	PCHK# Valid Delay	3	14	ns	14.4	(Note 3)
t_{8a}	BLAST#, PLOCK# Valid Delay	3	14	ns	14.5	(Note 3)
t_9	BLAST#, PLOCK# Float Delay		20	ns	14.6	(Note 1)
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	14	ns	14.5	(Note 3)
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	14.6	(Note 1)
t_{12}	EADS# Setup Time	5		ns	14.2	
t_{13}	EADS# Hold Time	3		ns	14.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	14.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	14.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	14.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	14.3	
t_{18}	HOLD, AHOLD, Setup Time	6		ns	14.2	
t_{18a}	BOFF# Setup Time	7		ns	14.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	14.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	14.2	(Note 4)
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	14.2	(Note 4)
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	14.2, 14.3	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	14.2, 14.3	

NOTES:
1. Not 100% tested. Guaranteed by design characterization.
2. All timing specifications assume C_L = 50 pF. Charts 14.4.3 provides the charts that may be used to determine the delay due to derating, depending on the lumped capacitive loading, that must be added to these specification values.
3. The minimum Intel486 DX2 output valid delays are hold times provided to external circuitry.
4. A reset pulse width of 15 CLK cycles is required for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Table 14.4.3. Intel486 DX2 Microprocessor A.C. Characteristics for Boundary Scan Test Signals

V_{CC} = 5V ±5%, T_{case} = 0°C to +85°C, C_L = 0 pF All Inputs and Outputs are TTL Level (Note 4)						
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Figure**	**Notes**
t_{24}	TCK Frequency		25	MHz		1x Clock
t_{25}	TCK Period	40		ns		(Note 2)
t_{26}	TCK High Time	10		ns		at 2.0V
t_{27}	TCK Low Time	10		ns		at 0.8V
t_{28}	TCK Rise Time		4	ns		(Note 1)
t_{29}	TCK Fall Time		4	ns		(Note 1)
t_{30}	TDI, TMS Setup Time	8		ns	14.7	(Note 3)
t_{31}	TDI, TMS Hold Time	7		ns	14.7	(Note 3)
t_{32}	TDO Valid Delay	3	25	ns	14.7	(Note 3)
t_{33}	TDO Float Delay		TBD			
t_{34}	All Outputs (Non-Test) Valid Delay	3	25	ns	14.7	(Note 3)
t_{35}	All Outputs (Non-Test) Float Delay		36	ns	14.7	(Note 3)
t_{36}	All Inputs (Non-Test) Setup Time	8		ns	14.7	(Note 3)
t_{37}	All Inputs (Non-Test) Hold Time	7		ns	14.7	(Note 3)

NOTES:
1. Rise/Fall times are measured between 0.8V and 2.0V. Rise/Fall times can be relaxed by 1 ns per 10 ns increase in TCK period.
2. TCK period ≥ CLK period.
3. Parameter measured from TCK.
4. Boundary Scan A.C. Specifications in the above table are target values. They have not been characterized. Therefore they are subject to change.

