AMD◢

Am486™DX/DX2 Microprocessor

Hardware Reference Manual

E86 FAMILY™

Am486™DX/DX2
Microprocessor Hardware Reference
Manual

Rev. 1, 1993

A D V A N C E D   M I C R O   D E V I C E S

Trademarks

AMD and Am386 are registered trademarks of Advanced Micro Devices, Inc.

Am486 is a trademark of Advanced Micro Devices, Inc.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Am486DX/DX2 MICROPROCESSOR FEATURES

1.1 GENERAL DESCRIPTION

The Am486DX and Am486DX2 CPUs offer the highest performance for DOS, OS/2, Windows, and UNIX applications. They are 100% binary compatible with the 386 architecture. One million plus transistors integrate cache memory, floating-point hardware, and memory management on-chip while retaining binary compatibility with previous members of the x86 architectural family. Frequently used instructions execute in one cycle, resulting in RISC performance levels. An 8-Kbyte unified code and data cache combine with a burst bus to ensure high system throughput.

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

The Am486DX/DX2 microprocessors feature boundary scan test signals. This provides additional testability features compatible with the IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Standard 1149.1 JTAG).

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.

This manual, when combined with the *Am486DX Microprocessor Data Sheet* (order # 17852), and the *Am486DX2 Microprocessor Data Sheet* (order # 17914), provides complete design documentation.

1.2 DISTINCTIVE CHARACTERISTICS

- Binary Compatible with Large Software Base
  - MS-DOS, OS/2, Windows
  - UNIX, Windows NT
- High Integration 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
  - Extensive third-party software support
- IEEE 1149.1 Boundary Scan Compatibility on all versions
- High-performance Design
  - Frequent instructions execute in one clock
  - 0.7-micron CMOS process technology

— Dynamic bus sizing for 8-, 16-, and 32-bit buses

■ Complete 32-bit Architecture

— All registers

— 8-, 16-, and 32-bit data types

■ Multiprocessor Support

— Multiprocessor instructions

— Cache consistency protocols

— Support for second-level cache

Figure 1-1   Am486DX/DX2 CPU Pipelined 32-Bit Microarchitecture Block Diagram
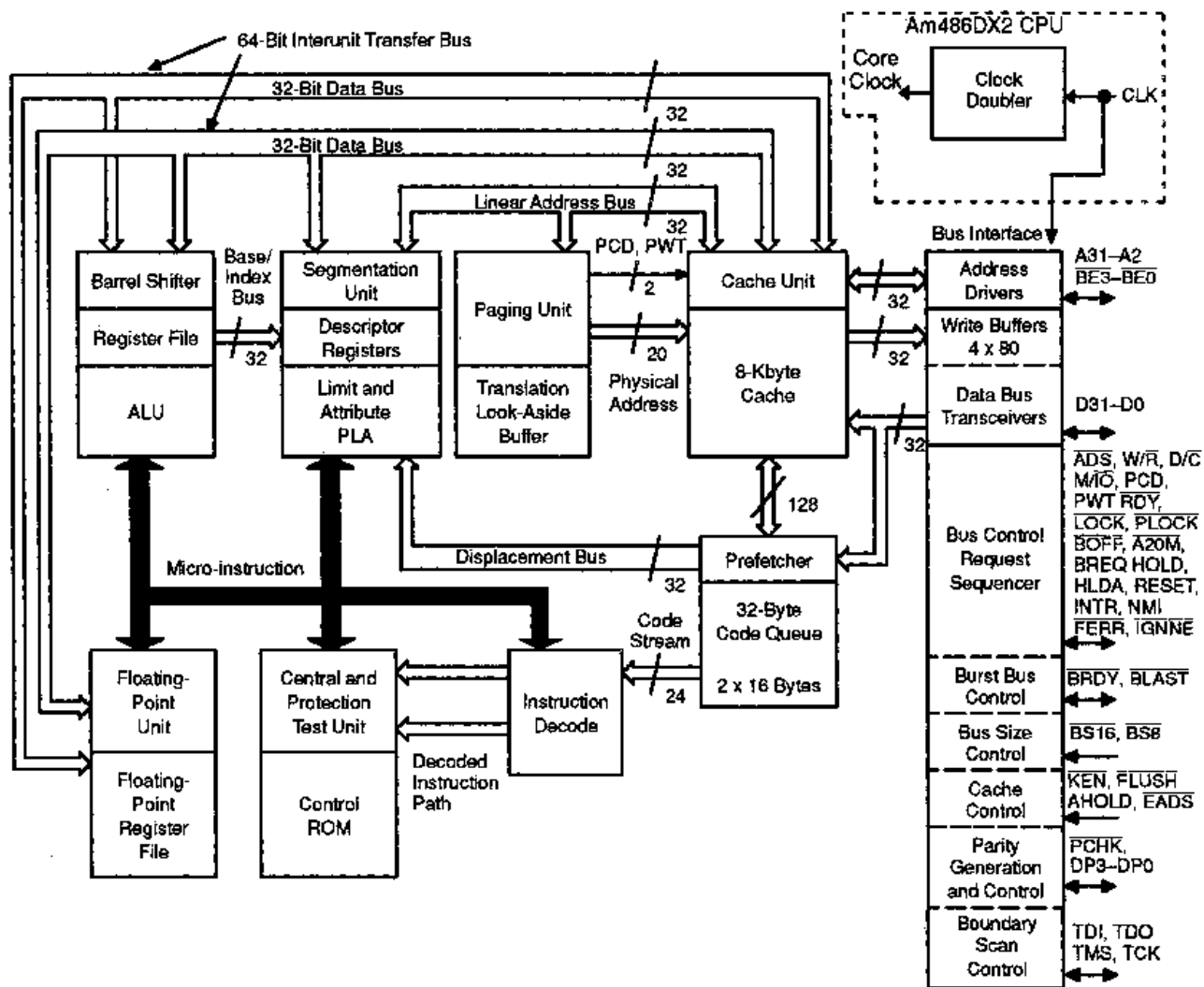
17852A-001

Figure 1-2   Logic Symbol

Clock → CLK

Address Mask → A20M

Address Bus { A31–A4 (28), A3–A2 (2), BE3–BE0 (4) }

Bus Cycle Control { BS8, BS16, ADS, READY }

Bus Cycle Definition { M/IO, D/C, W/R, LOCK, PLOCK }

Interrupts { INTR, NMI, RESET }

Am486DX/DX2 CPU

D31–D0 (32) Data Bus

DP3–DP0 (4), PCHK } Data Parity

BRDY, BLAST } Burst Control

PWT, PCD } Page Cacheability

KEN, FLUSH, AHOLD, EADS } Cache Control/Invalidation

HOLD BOFF
BREQ HLDA   IGNNE FERR   TCK TDI
                                          TMS TDO

Bus Arbitration   Numeric Error Reporting   IEEE Test Port Access

17852B–114

1.3   PIN DESCRIPTIONS

The following paragraphs define the pins (signals) of the Am486DX/DX2 microprocessor.

A31–A4/A3–2
Address Lines (Inputs/Outputs)/(Outputs)

Pins A31–A2, together with the byte enable pins $\overline{BE3}$–$\overline{BE0}$, constitute the address bus and 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.

$\overline{A20M}$
Address Bit 20 Mask (Active Low; Input)

When the address mask is asserted, the Am486DX/DX2 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. $\overline{A20M}$ emulates the address wraparound at 1 Mbyte, which occurs on the 8086. $\overline{A20M}$ 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, $\overline{A20M}$ should be sampled High at the falling edge of RESET.

$\overline{ADS}$
Address Status (Active Low; Output)

This pin is used to indicate that a valid bus cycle definition and address are available on the cycle definition lines and address bus. $\overline{ADS}$ is driven active in the same clock as the addresses are driven. $\overline{ADS}$ is active Low and is not driven during bus hold.

AHOLD
Address Hold (Active High; Input)

An address hold request allows another bus master access to the Am486DX/DX2 microprocessor's address bus for a cache invalidation cycle. The Am486DX/DX2 microprocessor stops driving its address bus in the clock following AHOLD going active. Only the address bus is floated during address hold; the remainder of the bus remains active. AHOLD is active High and is provided with a small internal pull-down resistor. For proper operation, AHOLD must meet setup and hold times t_{18} and t_{19}.

$\overline{BE3}$–$\overline{BE0}$
Byte Enables (Active Low; Outputs)

The address bus byte-enable pins 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. $\overline{BE3}$ applies to D31–D24, $\overline{BE2}$ applies to D23–D16, $\overline{BE1}$ applies to D15–D8, and $\overline{BE0}$ applies to D7–D0. $\overline{BE3}$–$\overline{BE0}$ are active Low and are not driven during bus hold.

$\overline{BS8}$/$\overline{BS16}$
Bus Size 8 (Active Low; Input)/Bus Size 16 (Active Low; Input)

The bus sizing pins cause the Am486DX/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 $\overline{RDY}$ is used by the Am486DX/DX2 microprocessor to determine the bus size. These signals are active Low and are provided with internal pull-up resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.

$\overline{BLAST}$
Burst Last (Active Low; Output)

This pin indicates that the next time $\overline{BRDY}$ is returned then the burst bus cycle is complete. $\overline{BLAST}$ is active for both burst and non-burst bus cycles. $\overline{BLAST}$ is active Low and is not driven during bus hold.

$\overline{BOFF}$
Backoff (Active Low; Input)

This input pin forces the Am486DX/DX2 microprocessor to float its bus in the next clock. The microprocessor floats all pins normally floated during bus hold, but HLDA is not asserted in response to $\overline{BOFF}$. $\overline{BOFF}$ has higher priority than $\overline{RDY}$ or $\overline{BRDY}$; if both are returned in the same clock, $\overline{BOFF}$ takes effect. The microprocessor remains in bus hold until $\overline{BOFF}$ is negated. If a bus cycle is in progress when $\overline{BOFF}$ is asserted, the cycle is

restarted. $\overline{\text{BOFF}}$ is active Low and must meet setup and hold times t_{18} and t_{19} for proper operation.

$\overline{\text{BRDY}}$
Burst Ready Input (Active Low; input)

The $\overline{\text{BRDY}}$ signal performs the same cycle during a burst cycle that $\overline{\text{RDY}}$ performs during a non-burst cycle. $\overline{\text{BRDY}}$ 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 write. $\overline{\text{BRDY}}$ is ignored when the bus is idle and at the end of the first clock in a bus cycle.

$\overline{\text{BRDY}}$ is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus is strobed into the microprocessor when $\overline{\text{BRDY}}$ is sampled active. If $\overline{\text{RDY}}$ is returned simultaneously with $\overline{\text{BRDY}}$, $\overline{\text{BRDY}}$ is ignored and the burst cycle is prematurely aborted.

$\overline{\text{BRDY}}$ is active Low and is provided with a small pull-up resistor. $\overline{\text{BRDY}}$ must satisfy the setup and hold times t_{16} and t_{17}.

BREQ
Internal Cycle Pending (Output)

BREQ indicates that the Am486DX/DX2 microprocessor has internally generated a bus request. BREQ is generated whether or not the Am486DX/DX2 microprocessor is driving the bus. BREQ is active High and is never floated, except during three-state test mode (see $\overline{\text{FLUSH}}$).

CLK
Clock (Input)

The CLK input provides the fundamental timing and the internal operating frequency for the Am486DX/DX2 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

D31–D0
Data Lines (Inputs/Outputs)

Lines D7–D0 define the least significant byte of the bus while lines D31–D24 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.

D/$\overline{\text{C}}$
Data/Control (Output)

This bus cycle definition pin distinguishes data cycles, either memory or I/O, from control cycles. These control cycles are: interrupt acknowledge, halt, and instruction fetching.

DP3–DP0
Data Parity (Inputs/Outputs)

Data parity is generated on all write data cycles with the same timing as the data driven by the Am486DX/DX2 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information; this ensures that the correct parity check status is indicated by the Am486DX/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}. DP3–DP0 should be connected to V_{CC} through a pull-up resistor in systems not using parity. DP3–DP0 are active High and are driven during the second and subsequent clocks of write cycles.

$\overline{\text{EADS}}$
Valid External Address (Active Low; Input)

This address indicates a valid external address has been driven onto the Am486DX/DX2 microprocessor address pins. This address is used to perform an internal cache invalidation cycle. $\overline{\text{EADS}}$ is active Low and is provided with an internal pull-up resistor. $\overline{\text{EADS}}$ must satisfy setup and hold times t_{12} and t_{13} for proper operation.

$\overline{\text{FERR}}$
Floating-Point Error (Active Low; Output)

Driven active when a floating-point error occurs, $\overline{\text{FERR}}$ is similar to the $\overline{\text{ERROR}}$ pin on a 387 math coprocessor. $\overline{\text{FERR}}$ is included for compatibility with systems using DOS-type floating-point error reporting. $\overline{\text{FERR}}$ is active Low and is not floated during bus hold, except during three-state test mode (see $\overline{\text{FLUSH}}$).

$\overline{\text{FLUSH}}$
Cache Flush (Active Low; Input)

$\overline{\text{FLUSH}}$ forces the Am486DX/DX2 microprocessor to flush its entire internal cache. This input pin is active Low and need only be asserted for one clock. $\overline{\text{FLUSH}}$ is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. $\overline{\text{FLUSH}}$ being sampled Low in the clock before the falling edge of RESET causes the Am486DX/DX2 microprocessor to enter the three-state test mode.

HLDA
Hold Acknowledge (Output)

The HLDA signal is activated in response to a hold request presented on the HOLD pin. HLDA indicates that the Am486DX/DX2 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock in which the Am486DX/DX2 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active High and remains driven during bus hold. HLDA is never floated except during three-state test mode (see $\overline{\text{FLUSH}}$).

HOLD
Bus Hold Request (Input)

HOLD gives another bus master complete control of the Am486DX/DX2 microprocessor bus. In response to HOLD going active, the Am486DX/DX2 microprocessor floats most of its output and input/output pins. HLDA is asserted after completing the current bus cycle, burst cycle, or sequence of locked cycles. The Am486DX/DX2 microprocessor remains in this state until HOLD is deasserted. 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 operation.

$\overline{\text{IGNNE}}$
Ignore Numeric Error (Active Low; Input)

When this pin is asserted, the Am486DX microprocessor ignores a numeric error and continues executing non-control floating-point instructions. When $\overline{\text{IGNNE}}$ is deasserted, the Am486DX/DX2 microprocessor freezes on a non-control floating-point instruction if a previous floating-point instruction caused an error. $\overline{\text{IGNNE}}$ has no effect when the NE bit in Control Register 0 is set. $\overline{\text{IGNNE}}$ is active Low and is provided with a small internal pull-up resistor. $\overline{\text{IGNNE}}$ is asynchronous but setup and hold times t_{20} and t_{21} must be met to ensure recognition on any specific clock.

INTR
Maskable Interrupt (Input)

When asserted, this signal indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing is initiated. The

Am486DX/DX2 microprocessor generates 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 ensure that the interrupt is recognized. INTR is active High and is not provided with an internal pull-down resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

$\overline{\text{KEN}}$
Cache Enable (Active Low; Input)

$\overline{\text{KEN}}$ is used to determine whether the current cycle is cacheable. When the Am486DX/DX2 microprocessor generates a cacheable cycle and $\overline{\text{KEN}}$ is active one clock before $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ during the first transfer of the cycle, the cycle becomes a cache line fill cycle. Returning $\overline{\text{KEN}}$ active one clock before $\overline{\text{RDY}}$ during the last read in the cache line fill causes the line to be placed in the on-chip cache. $\overline{\text{KEN}}$ is active Low and is provided with a small internal pull-up resistor. $\overline{\text{KEN}}$ must satisfy setup and hold times t_{14} and t_{15} for proper operation.

$\overline{\text{LOCK}}$
Bus Lock (Active Low; Output)

This pin indicates that the current bus cycle is locked. The Am486DX/DX2 microprocessor does not allow a bus hold when $\overline{\text{LOCK}}$ is asserted (but address holds are allowed). $\overline{\text{LOCK}}$ 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 $\overline{\text{RDY}}$ is returned. $\overline{\text{LOCK}}$ is active Low and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if $\overline{\text{KEN}}$ is returned active.

M/$\overline{\text{IO}}$
Memory (Input/Output) (Output)

This bus cycle definition pin distinguishes memory cycles from input/output cycles.

NMI
Non-maskable Interrupt (Input)

The NMI 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 pull-down resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.

PCD/PWT
Page Cache Disable/Page Write-Through (Outputs)

These 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 unpaged cycles, 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/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, and W/$\overline{\text{R}}$). 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.

$\overline{\text{PCHK}}$
Parity Status (Active Low; Output)

Parity status is driven on the $\overline{\text{PCHK}}$ pin the clock after $\overline{\text{RDY}}$ for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by $\overline{\text{PCHK}}$ being Low. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. $\overline{\text{PCHK}}$ is valid only in the clock immediately after read data is returned to the microprocessor; at all other times $\overline{\text{PCHK}}$ is inactive High. $\overline{\text{PCHK}}$ is never floated, except during three-state test mode (see $\overline{\text{FLUSH}}$).

PLOCK
Pseudo-Lock (Active Low; Output)

PLOCK 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), and cache line fills (128 bits). The Am486DX/DX2 microprocessor drives 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 are asserted. PLOCK is a function of the BS8, BS16, and KEN inputs. PLOCK should be sampled on if the clock RDY is returned. PLOCK is active Low and is not driven during bus hold.

RESET
Reset (Active High; Input)

RESET forces the Am486DX/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 ensure 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 a specific clock.

RDY
Non-Burst Ready (Active Low; Input)

This pin 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 Am486DX/DX2 microprocessor in response to a write. RDY is ignored when the bus is idle and at the end of the bus cycle's first clock.

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 pull-up resistor. RDY must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

TCK
Test Clock (Input)

Test Clock is an input to the Am486 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 component on the falling edge of TCK on TDO.

TDI
Test Data Input (Input)

TDI 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 (Tap Access Port) controller states. During all other tap controller states, TDI is a "don't care."

TDO
Test Data Output (Output)

TDO 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
Test Mode Select (Input)

TMS is decoded by the JTAG TAP 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.

UP
Upgrade Present (Input) (PQFP package only)

The Upgrade Present pin forces the Am486DX2 CPU to three-state all 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 expect during this edge. The CPU is also forced to three-state all of its outputs immediately in response to this signal. The UP signal must remain asserted in order to keep the pins three-stated. UP is active Low and is provided with an internal pull-up resistor.

Note: The UP pin is for the Am486DX2 CPU.

W/R
Write/Read (Output)

A bus cycle definition pin, W/R distinguishes write cycles from read cycles.

Table 1-1   Output Pins

Name	Active Level	Floated At
BREQ	High	
HLDA	High	
BE3–BE0	Low	Bus Hold
PCD/PWT	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	
A3–A2	High	Bus, Address Hold

Table 1-2   Input Pins

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

Table 1-3   Input/Output Pins

Name	Active Level	Floated At
D31–D0	High	Bus Hold
DP3–DP0	High	Bus Hold
A31–A4	High	Bus, Address Hold

Table 1-4   Bus Cycle Definition

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

Table 1-5   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 Am486DX/DX2 CPU ARCHITECTURAL OVERVIEW

2.1 INTRODUCTION

The Am486DX/DX2 microprocessor is a 32-bit architecture with on-chip memory management, floating-point, and cache memory units.

The Am486DX/DX2 microprocessor contains all the features of the 386 microprocessor with enhancements to increase performance. The instruction set includes the complete 386 microprocessor instruction set along with extensions to serve new applications. The on-chip Memory Management Unit (MMU) is completely compatible with the 386 MMU. The Am486DX/DX2 microprocessor brings the functions of a math coprocessor on-chip. All software written for the 386 microprocessor, 387 math coprocessor, and previous members of the x86/x87 architectural family can run on the Am486DX/DX2 microprocessor without any modifications.

Several enhancements have been added to the Am486DX/DX2 microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip, thereby reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features combined lead to performance greater than twice that of a 386 microprocessor.

The Am486 microprocessor 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 Am486DX/DX2 microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to 4 Gbytes (2^{32} bytes) in size. A segment can have attributes associated with it. These attributes include its location, size, type (i.e., stack, code, or data), and protection characteristics. Each task on an Am486DX/DX2 microprocessor can have a maximum of 16,381 segments, each up to 4 Gbytes in size. Thus, each task has a maximum of 64 Tbyte (terabytes) 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 high integrity system designs.

The Am486DX/DX2 microprocessor has two modes of operation: Real Address Mode (Real Mode) and Virtual Address Mode (Protected Mode). In Real Mode, the Am486DX/DX2 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 (FPU) operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. The FPU executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions) and conforms to the ANSI/IEEE standard 754-1985 for floating-point arithmetic.

The on-chip cache is 8 Kbytes. It is four-way set associative and follows a write-through policy. The on-chip cache includes features that 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 Am486DX/DX2 microprocessor has features that facilitate high-performance hardware designs. The 1X clock on the Am486DX CPU eases high frequency board level designs. The 2X clock doubler on the Am486DX/DX2 CPU 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 assesses. The burst bus feature enables fast cache fills.

2.2   REGISTER SET

The Am486DX/DX2 microprocessor register set includes all the registers contained in the 386 microprocessor and the 387 math coprocessor. The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers
- Floating-Point Registers
- Data Registers
- Tag Word
- Status Word
- Instruction and Data Pointers
- Control Word
- Debug and Test Registers

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

2.2.1   Base Architecture Registers

Figure 2-1 shows the Am486DX/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 Am486DX/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.

2.2.1.1 General Purpose Registers

The eight 32-bit general purpose registers (see Figure 2-1) 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.

Figure 2-1   Base Architecture Registers

General Purpose Registers

31	24	23	16	15	8	7	0	
				AH	AX	AL		EAX
				BH	BX	BL		EBX
				CH	CX	CL		ECX
				DH	DX	DL		EDX
					SI			ESI
					DI			EDI
					BP			EBP
					SP			ESP

Segment Registers

15   0

CS   Code Segment
SS   Stack Segment
DS ⎫
ES ⎬ Data Segments
FS ⎪
GS ⎭

Instruction Pointer

31   16   15   0

IP   EIP

Flags Register

31   16   15   0

FLAGS   EFLAGS
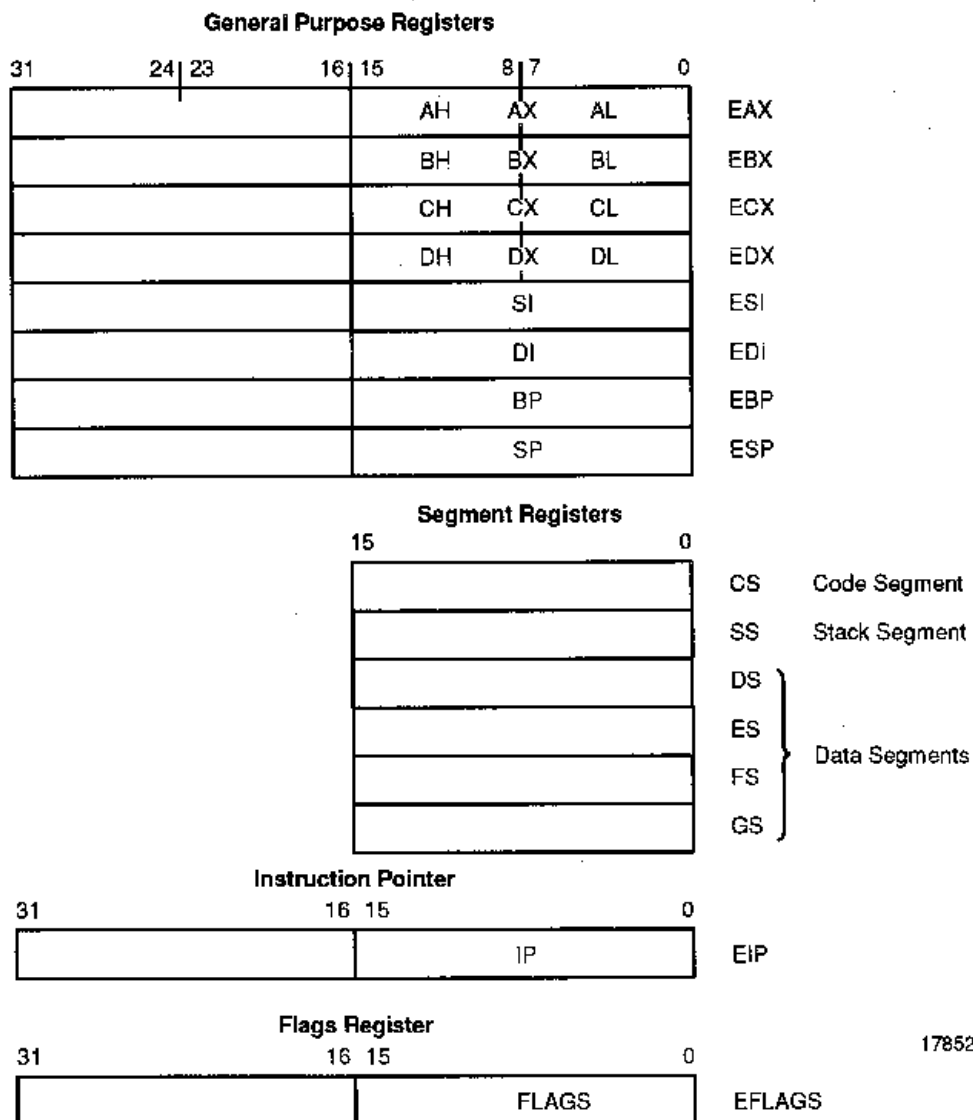
17852A–004

Finally, 8-bit operations can individually access the lower byte (bits 7–0) 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.

2.2.1.2   Instruction Pointer

The instruction pointer (see 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 15–0) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.2.1.3   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 the Am486DX/DX2 microprocessor's status. The lower 16 bits (bits 15–0) 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 1 is stored in bit 1 and 0s in bits 3, 5, 15, and 19–31.

Figure 2-2   Flags Register
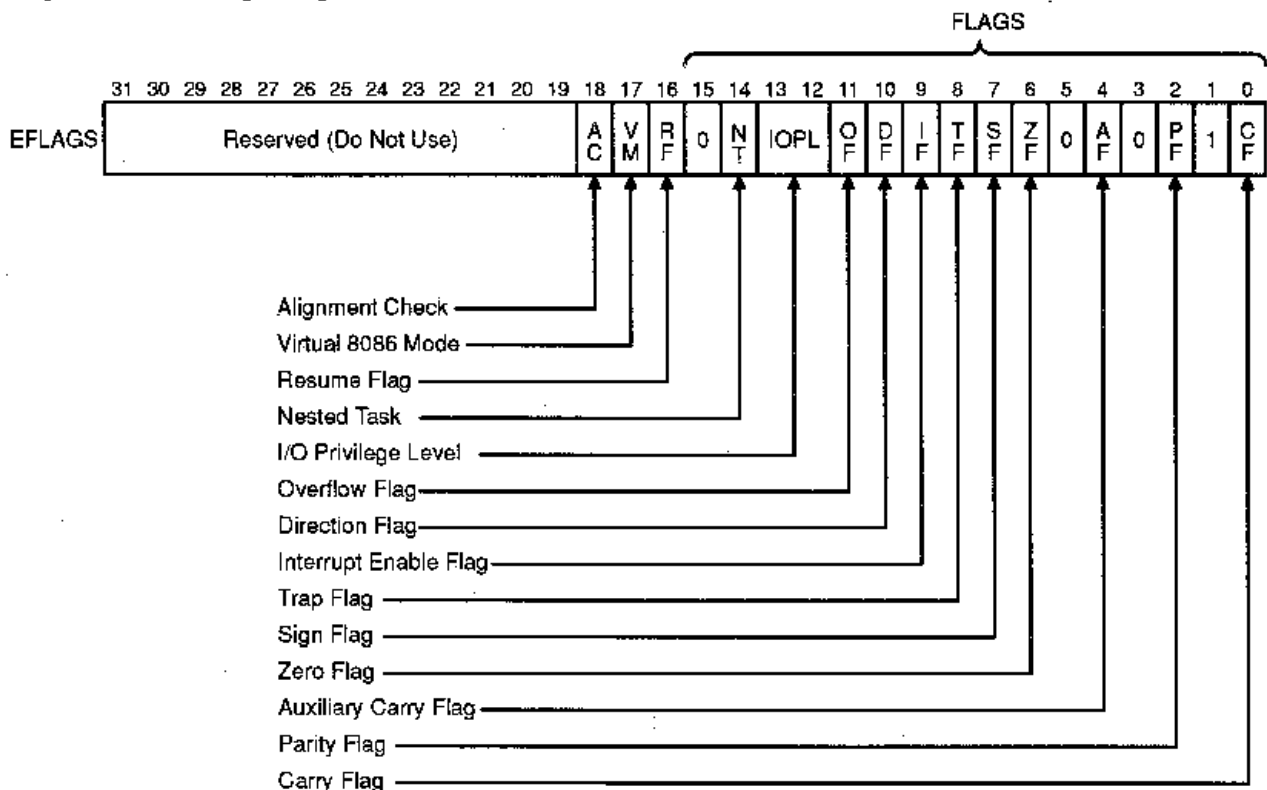
Note:

Bit positions shown as 0 or 1 are Reserved (Do Not Use). Always set them to the value previously read.

17852A–005

The EFLAGS register in the Am486DX/DX2 microprocessor contains a new bit not previously defined in the Am386® CPU. 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 misaligned 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 Am486DX/DX2 microprocessor data types.

Implementation Note: Several Am486DX/DX2 microprocessor instructions generate misaligned references, even if their memory address is aligned. For example, on the Am486DX/DX2 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 Am486DX/DX2 microprocessor generates misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating-point save and restore state) generate misaligned references for one-half of the register save/restore cycles. The Am486DX/DX2 microprocessor does 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 Am486DX/DX2 microprocessor is in Protected Mode, the Am486DX/DX2 microprocessor switches to Virtual 8086 operation. Virtual 8086 Mode handles segment loads as the 8086 does, but generates 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 contains 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, any debug fault is ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signaled) except the IRET instruction and the POPF instruction, (also JMP, CALL, and INT instructions that cause 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 EFLAGS 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.

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

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. NT 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. POPF or an IRET instruction affects the setting of this bit according to the image popped, at any privilege level.

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

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 Bit-map. It also indicates the maximum CPL value that allows alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAGS 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-, and 32-bit operations, OF is set according to overflow at bit 7, 15, and 31, respectively.

DF   (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers post decrement or post increment during the string instructions. Post increment occurs if DF is reset. Post decrement occurs if DF is set.

IF   (INTR Enable Flag, bit 9)

When set, the IF flag allows recognition of external interrupts signaled on the INTR pin. When IF is reset, external interrupts signaled on the INTR are not recognized. IOPL indicates the maximum CPL value that allows 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 Am486DX/DX2 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 DR3–DR0.

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-, and 32-bit operations, SF reflects the state of bit 7, 15, and 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 contain an even number of "1s" (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."

2.2.1.4   Segment Registers

Six 16-bit segment registers hold segment selector values that identify the currently addressable memory segments. In Protected Mode, each segment can 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.

2.2.1.5   Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer-visible, yet understanding their content is very useful. A programmer-invisible descriptor cache register is associated with each programmer visible segment register (see Figure 2-3). Each

descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and other necessary segment attributes.

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.

2.2.2 System Level Registers

The system level registers (see 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 AMD 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).