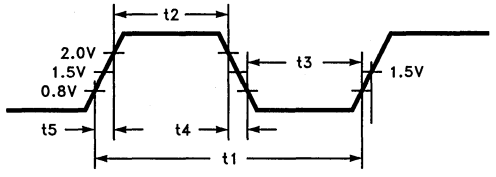


Figure 14.1. CLK Waveforms

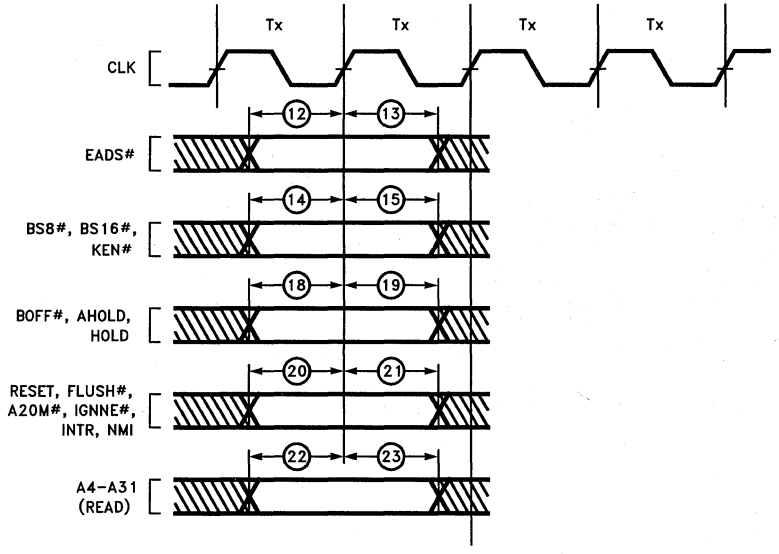


Figure 14.2. Input Setup and Hold Timing

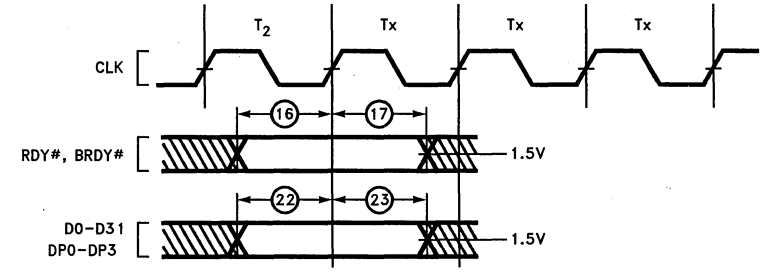


Figure 14.3. Input Setup and Hold Timing

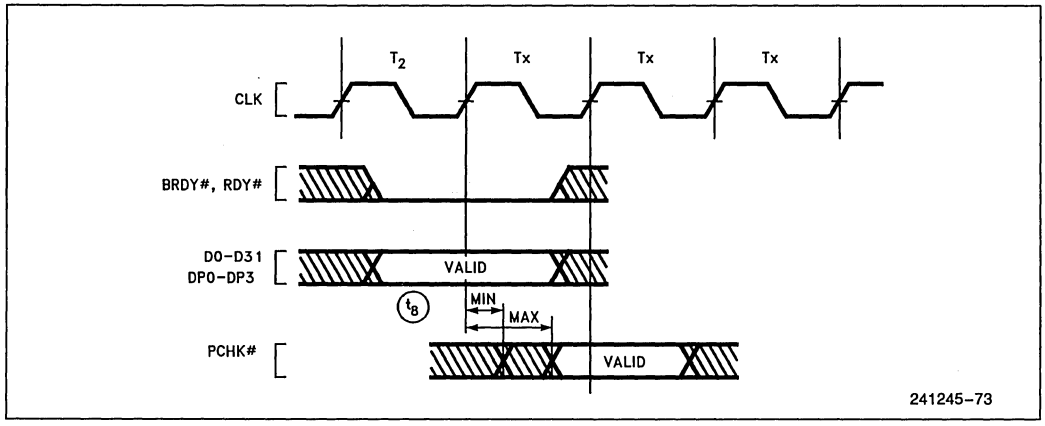


Figure 14.4. PCHK# Valid Delay Timing

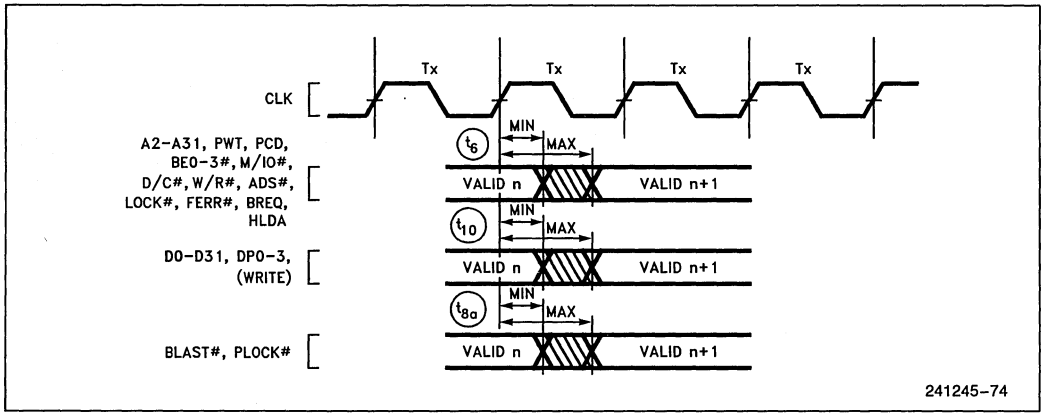


Figure 14.5. Output Valid Delay Timing