Figure 2-3   Am486 Microprocessor Segment Registers and Associated Descriptor Cache Registers
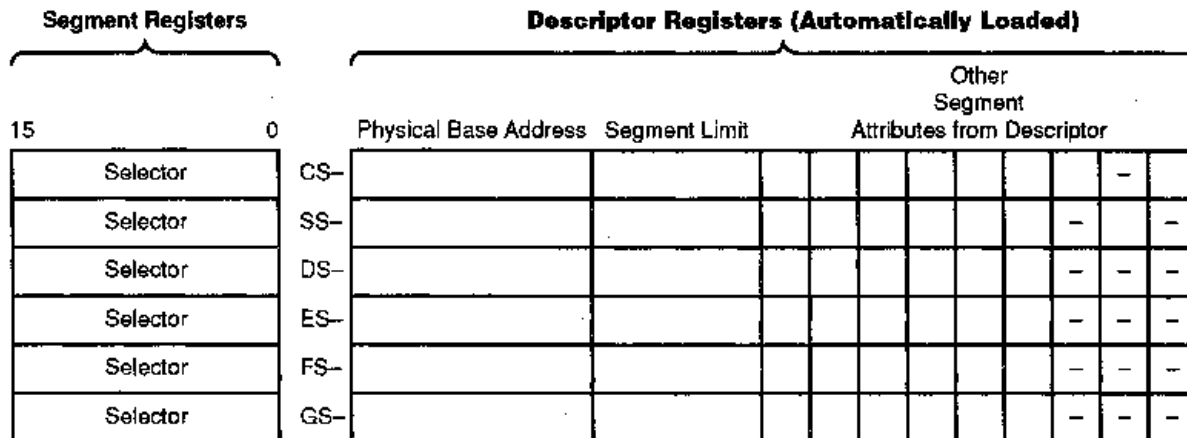
17852A-006

Figure 2-4   System Level Registers

```
            31        24 23        16 15        8 7         0
           +--------------------------------------------------+
           |                                                  | CR0
           +--------------------------------------------------+
           |        Page Fault Linear Address Register        | CR2
           +--------------------------------------------------+
           |        Page Directory Base Register    |         | CR3
           +--------------------------------------------------+
```

System Address Registers

```
       47            32-Bit Linear Base Address      16 15   Limit      0
      +---------------------------------------------+----------------------+
GDTR  |                                             |                      |
      +---------------------------------------------+----------------------+
IDTR  |                                             |                      |
      +---------------------------------------------+----------------------+
```

Descriptor Registers (Automatically Loaded)

System Segment Registers

```
       15                  0    32-Bit Linear Base Address   20-Bit Segment Limit   Attributes
      +--------------------+   +-------------------------+--------------------+------+
TR    |     Selector       |   |                         |                    |      |
      +--------------------+   +-------------------------+--------------------+------+
LDTR  |     Selector       |   |                         |                    |      |
      +--------------------+   +-------------------------+--------------------+------+
```
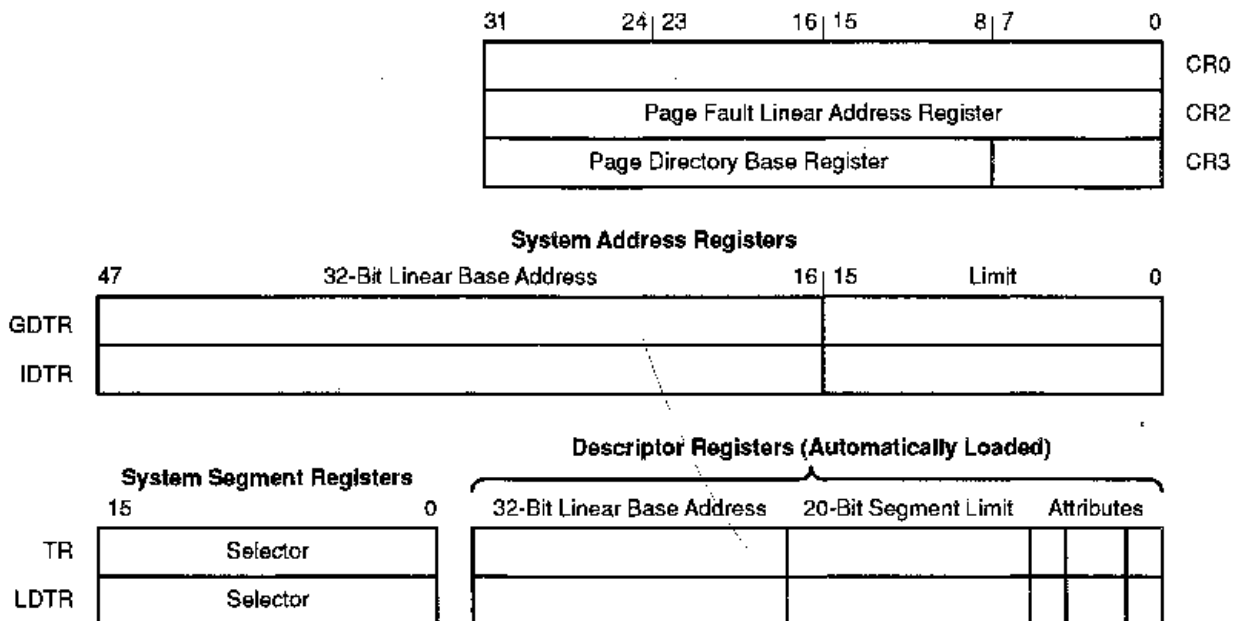
17852A–007

Figure 2-5   Control Register 0

```
 31               24 23      18   16 15          8 7   5   3 2 1 0
+---+---+---+-------------+---+---+---+----------------+---+--+--+--+--+--+
| P | C | N |             | A | W |   |                | N |1 |T |E |M |P |  CR0
| G | D | W |             | M | P |   |                | E |  |S |M |P |E |
+---+---+---+-------------+---+---+---+----------------+---+--+--+--+--+--+
                                    _____/
                                                    MSW
```
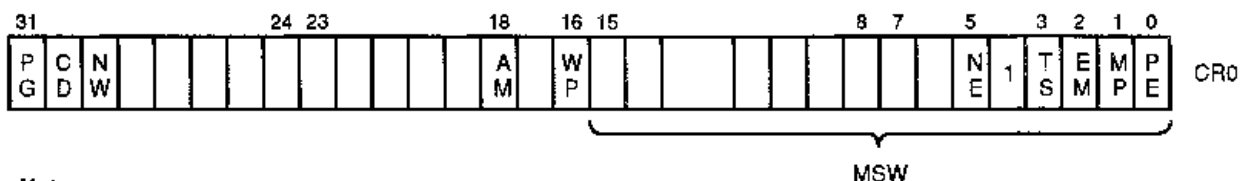
Note:
1 indicates AMD Reserved (Do Not Define); see Section 2.2.6.

17852A–008

2.2.2.1   Control Registers

Control Register 0 (CR0)

CR0 (see Figure 2-5) contains 10 bits for control and status purposes. Five of the bits defined in the Am486DX/DX2 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:

- Am486DX/DX2 Microprocessor Operating Modes: PG, PE (see Table 2-2)

- On-Chip Cache Control Modes: CD, NW (see Table 2-3)

- On-Floating-Point Unit Control: TS, EM, MP, NE (see Table 2-4)

- Alignment Check Control: AM

- Supervisor Write Protect: WP

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 Am486DX/DX2 microprocessor work the same as the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and they ignore the

new bits). New Am486DX/DX2 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 is not filled on cache misses. When CD = 0, cache fills can be performed on misses (see Table 2-3).

The state of the CD bit, the cache enable input pin ($\overline{KEN}$), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss is installed in the cache. A line is installed in the cache only if CD = 0 and $\overline{KEN}$ 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 386 microprocessor compatible mode. 386 microprocessor software can load incorrect data into the AC bit in the EFLAGS register. Setting AM = 0 will prevent AC faults from occurring before the Am486DX/DX2 microprocessor has created the AC interrupt service routine.

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 submode 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 raises a GP fault with error code 0.
1	1	Page Protection 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 combination 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

WP   (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The 386 microprocessor allows a read-only page to be written from privilege levels 0–2. The Am486DX/DX2 microprocessor is compatible with the 386 microprocessor when WP = 0. WP = 1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on the WP bit.

NE   (Numerics Exception, bit 5)

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

For any UFPE, the floating-point error output pin (FERR) is driven active.

For NE = 0, the Am486DX/DX2 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 Am486DX/DX2 microprocessor freezes immediately before executing the next floating-point instruction. An external interrupt controller supplies an interrupt vector when FERR is driven active. The UFPE is ignored if IGNNE is active and floating-point execution continues.

Note: The CPU freeze mentioned above 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 results in a software interrupt 16 immediately before executing the next non-control floating-point or WAIT instruction. The $\overline{\text{IGNNE}}$ signal is ignored.

TS (Task Switched, bit 3)

The TS bit is set when a task switch operation is performed. Execution of a floating-point instruction with TS = 1 causes a device not available (DNA) fault (trap vector 7). If TS = 1 and MP = 1 (monitor coprocessor in CR0), a WAIT instruction causes 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 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 (see Table 2-4). The TS bit is set to 1 on task switches by the Am486DX/DX2 microprocessor. Floating-point instructions are unaffected by the state of the MP bit. It is recommended that the MP bit be set to 1 for the normal operation of the Am486DX/DX2 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 Am486DX/DX2 microprocessor operates in Real Mode, with segment-based protection disabled and addresses formed as in an 8086 (see Table 2-2).

All new CR0 bits added to the Am386 CPU and Am486DX/DX2 microprocessors, except for ET and NE, are upward compatible with the 80286. These new bits are in register bits not defined in the 80286. For strict compatibility with the 80286, the LMSW instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future AMD microprocessors.

Control Register 2 (CR2)

CR2 holds the 32-bit linear address that caused the last page fault detected (see Figure 2-6). The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

Control Register 3 (CR3)

CR3 contains the physical base address of the page directory table (see Figure 2-6). The Am486DX/DX2 microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 31–20 in CR3.

In the Am486DX/DX2 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 sources 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 TSS that changes the values in CR3, or an explicit load into CR3 with any value invalidates 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.

2.2.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, 386, and Am486DX/DX2 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 (see Figure 2-4). These registers are GDTR, IDTR, LDTR, and TR, respectively. Chapter 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.

Figure 2-6   Control Registers 2 and 3

31		0	
Page Fault Linear Address Register			CR2

31		4 3	0	
Page Directory Base Register	0 0 0 0 0 0 0	P C D / P W T	0 0 0	CR3
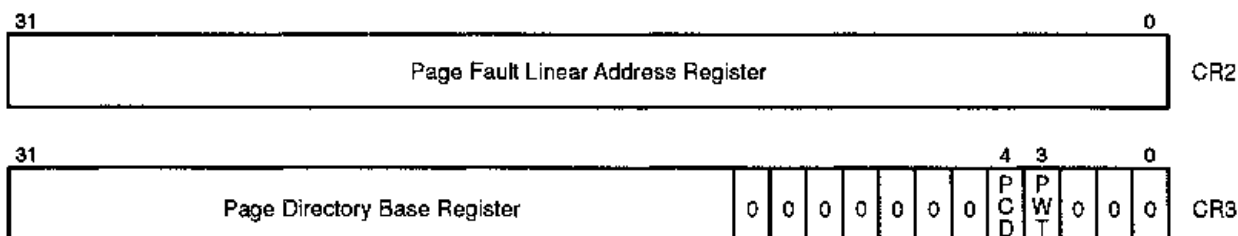
Note:
0 indicates AMD Reserved (Do Not Define); see Section 2.2.6.

17852A–009

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.

2.2.3 Floating-Point Registers

Figure 2-7 shows the floating-point register set. The on-chip floating-point unit (FPU) contains eight data registers, a tag word, a control register, a status register, an instruction pointer, and a data pointer.

The operation of the Am486DX/DX2 microprocessor's on-chip FPU is exactly the same as the 387 math coprocessor. Software written for the 387 math coprocessor runs on the on-chip FPU without any modifications.

2.2.3.1 Data Registers

Floating-point computations use the Am486DX/DX2 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 CPU'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 1 and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by 1. Like other Am486DX/DX2 microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

Instructions can 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.

Figure 2-7   Floating-Point Registers
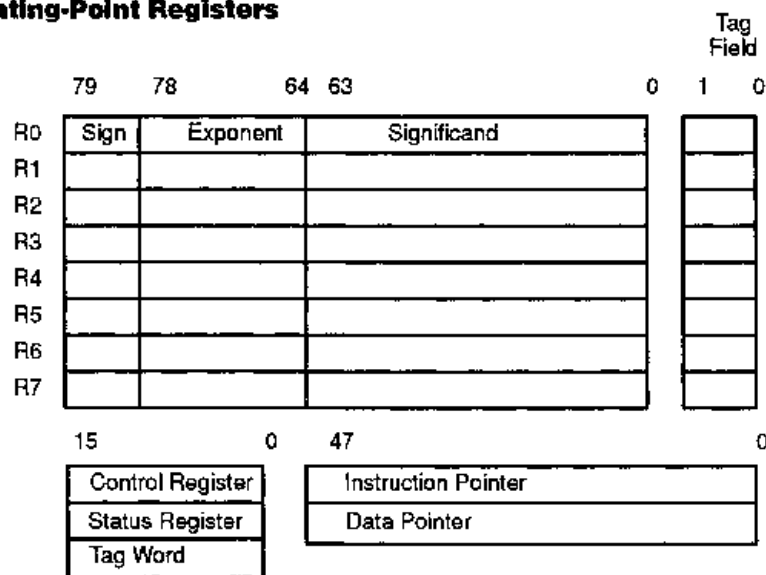
17852A–010

2.2.3.2   Tag Word

The tag word marks the content of each numeric data register (see 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 FPU's performance and stack handling by making it possible to distinguish between empty and non-empty register locations. Tag words also enable exception handlers to check the contents of a stack location without necessitating complex decoding of the actual data.

2.2.3.3   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

Figure 2-8   FPU Tag Word

15							0
TAG (7)	TAG (6)	TAG (5)	TAG (4)	TAG (3)	TAG (2)	TAG (1)	TAG (0)
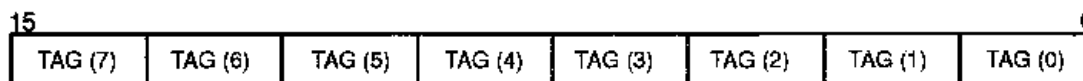
Note:

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

17852A–011

Figure 2-9   FPU Status Word
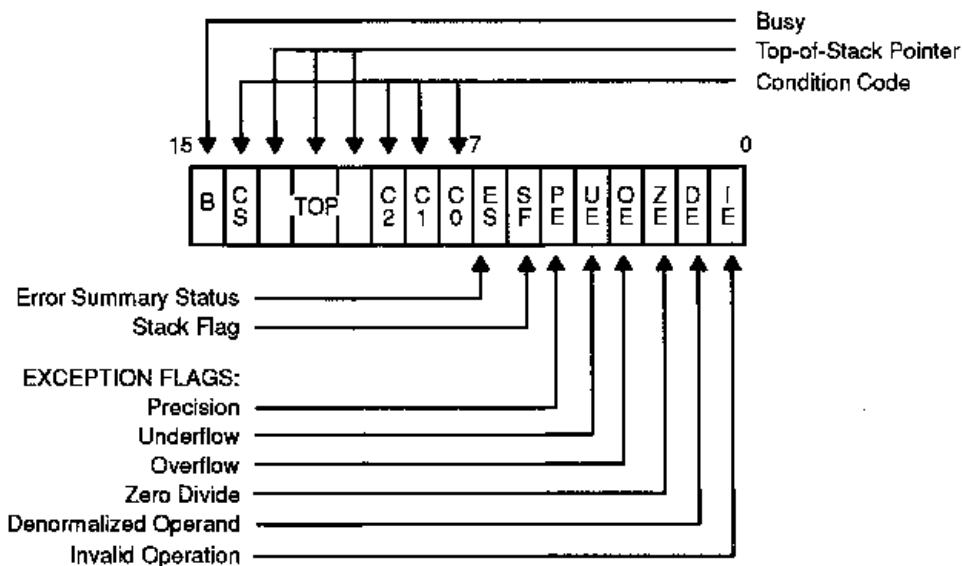
ES is set if any unmasked exception bit is set; cleared otherwise.
See Table 2.5 for interpretation of condition code.
TOP Values:
    000 = Register 0 is Top-of-Stack
    001 = Register 1 is Top-of-Stack
          •
          •
          •
    111 = Register 7 is Top-of-Stack
For definitions of exceptions, refer to the Section entitled
"Exception Handling".

17852A–012

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, C3–C0, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C3–C0 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Table 2-5 through Table 2-8.

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 5–0 in the status word) is set; ES is clear otherwise. The $\overline{\text{FERR}}$ (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 not masked by the control word causes three things to happen: the corresponding exception flag in the status word is set, the ES bit in the status word is set, and the $\overline{\text{FERR}}$ output signal is asserted. When the Am486DX/DX2 microprocessor attempts to execute another floating-point or WAIT instruction, either exception 16 occurs, or an external interrupt occurs if NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves both 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 2.2.3.4).

Note: 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 $\overline{\text{FERR}}$ output of the Am486DX/DX2 microprocessor is activated immediately.

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			Reduction 0 = complete 1 = incomplete
	Q2	Q0	Q1 or O/$\overline{U}$	
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2-7)		Zero or O/$\overline{U}$	Operand is not comparable (Table 2-7)
FXAM	Operand class (see Table 2-9)		Sign or O/$\overline{U}$	Operand class (Table 2-9)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/$\overline{U}$	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/$\overline{U}$	UNDEFINED
FPTAN, FSIN, FCOS, FSINCOS	UNDEFINED		Roundup or O/$\overline{U}$, 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/$\overline{U}$	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 that 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, and 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

2.2.3.4   Instruction and Data Pointers

Because the FPU operates in parallel with the Arithmetic Logic Unit (ALU) (in the Am486DX/DX2 microprocessor the ALU consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU executed the floating-point instruction that caused it. To allow identification of the failing numeric instruction, the Am486DX/DX2 microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

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 Am486DX/DX2 microprocessor decodes a new floating-point instruction, it saves the instruction (including any prefixes that might 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 Am486DX/DX2 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 Am486DX/DX2 microprocessor is in the Virtual 8086 Mode, the Real Address Mode formats are used. The four formats are shown in Figure 2-10 through Figure 2-13. The floating-point instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values to and from memory. Note that the data pointer value 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.

2.2.3.5   FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2-14 shows the format and encoding of fields in the control word.