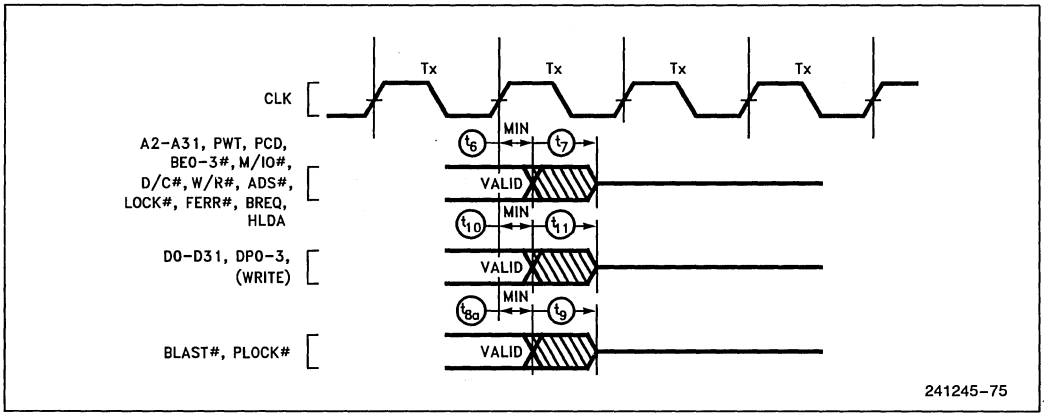


Figure 14.6. Maximum Float Delay Timing

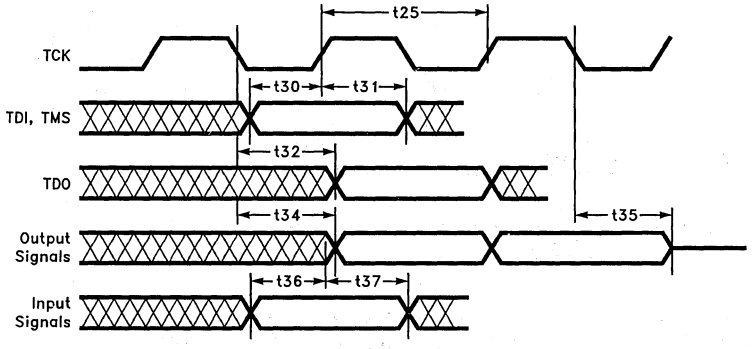
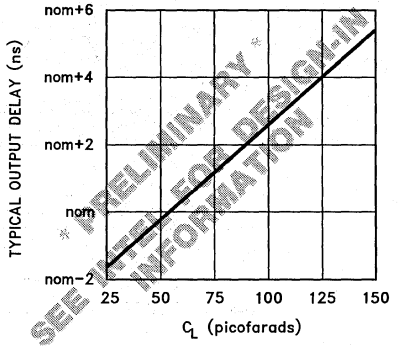


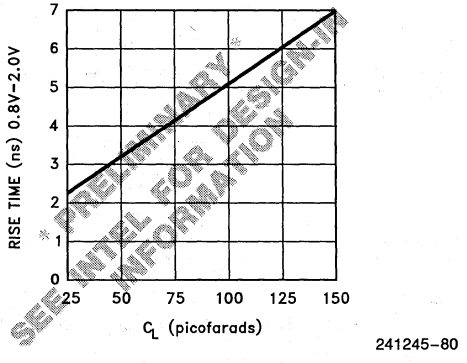
Figure 14.7. Test Signal Timing Diagram

14.4.1 TYPICAL OUTPUT VALID DELAY VERSUS LOAD CAPACITANCE UNDER WORST CASE CONDITIONS FOR THE 50 MHz AND 66 MHz INTEL486 DX2 CPU