Figure 2-10   Protected Mode FPU Instruction and Data Pointer Image In Memory, 32-Bit Format

32-bit Protected Mode Format

31	23	15	7	0	
Reserved		Control Word			0
Reserved		Status Word			4
Reserved		Tag Word			8
IP Offset					C
00000	Opcode 10..0	CS Selector			10
Data Operand Offset					14
Reserved		Operand Selector			18
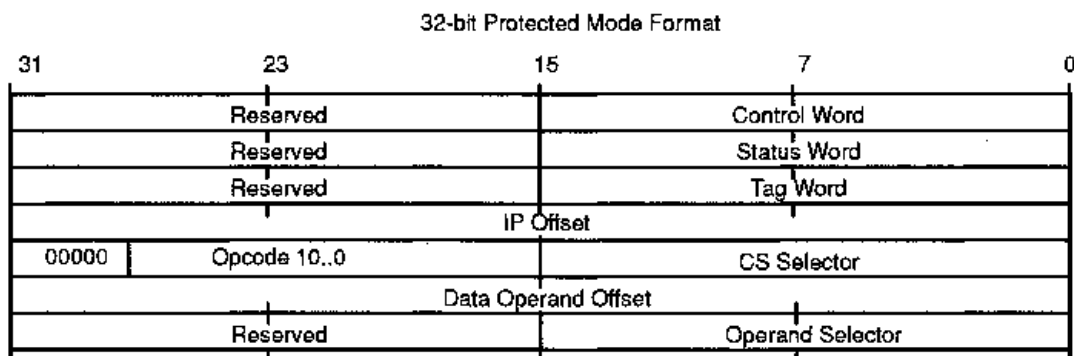
17852A–013

Figure 2-11   Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

32-bit Real Address Mode Format

31	23	15	7		0	
Reserved		Control Word				0
Reserved		Status Word				4
Reserved		Tag Word				8
Reserved		Instruction Pointer 15..0				C
0000	Instruction Pointer 31..16	0	Opcode 10..0			10
Reserved		Operand Pointer 15..0				14
0000	Operand Pointer 31..16		0000	00000000		18
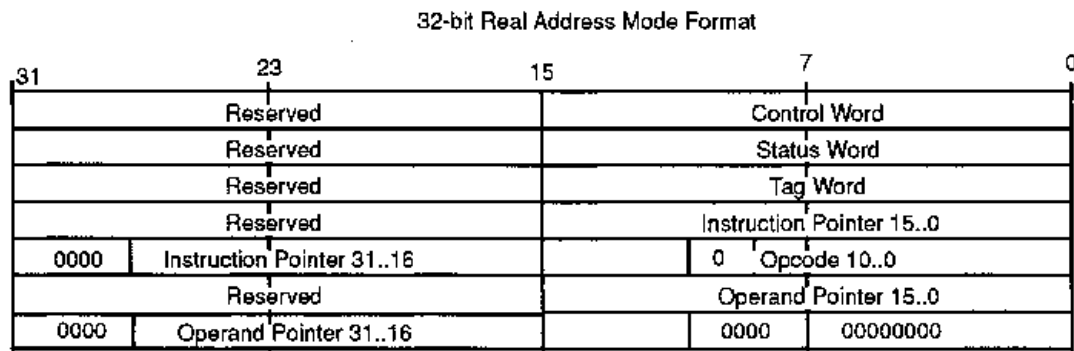
17852A–014

Figure 2-12   Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

16-bit Protected Mode Format

15	7	0	
Control Word			0
Status Word			2
Tag Word			4
IP Offset			6
CS Selector			8
Operand Offset			A
Operand Selector			C
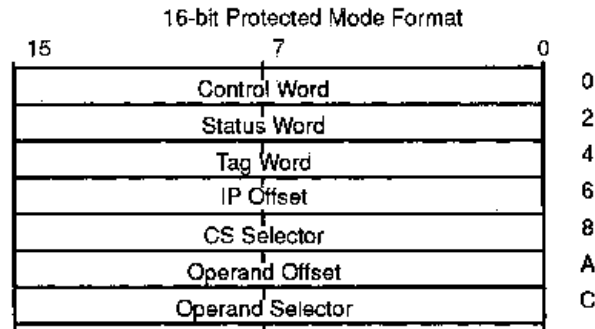
17852A–015

Figure 2-13   Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

16-bit Real-address Mode And
Virtual-8086 Mode Format

15		7	0	
Control Word				0
Status Word				2
Tag Word				4
Instruction Pointer 15..0				6
IP19.16	0	Opcode 10..0		8
Operand Pointer 15..0				A
DP 19.16	0	0 0 0 0 0 0 0 0 0 0 0		C
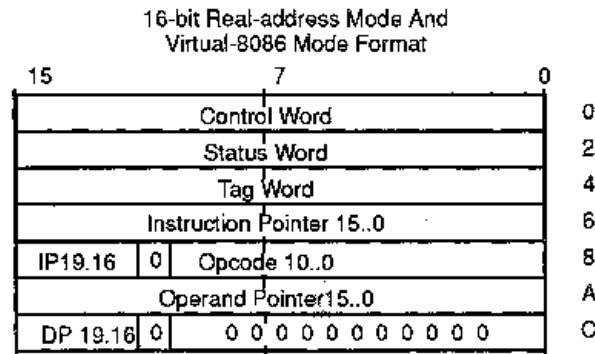
17852A–016

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 5–0 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 11–10)

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, FA8S, and FCHS), and all transcendental instructions.

PC   (Precision Control, bits 9–8)

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 opcode determines the precision or extended precision is used.

2.2.4 Debug and Test Registers

2.2.4.1 Debug Registers

The six programmer accessible debug registers (see Figure 2-15) provide on-chip support for debugging. Debug registers DR3–DR0 specify the four linear breakpoints. The Debug control register DR7 is used to set the breakpoints. The Debug status register DR6 displays the breakpoint's current state. The use of the Debug registers is described in Chapter 9.

2.2.4.2 Test Registers

The Am486DX/DX2 microprocessor contains five test registers (see Figure 2-15). TR6 and TR7 are used to control the TLB testing. TR3, TR4, and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Chapter 8.

2.2.5 Register Accessibility

There are a few differences regarding the accessibility of the registers in Real and Protected Mode (see Table 2-10). See Chapter 4, Protected Mode Architecture, for further details.

2.2.6 Compatibility With Future Processors

In the proceeding register descriptions, note certain Am486DX/DX2 microprocessor register bits are AMD reserved. When reserved bits are called out, treat them as fully undefined. This is essential for software compatibility with future processors! Follow these guidelines:

- Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.

- Do not depend on the states of any undefined bits when storing them to memory or another register.

- Do not depend on retaining information written into any undefined bits.

- When loading registers, always load the undefined bits as zeros.

- However, registers that have been previously stored can be reloaded without masking.

Depending upon the values of undefined register bits makes software dependent upon the unspecified Am486DX/DX2 microprocessor handling of these bits. Depending on undefined values risks making software incompatible with future processors that define usage for the Am486DX/DX2 microprocessor undefined bits.

AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED Am486DX/DX2 MICROPROCESSOR REGISTER BITS.

2.3 INSTRUCTION SET

The Am486DX/DX2 microprocessor instruction set can be divided into eleven 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 Am486DX/DX2 microprocessor instructions are listed in Chapter 10. Note that all floating-point unit instruction mnemonics begin with an F.

All Am486DX/DX2 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 0 operand instructions (e.g., CLI, STI) take only one byte. Generally, 1 operand instructions are two bytes long. The average instruction is 3.2 bytes long. Since the Am486DX/DX2 microprocessor has a 32-byte instruction queue, an average of 10 instructions is prefetched. Using 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 the Am486DX/DX2 or 386 microprocessor's code (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 that 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).

2.4 MEMORY ORGANIZATION

Memory on the Am486DX/DX2 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.

Figure 2-14   FPU Control Word

RESERVED
RESERVED*
ROUNDING CONTROL
PRECISION CONTROL

15   7   0

| X | X | X | X | RC | PC | X | X | P M | U M | O M | Z M | D M | I M |

RESERVED

EXCEPTION MASKS:
Precision
Underflow
Overflow
Zero Divide
Denormalized Operand
Invalid Operation

* "0" after RESET or FINIT;
changeable upon loading the control word
(CW). Programs must ignore this bit.

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

Rounding Control
00-Round to nearest or even
01-Round down (toward − ∞)
10-Round up (toward + ∞)
11-Chop (truncate toward zero)

17852A-017

Figure 2-15   Debug and Test Registers

Debug Registers

Linear Breakpoint Address 0	DR0
Linear Breakpoint Address 1	DR1
Linear Breakpoint Address 2	DR2
Linear Breakpoint Address 3	DR3
Reserved	DR4
Reserved	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

17852A-017

Table 2-9   FPU Exceptions

Exception	Cause	Default Action (If exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0^* \infty$, $0/0$, $(+\infty) + (-\infty)$, 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.

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 8086 Mode.

In addition to these basic data types, the Am486DX/DX2 microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments that 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 Am486DX/DX2 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 is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.4.1 Address Spaces

The Am486DX/DX2 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 a segment register's contents. An offset is formed by summing all of the addressing components (BASE, INDEX, and DISPLACEMENT), discussed in Section 2.6.2, into an effective address. Since each task on the Am486DX/DX2 microprocessor has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 Gbytes (2^{32} bits), this gives a total of 2^{46} bits (or 64 Tbytes) 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 shows the relationship between the various address spaces.

2.4.2 Segment Register Usage

The main data structure used to organize memory is the segment. On the Am486DX/DX2 microprocessor, segments are variable sized blocks of linear addresses that 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 Gbytes (2^{32} bytes).

In order to provide compact instruction encoding and increased processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the Segment Register Selection Rules (see Table 2-11). 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.

Figure 2-16   Address Translation
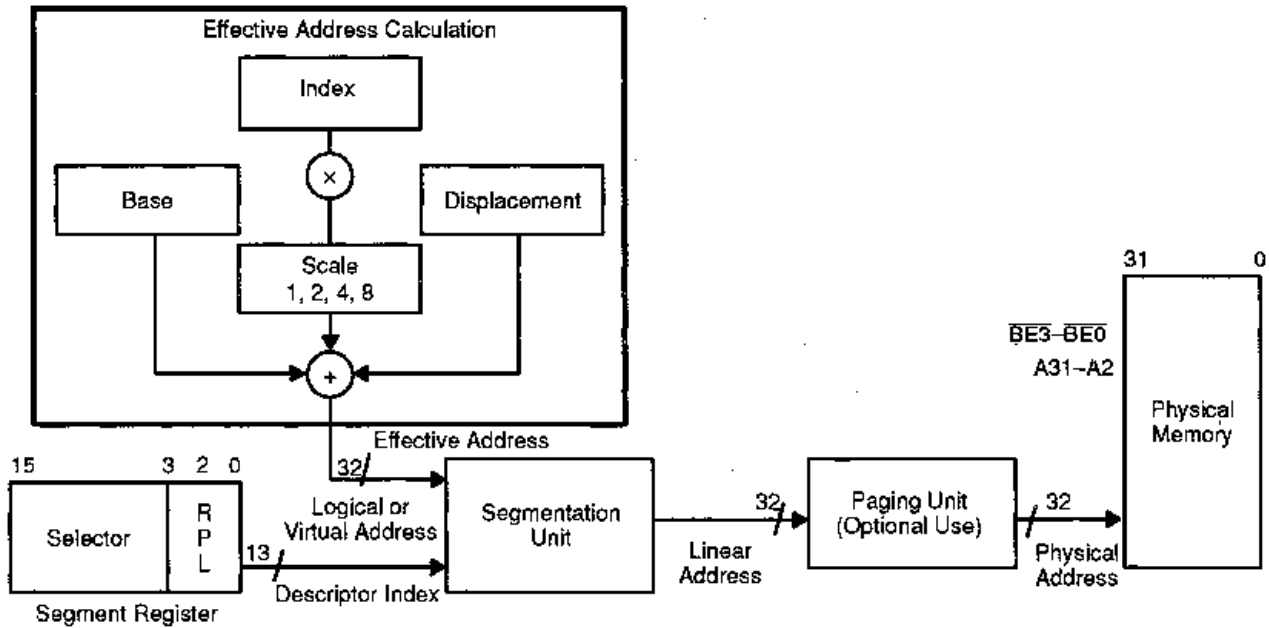
17852A-018

Figure 2-17   Addressing Mode Calculations
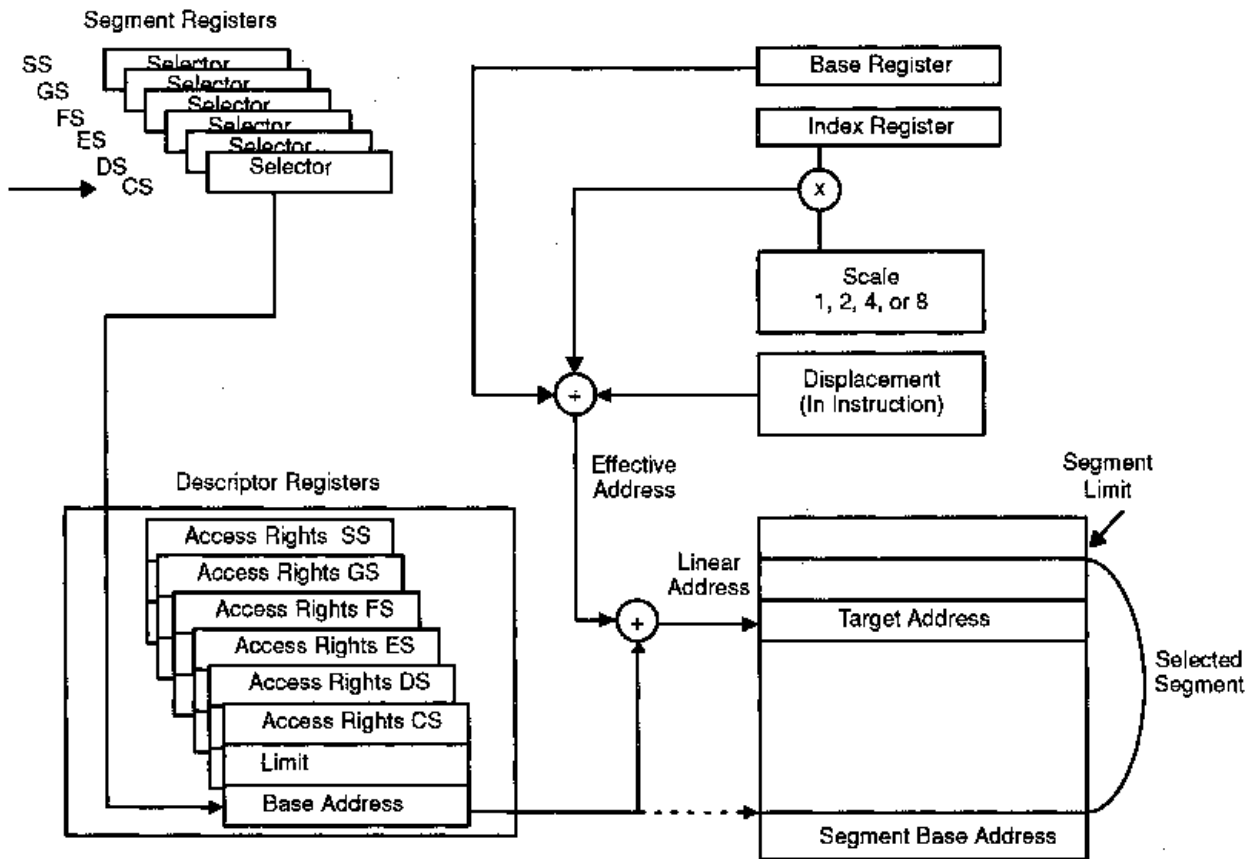
17852A-019

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] [EBX] [ECX] [EDX] [ESI] [EDI] [EBP] [ESP]	DS DS DS DS DS DS SS SS	All

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all six segments can have the base address set to 0 and create a system with a 4-Gbyte 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 4.3.

2.5   I/O SPACE

The Am486DX/DX2 microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Am486DX/DX2 microprocessor also supports memory mapped peripherals. The I/O space consists of 64 Kbytes and can be divided into 64K 8-bit ports, 32K 16-bit ports, 16K 32-bit ports, or any combination of ports that total 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 AMD.

2.6   ADDRESSING MODES

The Am486DX/DX2 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.

2.6.1 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.

2.6.2 32-Bit Memory Addressing Modes

The remaining nine 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 four components make up the nine 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.

The effective address (EA) of an operand is calculated according to the following formula (see Figure 2-17).

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 are added to a DISPLACEMENT to form the operand's offset.

Example: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents are added to a DISPLACEMENT to form the operand's offset.

Example: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents are multiplied by a scaling factor that 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 are 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 are multiplied by a SCALING factor. The result is added to the contents of a BASE register to obtain the operand's offset.

Example: MOV ECX, [EDX*8] [EAX]

Based Index Mode with Displacement: The contents of an INDEX register, a BASE register's contents, and a DISPLACEMENT are added together to form the operand's 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]

2.6.3   Differences Between 16- and 32-Bit Addresses

In order to provide software compatibility with the 80386, 80286, and the 8086, the Am486DX/DX2 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 Default Operation Size (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 Am486DX/DX2 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 the 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. The AMS486 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 together to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address exceeding FFFFH results in a General Protection Fault. An Address Length Prefix only allows the use of the additional Am486DX/DX2 microprocessor addressing modes.