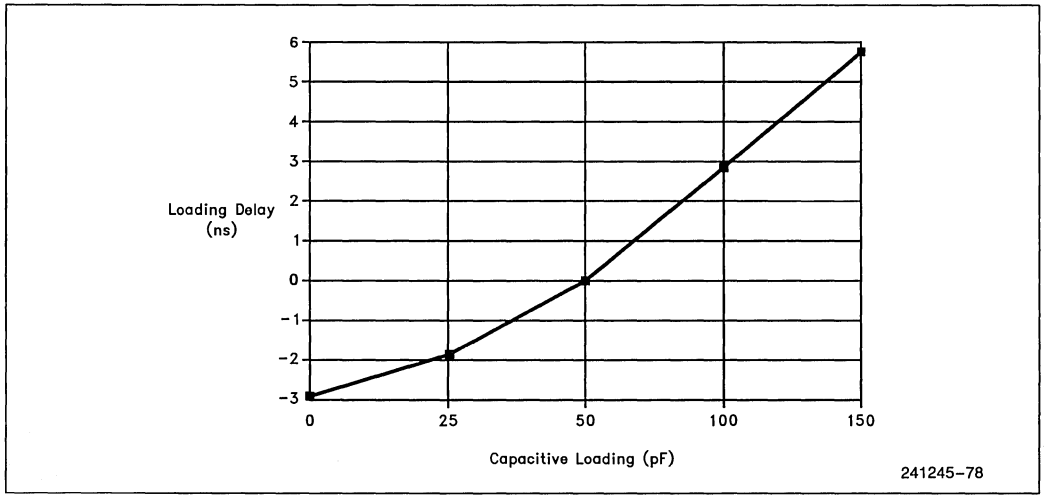
NOTE:
This graph will not be linear outside of the C_L range shown.
nom = nominal value given in A.C. Characteristics table.

14.4.2 TYPICAL OUTPUT RISE TIME VERSUS LOAD CAPACITANCE UNDER WORST-CASE CONDITIONS

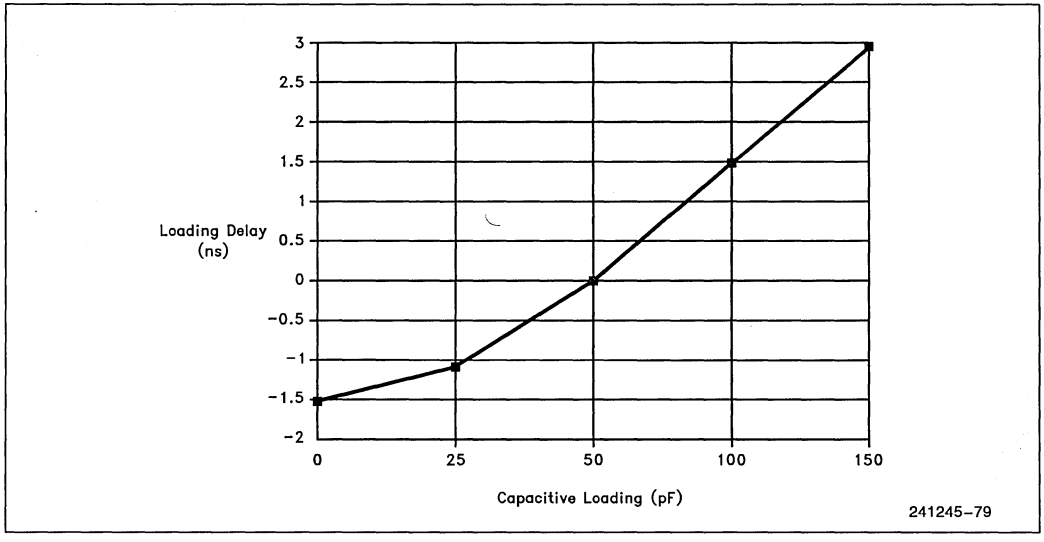


NOTE:
This graph will not be linear outside of the C_L range shown.