When executing 32-bit code, the Am486DX/DX2 microprocessor uses either 8- or 32-bit displacements, and any register can be used as a base or index register. When executing 16-bit code, the displacements are either 8 or 16 bits, and the base and index registers conform to the 80286 model (see Table 2-12).

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

2.7   DATA FORMATS

2.7.1   Data Types

The Am486DX/DX2 microprocessor supports a wide variety of data types. In the following descriptions, the on-chip FPU consists of the floating-point registers. The CPU consists of the base architecture registers.

2.7.1.1   Unsigned Data Types

The FPU does not support unsigned data types (see 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. The most significant bit (MSB) is 7.

2.7.1.2   Signed Data Types

All signed data types assume 2s complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the MSB. The number is negative if the sign bit is 1. The number is positive if the sign bit is 0. The magnitude field consists of the remaining bits in the number (see 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.

2.7.1.3   Floating-Point Data Types

Floating-point data type in the Am486DX/DX2 microprocessor contains 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. The number is positive if the sign bit is 0. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand (see 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.

2.7.1.4 BCD Data Types

The Am486DX/DX2 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 3–0 and the upper digit is in bits 7–4. An unpacked BCD data type contains 1 digit per byte stored in bits 3–0.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types (see Table 2-13).

2.7.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words, or dwords. A string can contain between 1 byte and 4 Gbytes (see 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 Am486DX/DX2 microprocessor, bit strings can be up to 4-Gbits long.

2.7.1.6 ASCII Data Types

The Am486DX/DX2 microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCII data (see Table 2-14).

Table 2-13 Am486DX/DX2 Microprocessor Data Types

Supported by Base Registers		Supported by FPU				Least Significant Byte

Data Format			Range	Precision	(bit layout)
Byte	X		0–255	8 bits	7 — 0
Word	X		0–64K	16 bits	15 — 0
Dword	X		0–4G	32 bits	31 — 0
8-Bit Integer	X		10²	8 bits	Two's Complement Sign Bit; 7 — 0
16-Bit Integer	X	X	10⁴	16 bits	Two's Complement Sign Bit; 15 — 0
32-Bit Integer	X	X	10⁹	32 bits	Two's Complement Sign Bit; 31 — 0
64-Bit Integer		X	10¹⁹	64 bits	Two's Complement Sign Bit; 63 — 0
8-Bit Unpacked BCD	X		0–9	1 Digit	One BCD Digit per Byte; 7 — 0
8-Bit Packed BCD	X		0–9	2 Digits	Two BCD Digits per Byte; 7 — 0
80-Bit Packed BCD		X	±10±18	18 Digits	79 72 ... 0; Ignored; Sign Bit
Single Precision Real		X	±10±38	24 Bits	31 23 0; Biased Exp.; Significand; Sign Bit
Double Precision Real		X	±10±308	53 Bits	63 52 0; Biased Exp.; Significand; Sign Bit
Extended Precision Real		X	±10±4932	64 Bits	79 63 0; Biased Exp.; 1; Significand; Sign Bit

10^2, 10^4, 10^9, 10^{19} ; $\pm10^{\pm18}$, $\pm10^{\pm38}$, $\pm10^{\pm308}$, $\pm10^{\pm4932}$

Table 2-14   String and ASCII Data Types

String Data Types

Byte String

Word String

Dword String

Bit String

ASCII Data Types

ASCII Character
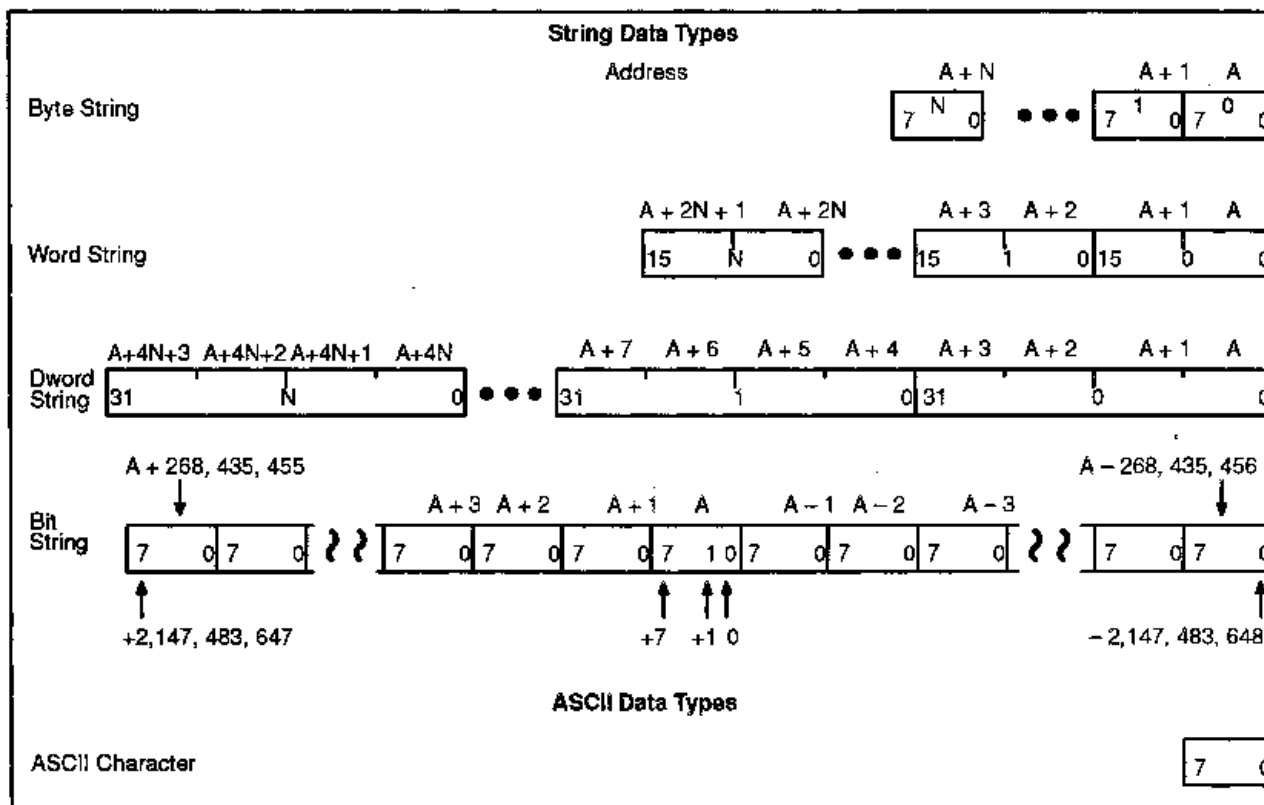
2.7.1.7   Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The Am486DX/DX2 microprocessor supports two types of pointers (see Table 2-15).

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

32-bit pointer: 32-bit offset

2.7.2   Little Endian vs Big Endian Data Formats

The Am486DX/DX2 microprocessor, as well as all other members of the Am486 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; the low-order byte is at the lowest address, the high-order byte is at the highest 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 Am486DX/DX2 microprocessor has two instructions that 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.

Table 2-15   Pointer Data Types

Least Significant Byte

Data Format										

48-Bit
Pointer

```
      47        31              0
        | Selector |   Offset   |
```

32-Bit
Pointer

```
                31              0
                   |   Offset   |
```
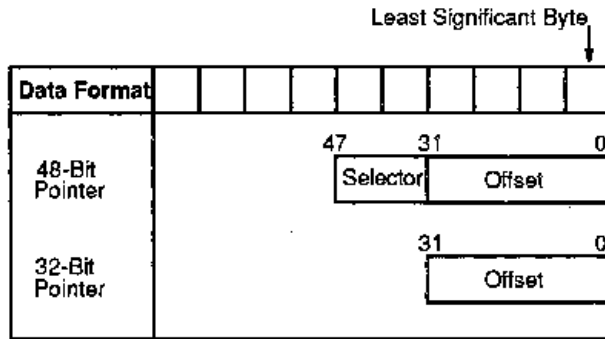
2.8   INTERRUPTS

2.8.1   Interrupts and Exceptions

In order to handle external events, interrupts and exceptions alter the normal program flow to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events, 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 2.8.3 and 2.8.4 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 that caused the exception is supported. Faults are exceptions that are detected and serviced before the execution of the faulting instruction. A fault occurs in a virtual memory system when the processor references a page or a segment that is not present. The operating system fetches the page or segment from disk and the Am486DX/DX2 microprocessor restarts the instruction. Traps are exceptions that are reported immediately after the execution of the instruction that caused the problem. User-defined interrupts are examples of traps. Aborts are exceptions that 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.

Figure 2-18   Big vs Little Endian Memory Format

```
        m + 3      m + 2      m + 1        m
    31     24 23      16 15      8 7        0
    |        |         |         |         |
         Dword in Little-Endian Memory Format
```

```
        m         m + 1      m + 2      m + 3
    31     24 23      16 15      8 7        0
    |        |         |         |         |
         Dword in Big-Endian Memory Format
```
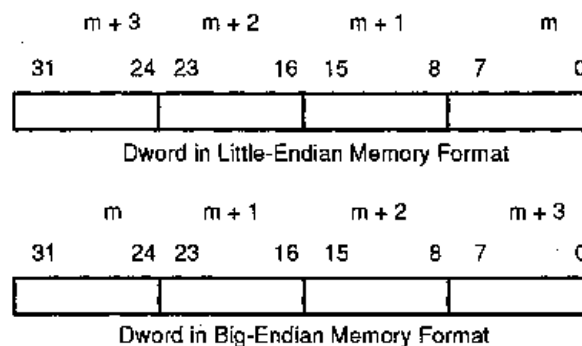
17852A–020

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 always points to the instruction causing the exception and includes any leading instruction prefixes. Table 2-16 summarizes the possible interrupts for the Am486DX/DX2 microprocessor and shows where the return address points.

The Am486DX/DX2 microprocessor can 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 that are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by the Am486DX/DX2 microprocessor, the remaining 224 can be used by the system designer.

2.8.2 Interrupt Processing

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the Am486DX/DX2 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 Am486DX/DX2 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.

2.8.3 Maskable Interrupt

Maskable interrupts are the most common way the Am486DX/DX2 microprocessor responds 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. This allows interrupts during long string moves). When an interrupt occurs, the processor reads an 8-bit vector supplied by the hardware. This vector 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.

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 can 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.

2.8.4 Non-Maskable Interrupt

Non-maskable interrupts (NMI) provide a method of servicing very high priority interrupts. A common example of the use of a NMI is activating 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 Am486DX/DX2 microprocessor does 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 is 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.

2.8.5 Software Interrupts

A third type of interrupt/exception for the Am486DX/DX2 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 the program as a debugging tool. A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.8.6 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 Am486DX/DX2 microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Am486DX/DX2 microprocessor invokes the appropriate interrupt service routine. The priority for invoking service routines in case of simultaneous external interrupts is

1. NMI

2. INTR

Exceptions are internally-generated events. Exceptions are detected by the Am486DX/DX2 CPU if, in the course of executing an instruction, the Am486DX/DX2 microprocessor detects a problematic condition. The Am486DX/DX2 microprocessor then immediately invokes the appropriate exception service routine. The state of the Am486DX/DX2 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 executes without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand can 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 Am486DX/DX2 microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in the following paragraphs. This cycle is repeated as each instruction is executed and occurs in parallel with instruction decoding and execution.

Table 2-16   Interrupt Vector Assignments

Function	Interrupt Number	Instructions That 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 Opcode	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
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
Reserved	15			
Floating-Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Reserved	18–31			
Two-Byte Interrupt	0–255	INT n	NO	TRAP

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

Consider the case of the Am486DX/DX2 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 the 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 FPU, check if EM = 1 or TS = 1 (exception 7 if either are 1).

9. If opcode for 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.

2.8.7 Instruction Restart

The Am486DX/DX2 microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10), the Am486DX/DX2 microprocessor invokes the appropriate exception service routine. The Am486DX/DX2 microprocessor is in a state that permits restart of the instruction for all cases but those in the following paragraph. All such cases are easily avoided by proper design of the operating system.

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 TSSs can be avoided either by keeping the TSSs 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 above.

2.8.8 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) is also 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 Am486DX/DX2 microprocessor invokes the exception service routine for exception 8.

2.8.9 Floating-Point Interrupt Vectors

Several interrupt vectors of the Am486DX/DX2 microprocessor are used to report exceptional conditions while executing numeric programs in either Real or Protected Mode. Table 2-17 shows these interrupts and their causes.

Table 2-17   Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A floating-point instruction is encountered when EM or TS or the 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 might not belong to the current task.
13	The first word or dword 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 to 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 Am486 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 REAL MODE ARCHITECTURE

3.1 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 Am486DX/DX2 microprocessor's 32-bit register set. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286 (see Figure 3-1).

All of the Am486DX/DX2 microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The Real Mode default operand size 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 of the Am486DX/DX2 microprocessor in Real Mode is 64 Kbytes, so 32-bit effective addresses must have a value less than 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the Am486DX/DX2 microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Am486DX/DX2 microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Am486DX/DX2 microprocessor cannot 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 cannot be supported during repeated string instructions.

Table 3-1 shows the only instruction forms where the LOCK prefix is legal on the Am486DX/DX2 microprocessor.

An exception 6 is generated if a LOCK prefix is placed before any instruction form or opcode not listed in Table 3-1. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions in Table 3-1. For example, even

Figure 3-1   Real Address Mode Addressing
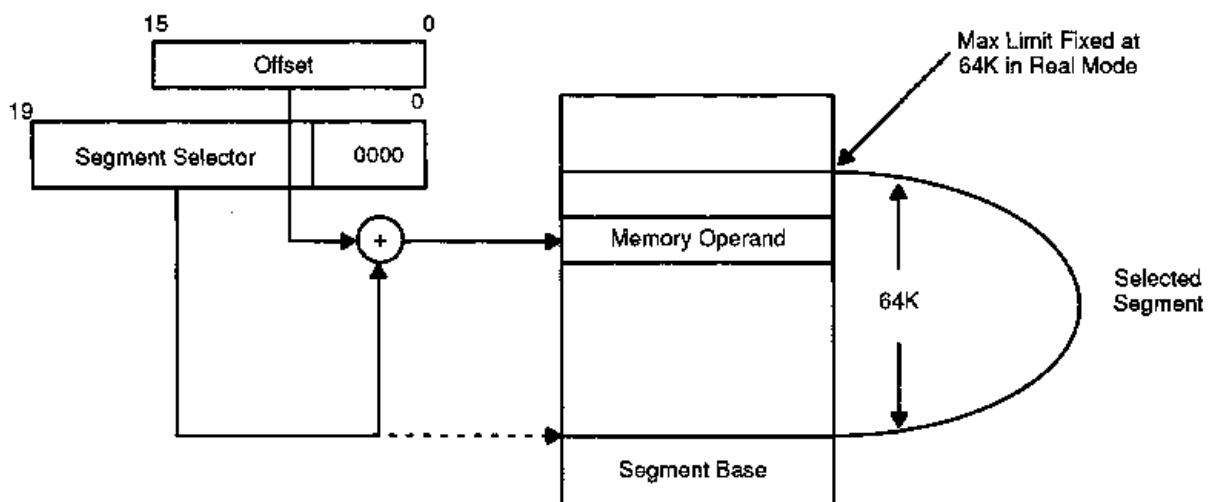
17852A–021

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 Am486DX/DX2 microprocessor, repeated string instructions are not LOCKable, it is impossible to LOCK the bus for a long time. Therefore, the LOCK prefix is not IOPL-sensitive on the Am486DX/DX2 microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed in Table 3-1.

3.2   MEMORY ADDRESSING

In Real Mode the maximum memory size is limited to 1 Mbyte. Thus, only address lines A19–A2 are active. (Exception: after RESET, address lines A31–A2 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 and 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 can be read, written, or executed. The Am486DX/DX2 microprocessor generates 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 a high byte at 0000H).

Segments can 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 overlapping lets the programmer minimize the physical memory needed for a program.

3.3   RESERVED LOCATIONS

There are two fixed areas in memory that are reserved in Real Address Mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations 00000009H–FFFFFFFFH are reserved for system initialization.

Table 3-1   Legal LOCK Prefix Instruction Forms

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

3.4 INTERRUPTS

Many of the exceptions covered in Table 3-2 and Section 2.8 are not applicable to Real Mode operation; in particular, exceptions 10, 11, 14, and 17 do not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode (see Table 3-2).

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 forces the Am486DX/DX2 microprocessor out of halt. If interrupted, the saved CS:IP points to the next instruction after the HLT.

Like Protected Mode, the shutdown occurs when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

1. An interrupt or an exception occurs (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).

2. 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 Am486DX/DX2 CPU is unable to execute the NMI and executes another shutdown cycle. In this case, the processor remains in shutdown and can only exit via the RESET input.

Table 3-2   Exceptions with Different Meanings in Real Mode

Function	Interrupt Number	Related Instruction	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 PROTECTED MODE ARCHITECTURE

4.1 INTRODUCTION

The complete capabilities of the Am486DX/DX2 microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to 4 Gbytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 Tbytes or 2^{46} bytes). In addition, Protected Mode allows the Am486DX/DX2 microprocessor to run all existing 8086, 80286, and 386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The Am486DX/DX2 microprocessor base architecture 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 determines the linear base address of a segment and 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 specifies 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 that operates only in Protected Mode. Paging provides a means of managing the very large segments of the Am486DX/DX2 microprocessor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address that comes from the segmentation unit into a physical address. Figure 4-2 shows the complete Am486DX/DX2 microprocessor addressing mechanism with paging enabled.

4.3 SEGMENTATION

4.3.1 Introduction

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

Figure 4-1   Protected Mode Addressing

48/32 Bit Pointer

Selector | Offset

47 | 31 | 15 | 0

Access Rights
Limit
Base Address

Segment
Descriptor

Memory Operand

Up To
4 Gbytes

Segment Limit

Selected
Segment

Segment Base
Address
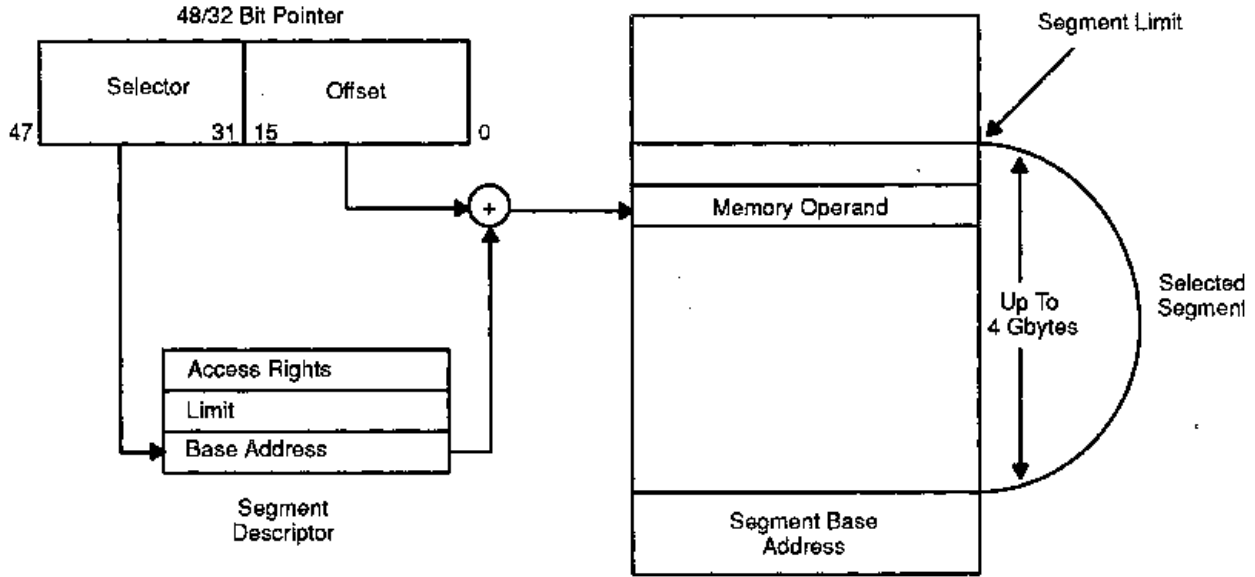
17852A–022

Figure 4-2   Paging and Segmentation

48-Bit Pointer

Segment | Offset

15 | 31 | 0

Access Rights
Limit
Base Address

Segment
Descriptor

32

Linear
Address

Am486 CPU
Paging
Mechanism

Physical
Address

Page
Frame
Address

Physical Address

Memory Operand

4 Kbytes
4 Kbytes
4 Kbytes
Physical
Page:
4 Kbytes
4 Kbytes
4 Kbytes
4 Kbytes
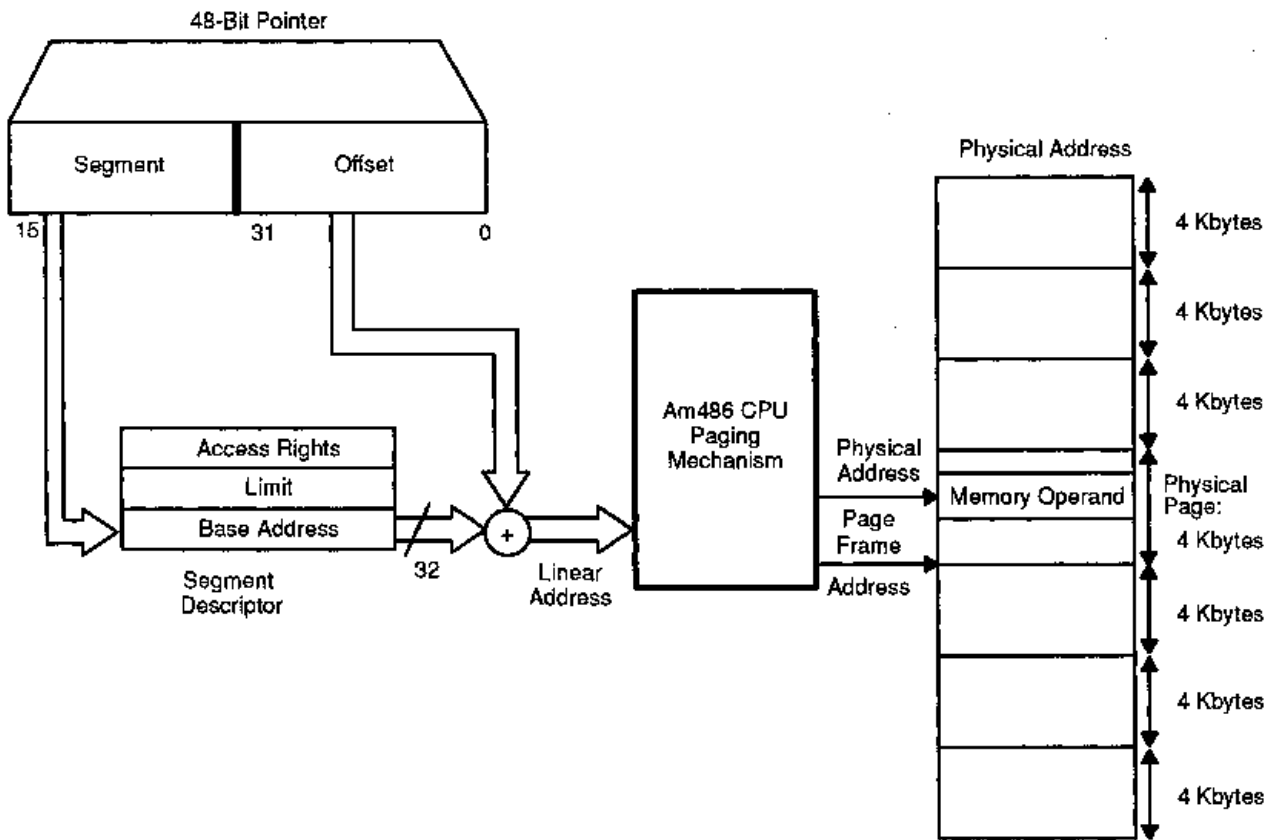
17852A–023

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 and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requester Privilege Level

The privilege level of the original supplier of the selector. RPL is determined by the two least significant bits of a selector.

DPL: Descriptor Privilege Level

This is the least privileged level where a task can access that descriptor (and the segment associated with the descriptor). DPL is determined by bits 6–5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level

The privilege level where a task is currently executing. CPL equals the privilege level of the executing code segment. 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 program execution. Tasks are also referred to as processes.

4.3.3 Descriptor Tables

4.3.3.1 Introduction

The descriptor tables define all the segments that an Am486DX/DX2 microprocessor system uses. Three types of tables on the Am486DX/DX2 microprocessor hold descriptors: the Global Descriptor Table (GDT), Local Descriptor Table (LDT), and the Interrupt Descriptor Table (IDT). All three tables are variable length memory arrays. They range in size between 8 bytes and 64 Kbytes. Each table holds 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 that hold the 32-bit linear base address and the 16-bit limit of each table.

Each table is associated with a register: the GDTR, LDTR, and the IDTR (see Figure 4-3). The LGDT, LLDT, and LIDT instructions load the base and limit of the GDT, LDT, and IDT, respectively, into the appropriate register. The SGDT, SLDT, and SIDT instructions store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

4.3.3.2 Global Descriptor Table

The GDT contains descriptors that are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor, except for descriptors that are used for servicing interrupts (i.e., interrupt and trap descriptors). Every Am486DX/DX2

Figure 4-3   Descriptor Table Registers

17852A–024
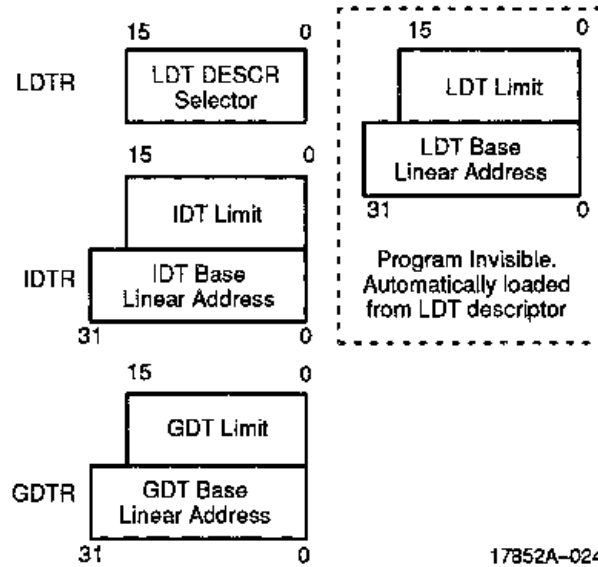
microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the GDT 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 that are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT can contain only code, data, stack, task gate, and call gate descriptors. LDTs allow a mechanism to isolate a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments that 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 that contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to an LDT descriptor in the GDT.

4.3.3.4   Interrupt Descriptor Table

The third table needed for Am486DX/DX2 microprocessor systems is the IDT (see Figure 4-4). The IDT contains the descriptors that point to the location of up to 256 interrupt service routines. The IDT can 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 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 2.8, Interrupts).

Figure 4-4   Interrupt Descriptor Table Register Usage
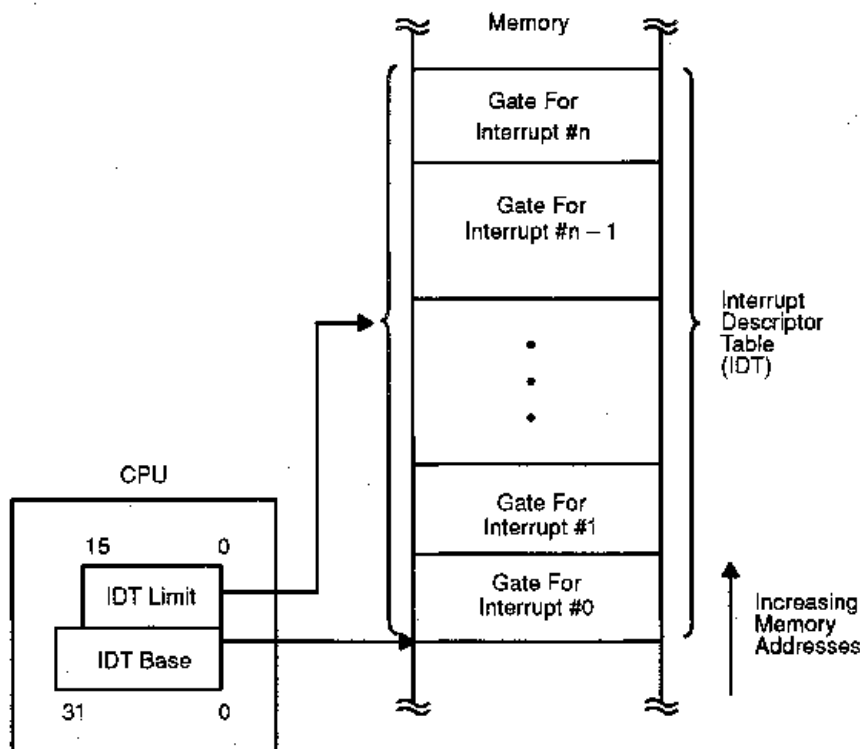
17852A–025

4.3.4   Descriptors

4.3.4.1   Descriptor Attribute Bits

The object the segment selector points to is called a descriptor. Descriptors are 8-byte quantities that 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 Am486DX/DX2 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 DPL is a two-bit field that specifies the protection level 0–3 associated with a segment.

The Am486DX/DX2 microprocessor has two main segment categories: 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   Am486DX/DX2 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.

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

Figure 4-5   General Format of Segment Descriptors

31   0   Byte Address

| Base Address 15...0 | Segment Limit 15...0 | 0 |

| Base 31...24 | G | D | 0 | AVL | Limit 19...16 | P | DPL | S | Type | A | Base 23...16 | +4 |
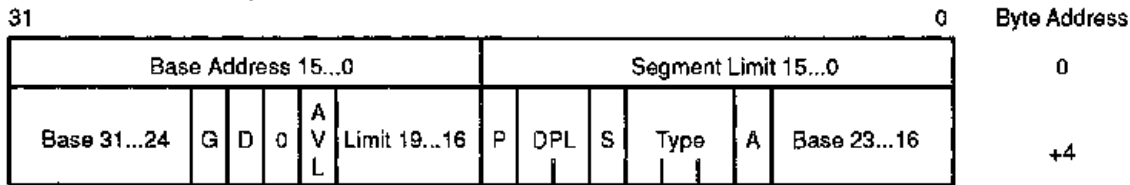
0   Bit must be zero (0) for compatibility with future processors
A   Accessed Bit
AVL   Available field for user or operating system
Base   Base Address of the segment
D   Default Operation size (recognized in code segment descriptors only)
        1 = 32-bit segment; 0 = 16-bit segment
DPL   Descriptor Privilege Level 0–3
G   Granularity Bit: 1 = Segment length is page granular; 0 = Segment length is byte granular
Limit   Length of the segment
P   Present Bit: 1 = Present; 0 = Not Present
S   Segment Descriptor: 0 = System Descriptor; 1 = Code or Data Segment Descriptor
Type   Type of Segment

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 11–0 = 000H).

17852A–026

Figure 4-6   Code and Data Segment Descriptors

31   0   Byte Address

| Base Address 15...0 | Segment Limit 15...0 | 0 |

| Base 31...24 | G | D | 0 | AVL | Limit 19...16 | Access Rights Byte | Base 23...16 | +4 |
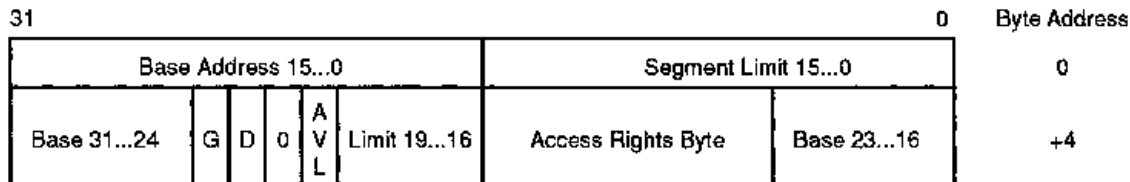
D/B   1 = Default Instruction Attributes are 32 bits
        0 = Default Instruction Attributes are 16 bits
AVL   Available field for user or operating system
G   Granularity Bit: 1 = Segment length is page granular; 0 = Segment length is byte granular
0   Bit must be zero (0) for compatibility with future processors.

17852A–027

systems to keep usage statistics on a given segment. The granularity (G) bit specifies if a segment length is byte-granular or page-granular. Am486DX/DX2 microprocessor segments can be 1 Mbyte long with byte granularity (G = 0), or 4 Gbytes with page granularity (G = 1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. An Am486DX/DX2 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) can 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 can never be written into.

Note: Code segments can be modified via aliases. Aliases are writable data segments that occupy the same range of linear address space as the code segment.

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.	
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	Writable (W)	W = 0	Data segment cannot be written into.	
		W = 1	Data segment can be written into.	
3	Executable (E)	E = 1	Descriptor type is code segment.	If Data 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 cannot be read.	
		R = 1	Code segment can 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.	

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 execute on the Am486DX/DX2 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 Am486DX/DX2 CPU 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 grows 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 8 = 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. Am486DX/DX2 CPU 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 all being 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 GDT.

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

A TSS descriptor contains information about the location, size, and privilege level of a Task State Segment. A TSS is a special fixed format segment that contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate if either 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 Am486DX/DX2 microprocessor TSS. The Task Register (TR) contains the selector that points to the current TSS.