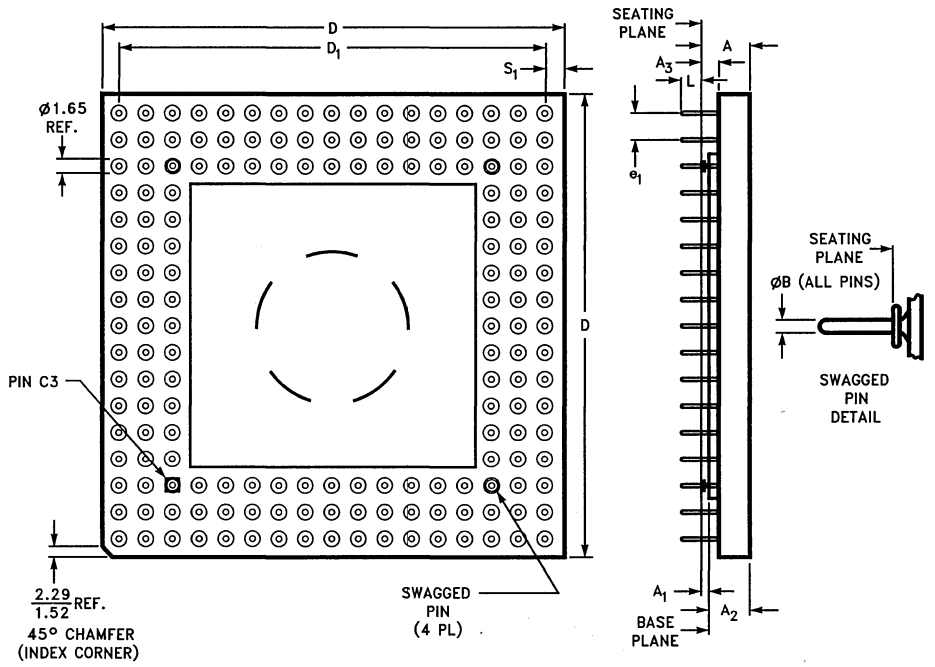
14.4.3.a TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A HIGH TO LOW TRANSITION ON THE INTEL486 DX2 CPU



14.4.3.b TYPICAL LOADING DELAY VERSUS CAPACITIVE LOADING UNDER WORST-CASE CONDITIONS FOR A LOW TO HIGH TRANSITION ON THE INTEL486 DX2 CPU



15.0 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	2.8	3.5	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	

Figure 15.1. 168 Lead Ceramic PGA Package Dimensions

Table 15.1 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.

15.1 Package Thermal Specifications

The Intel486 DX2 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 Intel486 DX2 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$T_J = T_C + P * \theta_{JC}$

$T_A = T_J - P * \theta_{JA}$

$T_A = T_C - (P * \theta_{CA})$

$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$

where T_J, T_A, T_C = Junction, Ambient and Case Temperature respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 13.2 for the 1.75 sq. in., 168-pin, ceramic PGA.

Table 13.3 shows the 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 14.

Table 15.2. Thermal Resistance (°C/W) θ_{JC} and θ_{CA} for the 50 MHz and 66 MHz Intel486 DX2 CPU

	θ_{JC}	θ_{CA} vs Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	2.5	10.5	7.0	4.5	3.5	3.0	2.5
Without Heat Sink	2.0	16	14.0	10.5	9.0	8.0	7.5

*0.350" high omnidirectional heat sink (Al alloy 6063, 40 mil fin width, 155 mil center-to-center fin spacing).

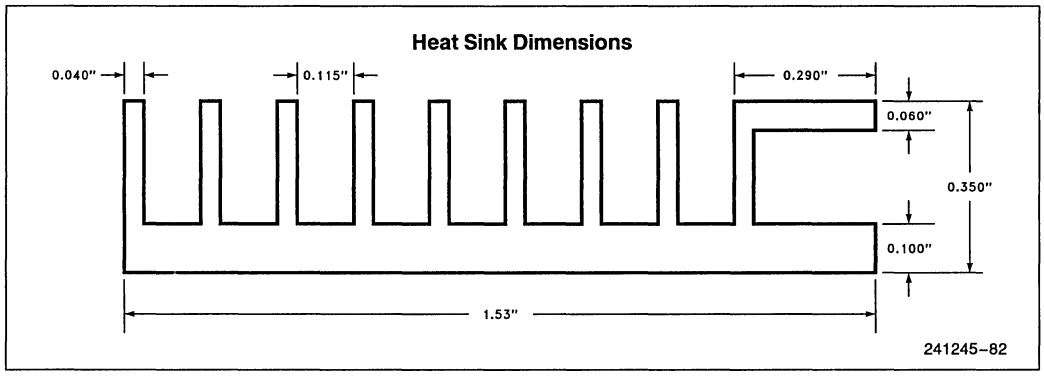


Table 15.3. Maximum T_A at Various Airflows In °C

		Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink	50 MHz	32.6	50.0	62.5	67.5	70.0	72.5
	66 MHz	18.9	40.9	56.7	62.9	66.1	69.3
T_A without Heat Sink	50 MHz	5.2	15.1	32.6	40.1	45.1	47.6
	66 MHz	−15.8	−3.2	18.9	28.3	34.6	37.8