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) that points to the start of a routine, and a word count that specifies how many parameters are to be copied from the caller's stack to the called routine stack. 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

Figure 4-7 System Segment Descriptors
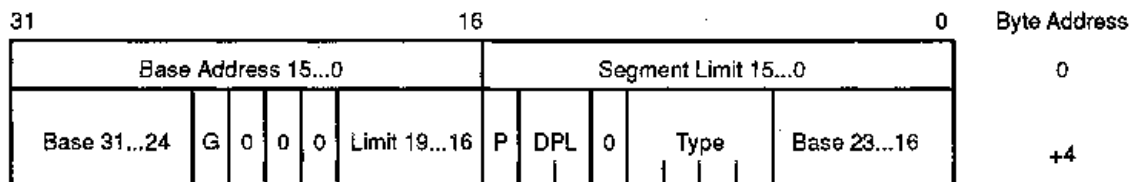
Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Am486 CPU TSS
2	LDT	A	Undefined (reserved)
3	Busy 80286 TSS	B	Busy Am486 CPU TSS
4	80286 Call Gate	C	Am486 CPU Call Gate
5	Task Gate (for 80286/Am486 CPU Task)	D	Undefined (reserved)
6	80286 Interrupt Gate	E	Am486 CPU Interrupt Gate
7	80286 Call Gate	F	Am486 CPU Trap Gate

17852A–028

Figure 4-8 Gate Descriptor Formats

31	24	16				8		5		0	Byte Address

Selector			Offset 15...0								0

Offset 31...16		P	DPL	0	Type	0	0	0	Word Count 4...0	+4
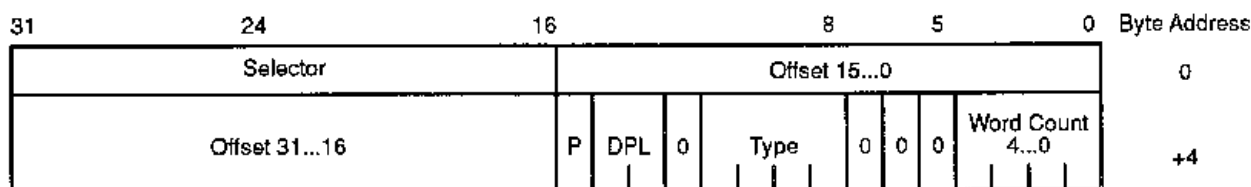
Name	Value	Description
Type	4	80286 Call Gate
	5	Task Gate (for 80286 or Am486 CPU Task)
	6	80286 Interrupt Gate
	7	80286 Call Gate
	C	Am486 CPU Call Gate
	E	Am486 CPU Interrupt Gate
	F	Am486 CPU Trap Gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid
DPL		Least privileged level 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 Am486 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 Am486 CPU	Entry point within the target code segment

17852A–029

difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

Task gates are used to switch tasks. Task gates can 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 can 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 Am486DX/DX2 CPU and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Am486DX/DX2 microprocessor, the Am486DX/DX2 microprocessor supports all 80286 segment descriptors. Figure 4-9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Am486DX/DX2 microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields, have been expanded for the Am486DX/DX2 microprocessor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit. The Am486DX/DX2 microprocessor system segment descriptors have a 32-bit base address, a 2-bit limit field, and a granularity bit.

By supporting 80286 system segments, the Am486DX/DX2 microprocessor can execute 80286 application programs on an Am486DX/DX2 microprocessor operating system. This is possible because the processor automatically distinguishes between 80286-style descriptors and Am486DX/DX2 microprocessor-style descriptors. In particular, if the upper word of a descriptor is 0, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and Am486DX/DX2 microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Am486DX/DX2 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 Requester (the selector's) Privilege Level (RPL) (see 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 is associated with a segment descriptor cache register. 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 reassessing 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 that modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

4.3.4.10   Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the Am486DX/DX2 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 16 times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed 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 can be executed.

Figure 4-9   80286 Code and Data Segment Descriptors

31						0	Byte Address
Segment Base 15...0				Segment Limit 15...0			0
Reserved—Set to Zero (0)	P	DPL	S	Type	Base 23...16		+4

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
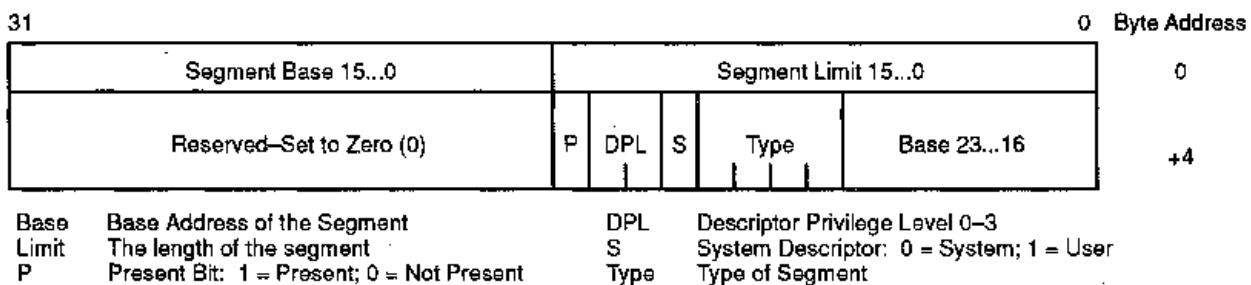
17852A–030

Figure 4-10   Example Descriptor Selection
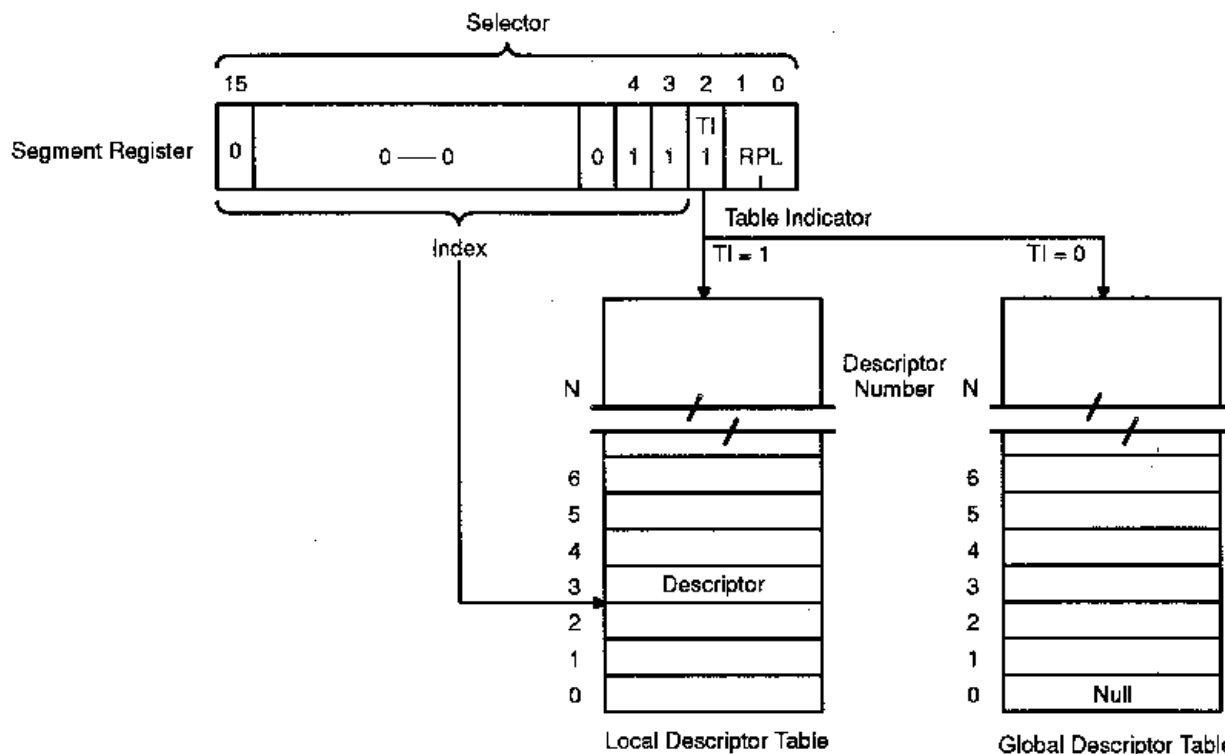
17852A-031

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.

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 16 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 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

4.4   PROTECTION

4.4.1   Protection Concepts

The Am486DX/DX2 microprocessor has four levels of protection. They are optimized to support the needs of a multitasking 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 complex external hardware and software, the Am486DX/DX2 CPU provides the protection as part of its integrated MMU. The Am486DX/DX2 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; in fact, the user/supervisor mode is fully supported by the Am486DX/DX2 microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

Figure 4-11 Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

Segment Descriptor Cache Register Contents

32-Bit Base (Updated during Selector Load into Segment Register)	32-Bit Limit (Fixed)	Other Attributes (Fixed)

Conforming Privilege
Stack Size
Executable
Writeable
Readable
Expansion Direction
Granularity
Accessed
Privilege Level
Present

	BASE	LIMIT										
CS	16X Current CS Selector*	0000FFFFH	Y	0	Y	B	U	Y	Y	Y	–	N
SS	16X Current SS Selector	0000FFFFH	Y	0	Y	B	U	Y	Y	N	W	–
DS	16X Current DS Selector	0000FFFFH	Y	0	Y	B	U	Y	Y	N	–	–
ES	16X Current ES Selector	0000FFFFH	Y	0	Y	B	U	Y	Y	N	–	–
FS	16X Current FS Selector	0000FFFFH	Y	0	Y	B	U	Y	Y	N	–	–
GS	16X Current GS Selector	0000FFFFH	Y	0	Y	B	U	Y	Y	N	–	–

*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.

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

17852A–032

4.4.2 Rules of Privilege

The Am486DX/DX2 microprocessor controls access to both data and procedures between task levels, 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 Am486DX/DX2 microprocessor always executes at one of the four privilege levels (PLs). The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL can only be changed by control transfers, through

Figure 4-12 Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

Segment Descriptor Cache Register Contents

32-Bit Base (Updated during Selector Load into Segment Register)	32-Bit Limit (Updated during Selector Load into Segment Register)	Other Attributes (Updated during Selector Load into Segment Register)

Conforming Privilege
Stack Size
Executable
Writeable
Readable
Expansion Direction
Granularity
Accessed
Privilege Level
Present

	BASE	LIMIT	Present	Privilege Level	Accessed	Granularity	Expansion Direction	Readable	Writeable	Executable	Stack Size	Conforming Privilege
CS	Base per Segment Descriptor		p	d	d	d	d	d	N	Y	--	d
SS	Base per Segment Descriptor	Limit per Segment Descriptor	p	d	d	d	d	r	w	N	d	–
DS	Base per Segment Descriptor		p	d	d	d	d	d	d	N	–	--
ES	Base per Segment Descriptor		p	d	d	d	d	d	d	N	–	–
FS	Base per Segment Descriptor		p	d	d	d	d	d	d	N	–	–
GS	Base per Segment Descriptor		p	d	d	d	d	d	d	N	–	–

Key: Y = Fixed Yes
 N = Fixed No
 d = per segment descriptor
 p = per segment descriptor; but 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

17852A–033

gate descriptors, to a code segment with a different PL. (See Section 4.4.4, Privilege Level Transfers). Thus, an application program running at PL = 3 can call an operating system routine at PL = 1 (via a gate) that causes the task's CPL to be set to 1 until the operating system routine is finished.

4.4.3.2 Selector Privilege (RPL)

The PL 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 the task's effective privilege level (EPL). The EPL is defined as the least privileged (i.e., numerically larger) level of a task's CPL and a selector's RPL. Thus, if the selector's RPL = 0, then the CPL always specifies the privilege level for making an access. 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 usually used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that

Figure 4-13   Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are fixed)

Segment Descriptor Cache Register Contents

32-Bit Base (Updated during Selector Load into Segment Register)	32-Bit Limit (Fixed)	Other Attributes (Fixed)

Conforming Privilege
Stack Size
Executable
Writeable
Readable
Expansion Direction
Granularity
Accessed
Privilege Level
Present

	BASE	LIMIT	Present	Privilege Level	Accessed	Granularity	Expansion Direction	Readable	Writeable	Executable	Stack Size	Conforming Privilege
CS	16X Current CS Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	Y	–	N
SS	16X Current SS Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	N	W	–
DS	16X Current DS Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	N	–	–
ES	16X Current ES Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	N	–	–
FS	16X Current FS Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	N	–	–
GS	16X Current GS Selector	0000FFFFH	Y	3	Y	B	U	Y	Y	N	–	–

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

17852A–034

originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction forces the RPL bits to the originator's CPL.

4.4.3.3   I/O Privilege and I/O Permission Bit-map

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 Am486DX/DX2 microprocessor TSS, the I/O Permission Bit-map (part of an Am486DX/DX2 microprocessor TSS) is consulted on whether I/O to the port is allowed, or if an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bit-map, refer to Figure 4-15 and Figure 4-16. For further information on how the I/O Permission Bit-map is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4, Protection and I/O Permission Bit-map.

The IOPL also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are "IOPL-sensitive" instructions and they

Figure 4-14   Four-level Hierarchical Protection

17852A–035

Figure 4-15   Sample I/O Permission Bit Map

I/O Ports Accessible: 2 → 9, 12, 13, 15, 20 → 24, 27, 33, 34, 40, 41, 48, 50, 52, 53, 58 → 60, 62, 63, 96 → 127

17852A–036

are CLI and STI. (Note that the LOCK prefix is not IOPL sensitive on the Am486DX/DX2 microprocessor.)

The IOPL also affects whether the interrupts enable flag (IF) 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 POPed 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 Am486DX/DX2 microprocessor provides several instructions to speed pointer testing and help maintain system integrity. These instructions verify that the selector value refers to an appropriate segment. Table 4-2 summarizes the selector validation procedures available for the Am486DX/DX2 microprocessor.

Pointer verification prevents the common problem of an application at PL = 3 calling an operating system's routine at PL = 0 and passing the operating system's routine a "bad" pointer that 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.

Figure 4-16   Am486 CPU TSS and TSS Registers

Note:
BIT_MAP_OFF-
SET
must be ≤ DFFFH
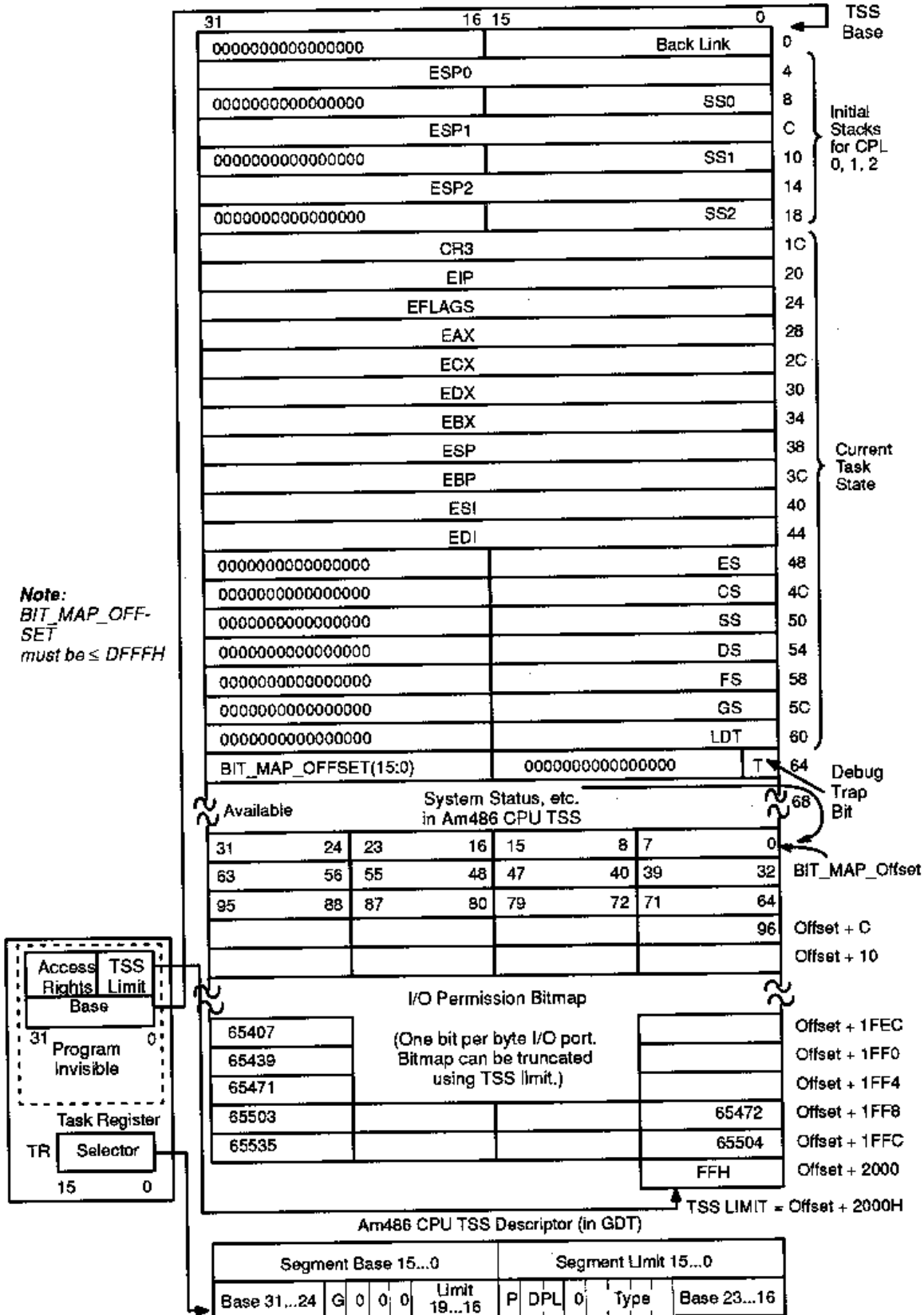
Type = 9: Available Am486 CPU TSS
Type = B: Busy Am486 CPU TSS

17852A–037

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 is changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.5   Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining a task's ability 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, and GS), the Am486DX/DX2 microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, and 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 these rules is readable conforming code segments that 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 stack segment rules are slightly different than data segment rules. Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation 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 (see 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 that loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules causes an exception 13 (e.g., JMP through a call gate, or IRET from a normal subroutine call).