16.0 SUGGESTED SOURCES FOR INTEL486 DX2 ACCESSORIES

Following are some suggested sources of accessories for the Intel486 DX2. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. McKenzie Technology
   44370 Old Palmspring Blvd.
   Fremont, CA 94538
   Tel: (415) 651-2700
2. E-CAM Technology, Inc.
   14455 North Hayden Rd.
   Suite 208
   Scottsdale, AZ 85260
   Tel: (602) 443-1949
3. Augat Inc. (for sockets with decaps)
   Interconnection Products Group
   33 Perry Ave.
   P.O. Box 779
   Attleboro, MA 02703
   Tel: (508) 222-2202

Heat Sinks/Fins

1. AAVID Engineering, Inc.
   One Kool Path
   P.O. Box 400
   Laconia, NH 03247
   Tel: (603) 528-3400

TTL Crystals/Oscillators

1. NFL Frequency Controls, Inc.
   357 Beloit Street
   Burlington, WI 53105
   Tel: (414) 763-3591
2. M-Tron
   P.O. Box 630
   Yankton, SD 57078
   Tel: (605) 665-9321

Debugging Tower

1. Emulation Technology
   2344 Walsh Ave., Building F
   Santa Clara, CA 95051
   Tel: (408) 982-0664

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

Intel Corp.
7225 N. Mona Lisa Rd.
Suite 215
Tucson 85741
Tel: (602) 544-0227
FAX: (602) 544-0232

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95678
Tel: (916) 782-8086
FAX: (916) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
600 S. Cherry St.
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (708) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-0960
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

MISSOURI

†Intel Corp.
3300 Rider Trail South
Suite 170
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

NEW JERSEY

Intel Corp.
Arbor Circle South
8 Campus Drive
Parsippany 07054
Tel: (201) 455-1868
FAX: (201) 644-0680

†Intel Corp.*
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

NEW YORK

Intel Corp.*
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Express Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

Intel Corp.
Seventeen State Street
Suite 1400
New York 10004
Tel: (212) 248-8086
FAX: (212) 248-0888

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

†Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

†Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393
FAX: (804) 330-3019

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

EUROPEAN SALES OFFICES

FINLAND

Intel Finland OY
Ruosilantie 2
00390 Helsinki
Tel: (358) 0 544 644
FAX: (358) 0 544 030

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex
Tel: (33) (1) 30 57 70 00
FAX: (33) (1) 30 64 6032

GERMANY

Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
FAX: (49) 089/904/3948

Intel GmbH
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (49) 06121/7605-0
FAX: (49) 06121 718615

Intel GmbH
Zettachring 10A
7000 Stuttgart 80
Tel: (49) 0711/7287-280
FAX: (49) 0711 7280137

ISRAEL

Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03-498080
FAX: (972) 03-491870

ITALY

Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano
Tel: (39) (02) 89200950
FAX: (39) (2) 3498464

NETHERLANDS

Intel Semiconductor B.V.
Postbus 84130
3009 CC Rotterdam
Tel: (31) 10 407 11 11
FAX: (31) 10 455 46 88

SPAIN

Intel Iberia S.A.
Zurbaran, 28
28010 Madrid
Tel: (34) 308 25 52
FAX: (34) 410 7570

SWEDEN

Intel Sweden A.B.
Dalvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
FAX: (46) 8 278085

SWITZERLAND

Intel Semiconductor A.G.
Zuerichstrasse
8185 Winkel-Rueti bei Zuerich
Tel: (41) 01/860 62 62
FAX: (41) 01/860 0201

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0793) 696000
FAX: (44) (0793) 641440

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics GmbH
Rotenmuehlgasse 26
A-1120 Wien
Tel: 43 222 81356460
FAX: 43 222 834276

BELGIUM

Inelco Belgium S.A.
Oorlogskruisenlaan 94
B-1120 Bruxelles
Tel: 32 2 244 2811
FAX: 32 2 216 3304

FRANCE

Almex
48, Rue de l'Aubepine
B.P. 102
92164 Antony Cedex
Tel: 33 1 4096 5400
FAX: 33 1 4666 6028

Jermyn
73-79 Rue des Solets
Silic 585
94663 Rungis Cedex
Tel: 33 1 4978 4878
FAX: 33 1 4978 0599

Metrologie
Tour d'Asnieres
4, Avenue Laurent Cely
92606 Asnieres Cedex
Tel: 33 1 4790 6240
FAX: 33 1 4790 5947

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle Vernet - BP 2
92310 Sevres
Tel: 33 1 4534 7535
FAX: 33 1 4507 2191

GERMANY

E2000 Vertriebs-AG
Stahlgruberring 12
8000 Muenchen 82
Tel: 49 89 420010
FAX: 49 89 42001209

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: 49 6431 5080
FAX: 49 6431 508289

Metrologie GmbH
Steinerstrasse 15
8000 Muenchen 70
Tel: 49 89 724470
FAX: 49 89 72447111

ITT Multikomponent Elektronik
Vertrieb
Bahnhofstr. 44
7141 Moeglingen
Tel: 49 7141 4879
FAX: 49 7141 487210

Proelectron Vertriebs GmbH
Max-Planck-Strasse 1-3
6072 Dreieich
Tel: 49 6103 3040
FAX: 49 6103 304344

GREECE

Pouliadis Associates Corp.
5 Koumbari Street
Kolonaki Square
106 74 Athens
Tel: 30 1 360 3741
FAX: 30 1 360 7501

IRELAND

Micro Marketing
Tony Hall
Eglinton Terrace
Dundrum, Dublin
Tel: 0001 989 400
FAX: 0001 989 8282

ISRAEL

Eastronics Ltd.
Rozanis 11
P.O.B. 39300
Tel Baruch, Tel-Aviv 61392
Tel: 972 3 475151
FAX: 972 3 475125

ITALY

Intesi Div. Della Deutsche
Divisione ITT Industries GmbH
P.I. 06550110156
Milanofiori palazzo E5
20094 Assago (Milano)
Tel: 39 2 824701
FAX: 39 2 8242631

Lasi Elettronica S.p.A.
P.I. 00839000155
Viale Fulvio Testi, N.280
20126 Milano
Tel: 39 2 66101370
FAX: 39 2 66101385

ITT Multicomponents
P.I. 06550110156
Palazzo E5 Milanofiori
20094 Assago (Milano)
Tel: 39 2 824701
FAX: 39 2 8242631

Silverstar Ltd. S.p.A.
P.I. 00751300153
Viale Fulvio Testi N.280
20126 Milano
Tel: 39 2 66125
FAX: 39 2 66101359

Telcom s.r.l. – Divisione MDS
Via Trombetta
Zona Marcomi – Strada Cassanese
Segrate – Milano
Tel: 39 2 48704100
FAX: 39 2 48705355

NETHERLANDS

Koning en Hartman B.V.
Energieweg 1
2627 AP Delft
The Netherlands
Tel: 31 15 609 906
FAX: 31 15 619 194

PORTUGAL

ATD Electronica LDA
Rua Dr. Faria de Vasconcelos, 3a
1900 Lisboa
Tel: 351 1 8472200
FAX: 351 1 8472197

SPAIN

ATD Electronica
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: 34 1 534 4000/09
FAX: 34 1 534 7663

Metrologia Iberica
Ctra De Fuencarral N.80
28100 Alcobendas (Madrid)
Tel: 34 1 6538611
FAX: 34 1 6517549

SCANDINAVIA

OY Fintronic AB
Heikkilantie 2a
SF-02100 Helsinki
Finland
Tel: 358 0 6926022
FAX: 358 0 6821251

ITT Multikomponent A/S
Naverland 29
DK-2600 Glostrup
Denmark
Tel: 010 45 42 451822
FAX: 010 45 42 457624

Nordisk Elektronik A/S
Postboks 122
Smedsvingen 4
N-1364 Hvalstad
Norway
Tel: 47 2 846210
FAX: 47 2 846545

Nordisk Electronik AB
Box 36
Torshamnsgatan 39
S-16493 Kista
Sweden
Tel: 46 8 7034630
FAX: 46 8 7039845

SWITZERLAND

Industrade A.G.
Hertistrasse 31
CH-8304 Wallisellen
Tel: 41 1 8328111
FAX: 41 1 8307550

UNITED KINGDOM

Accent Elect Comp Ltd.
Jubilee House
Jubilee Road
Letchworth
Hertsfordshire
SG6 1QH
Tel: 0462 480888
FAX: 0462 682467

Bytech Components Limited
12a Cedarwood
Chineham Business Park
Crockford Lane
Basingstoke
Hants RG12 1RW
Tel: 0256 707107
FAX: 0256 707162

Bytech Systems
Unit 3
The Western Centre
Western Road
Bracknell
Berks RG12 1RW
Tel: 0344 55333
FAX: 0344 867270

Conformix
Rapid House
Oxford Road
High Wycombe
Bucks
Herts HP11 2EE
Tel: 0494 474147
FAX: 0494 452144

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: 0732 450144
FAX: 0732 451251

MMD
3 Bennet Court
Bennet Road
Reading
Berkshire RG2 0QX
Tel: 0734 313232
FAX: 0734 313255

Rapid Silicon
3 Bennet Court
Bennet Road
Reading
Berks RG2 0QX
Tel: 0734 750697
FAX: 0734 312728

Metro Systems
Rapid House
Oxford Road
High Wycombe
Bucks HP11 2EE
Tel: 0494 474171
FAX: 0494 21860

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (1) (408) 988-0286
TLX: 387452

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.
Unit 13
Allamble Grove Business Park
25 Frenchs Forest Road East
Frenchs Forest, NSW, 2086
Tel: 61-2975-3300
FAX: 61-2975-3375

BRAZIL

Intel Semiconductores do Brazil LTDA
Avenida Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 11-37-557-ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (852) 844-4555
FAX: (852) 868-1989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 91-812-215773
TLX: 953-845-2646 INTEL IN
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
FAX: 0298-47-8450

Intel Japan K.K.*
Hachioji ON Bldg.
4-7-14 Myojin-machi
Hachioji-shi, Tokyo 192
Tel: 0426-48-8770
FAX: 0426-48-8775

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222
Tel: 045-474-7661
FAX: 045-471-4394

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-3201-3621
FAX: 03-3201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #08-03/06
United Square
Singapore 1130
Tel: (65) 250-7811
FAX: (65) 250-9256

TAIWAN

Intel Technology Far East Ltd.
Taiwan Branch Office
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-5144202
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Dafsys S.R.L.
Chacabuco, 90-6 Piso
1069-Buenos Aires
Tel: 54-1-34-7726
FAX: 54-1-34-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Componentes
Rua Geraldo Flausina Gomes, 78
7 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Room 728 Trade Square
681 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: (852) 360-8999
TWX: 32032 NVTNL HX
FAX: (852) 725-3695

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

CTC Components Systems Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-3439-1600
FAX: 03-3439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-3546-5011
FAX: 03-3546-5044

KOREA

J-Tek Corporation
Dong Sung Bldg. 9/F
158-24, Samsung-Dong, Kangnam-Ku
Seoul 135-090
Tel: (822) 557-8039
FAX: (822) 557-8304

Samsung Electronics
Samsung Main Bldg.
150 Taepyung-Ro-2KA, Chung-Ku
Seoul 100-102
C.P.O. Box 8780
Tel: (822) 751-3680
TWX: KORSST K 27970
FAX: (822) 753-9065

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PSI S.A. de C.V.
Fco. Villa esq. Ajusco s/n
Cuernavaca – Morelos
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SAUDI ARABIA

AAE Systems, Inc.
642 N. Pastoria Ave.
Sunnyvale, CA 94086
U.S.A.
Tel: (408) 732-1710
FAX: (408) 732-3095
TLX: 494-3405 AAE SYS

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road
#03-01 Singapore 1336
Tel: (65) 283-0888
TWX: RS 56541 ERS
FAX: (65) 289-5327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus St. (off Watermeyet St.)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
12th Floor, Section 3
285 Nanking East Road
Taipei, R.O.C.
Tel: (886) 2-7198419
FAX: (886) 2-7197916

Acer Sertek Inc.
15th Floor, Section 2
Chien Kuo North Rd.
Taipei 18479 R.O.C.
Tel: 886-2-501-0055
TWX: 23756 SERTEK
FAX: (886) 2-5012521

*Field Application Location

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

GERMANY
Intel 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: 241245-001
Printed in U.S.A./10K / 0392 / EI
Microprocessors