To provide further system security, all control transfers are 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 that are the same or less privileged than the conforming-code segment's DPL.

- Both the 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 can 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

Note:

*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 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 correctly restores 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 the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those that 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 can 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 interlevel Am486DX/DX2 microprocessor call gate is activated, the following actions occur:

1. Load CS:EIP from gate check for validity.

2. SS is pushed zero-extended to 32 bits.

3. ESP is pushed.

4. Copy Word Count 32-bit parameters from the old stack to the new stack.

5. Push Return address on stack.

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

Interrupt gates and trap gates work like 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 multitasking/multiuser operating system is the ability to rapidly switch between tasks or processes. The Am486DX/DX2 microprocessor directly supports this operation by providing a task switch instruction in hardware. The Am486DX/DX2 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, all 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 that refers to a TSS, or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt can 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 4-16) containing the entire Am486DX/DX2 microprocessor execution state while a task gate descriptor contains a TSS selector. The Am486DX/DX2 CPU supports both 80286- and Am486DX/DX2 microprocessor-style TSSs. Figure 4-17 shows an 80286 TSS. The limit of an Am486DX/DX2 CPU TSS must be greater than 0064H (002BH for a 80286 TSS), and

can be as large as 4 Gbytes. 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 be associated with TSS. The current TSS is identified by a special register in the Am486DX/DX2 microprocessor, 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 that 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 that is 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 as shown in Figure 4-17.

When a CALL or INT instruction initiates a task switch, the new TSS is marked busy and the back link field of the new TSS is 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

Figure 4-17   80286 TSS

15	0		
Back Link Selector to TSS	0		
SP for CPL 0	2		
SS for CPL 0	4		Initial
SP for CPL 1	6		Stacks
SS for CPL 1	8		for CPL 0, 1, 2
SP for CPL 2	A		
SS for CPL 2	C		
IP (Entry Point)	E		
Flags	10		
AX	12		
CX	14		
DX	16		
BX	18		Current
SP	1A		Task State
BP	1C		
SI	1E		
DI	20		
ES Selector	22		
CS Selector	24		
SS Selector	26		
DS Selector	28		
Task's LDT Selector	2A		
Available			
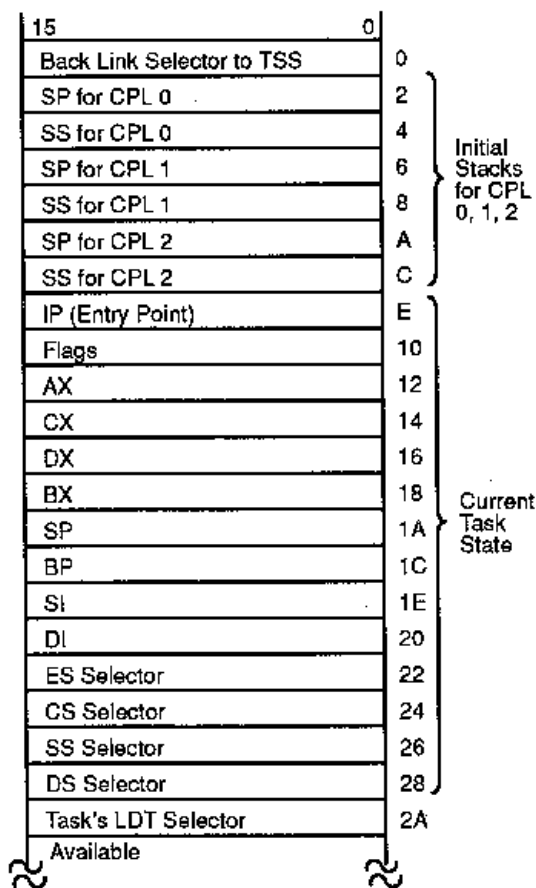
17852A–038

switch clears NT. (The NT bit is restored after execution of the interrupt handler) NT can also be set or cleared by POPF or IRET instructions.

The Am486DX/DX2 microprocessor TSS 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. Using a selector that references a busy TSS 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 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 Am486DX/DX2 microprocessor switches tasks it sets the TS Bit. The Am486DX/DX2 CPU detects the first use of a processor extension instruction after a task switch and causes the processor extension not present exception 7. The exception handler for exception 7 can then decide whether to save the state of the FPU. A processor extension not present (exception 7) occurs 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 Am486DX/DX2 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 is generated.

4.4.7 Initialization and Transition to Protected Mode

Since the Am486DX/DX2 microprocessor begins executing in Real Mode immediately after RESET, initializing the system tables and registers with the appropriate values is necessary.

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 4-18 shows the tables and Figure 4-19 shows the descriptors needed for a simple Protected Mode Am486DX/DX2 CPU system. It has a single code and single data/stack segment (each 4 Gbytes 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 Am486DX/DX2 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 the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode is especially appropriate for multitasking operating systems. This method uses the built-in task switch to load all the registers. In this case, the GDT contains two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode jumps to the TSS, causing a task switch and loading all of the registers with the values stored in the TSS. The TSS Register should be initialized to point to a valid TSS descriptor since a task switch saves the current task state in a TSS.

Figure 4-18 Simple Protected System
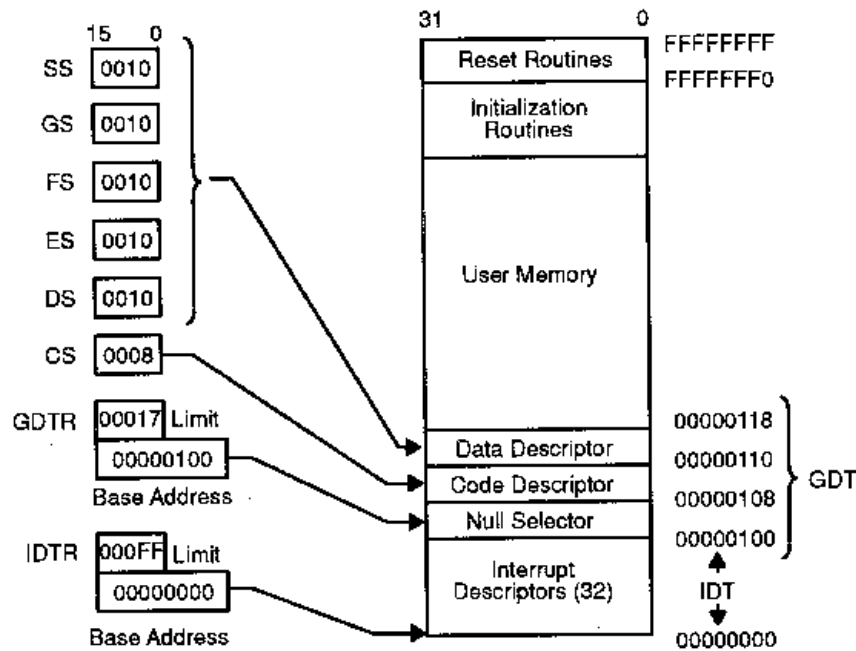
Figure 4-19 GDT Descriptors for Simple System

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 must be in memory at any one moment.

4.5.2 Paging Organization

4.5.2.1 Page Mechanism

The Am486DX/DX2 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 Am486DX/DX2 microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Am486DX/DX2 microprocessor paging mechanism are the same size, 4 Kbytes. A uniform size for all the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4-20 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 that 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 does a task switch through a TSS that changes the value of CR0. (See Section 4.5.5, Translation Lookaside Buffer).

Figure 4-20 Paging Mechanism

Two Level Paging Scheme

17852A–041
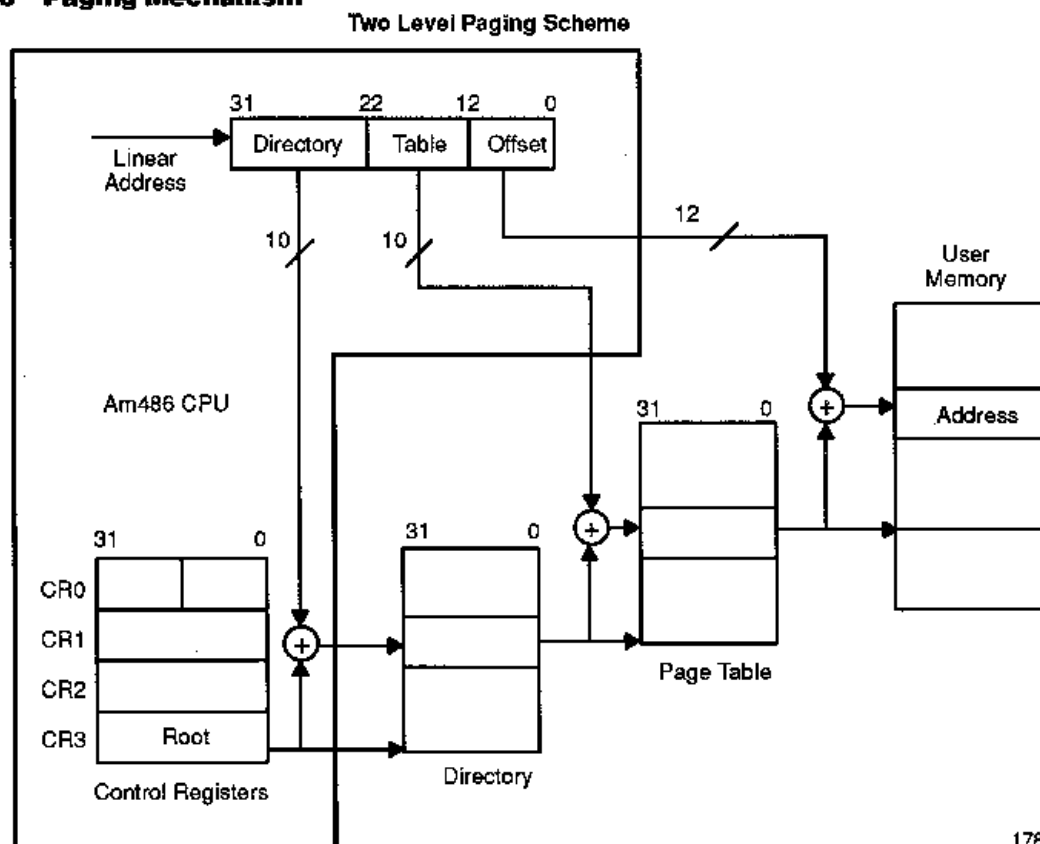
4.5.2.3 Page Directory

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 4-21. The upper 10 bits of the linear address (A31–A22) are used as an index to select the correct page directory entry. The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction.

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 4-22). Address bits A21–A12 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 cannot be used for translation and all 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 Am486DX/DX2 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 Am486DX/DX2 microprocessor, the processor generates a read-modify-write cycle that locks the bus and prevents conflicts with other processors or peripherals. Software that modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multimaster systems.

The three bits marked "OS Reserved" in Figure 4-21 and Figure 4-22 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. One use of the OS Reserved bits could be storing information about page aging. By keeping

Figure 4-21   Page Directory Entry (Points to Page Table)

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	P C D	P W T	U – S	R – W	P
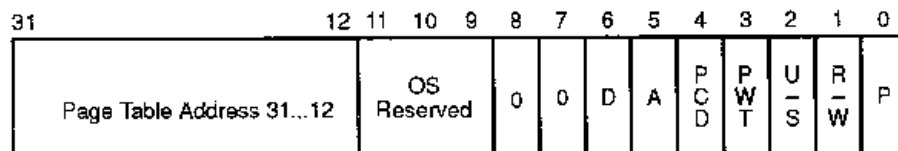
17852A–042

Figure 4-22   Page Table Entry (Points to Page)

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	P C D	P W T	U – S	R – W	P
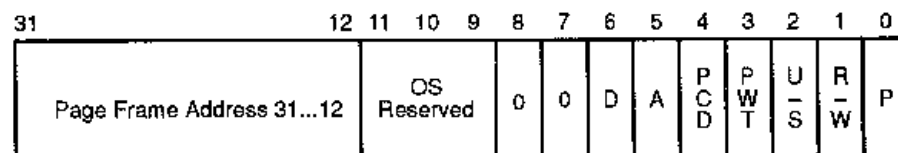
17852A–043

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 Am486DX/DX2 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, and 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The 386 microprocessor does not contain the WP bit. The WP bit has been added to the Am486DX/DX2 microprocessor to protect read-only pages from supervisor write accesses. The 386 microprocessor allows a read-only page to be written from protection levels 0, 1, or 2. WP = 0 is the 386 microprocessor-compatible mode. When WP = 0, the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP = 1, supervisor access to a read-only page (R/W = 0) causes 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 causes 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 are 10 (user read/execute) and the U/S and R/W bits for the page table entry are 01 (no user access at all), the access rights for the page is 01, the numerically smaller of the two.

Note: A given segment can be easily made read-only for level 0, 1, or 2 via use of segmented protection mechanisms. (See 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 Am486DX/DX2 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 $\overline{KEN}$ (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 $\overline{KEN}$, CD, and NW. (See Chapter 5, On-Chip Cache).

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

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 that 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 Am486DX/DX2 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 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 Am486DX/DX2 microprocessor paging hardware is designed to support demand-paged virtual memory systems. However, performance would degrade substantially if the processor is required to access two levels of tables for every memory reference. To solve this problem, the Am486DX/DX2 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 multitasking systems, the TLB has a hit rate of about 98%. This means the processor only has to access the two-level page structure on 2% of all memory references. Figure 4-23 illustrates how the TLB complements the Am486DX/DX2 microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the Am486DX/DX2 microprocessor hardware. The sequence of data cycles to perform a TLB refresh are

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 CR3.

Figure 4-23   Translation Lookaside Buffer

● 98% Hit Rate   17852A-044
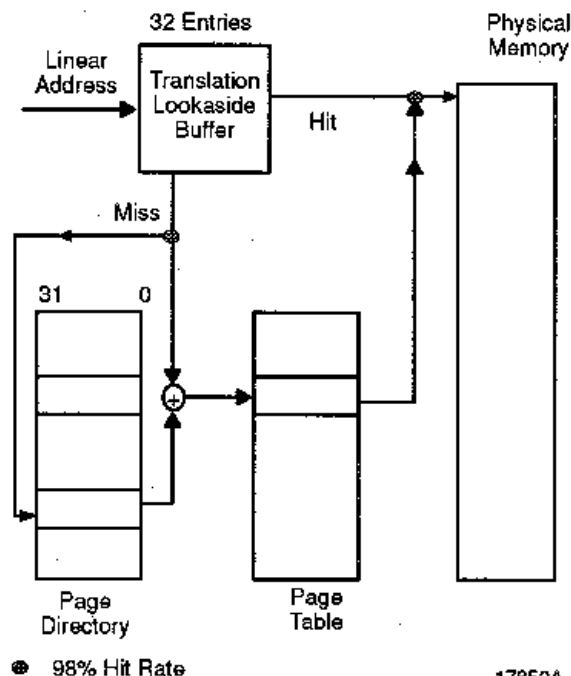
1a. Optionally, perform a locked read/write to set the accessed bit in the directory entry. The directory entry actually gets read twice if the Am486DX/DX2 microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned 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 actually gets read twice if the Am486DX/DX2 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 is used.

Note: 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 can be placed in the Am486DX/DX2 on-chip cache just like normal data.

4.5.6   Paging Operation

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 is placed on the address bus.

However, if the page table entry is not in the TLB, the Am486DX/DX2 microprocessor reads the appropriate page directory entry. If P = 1 on the page directory entry indicating that the page table is in memory, then the Am486DX/DX2 microprocessor reads the appropriate page table entry and sets the Access bit. If P = 1 on the page table entry indicating that the page is in memory, the Am486DX/DX2 microprocessor updates the Access and Dirty bits as needed and fetches the operand. The upper 20 bits of the linear address, read from the page table, are 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 generates a page fault, an exception 14.

The processor also generates 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 holds the linear address that 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 invokes 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 points to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler contains status bits that 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. Figure 4-24 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. Table 4-5 indicates what type of access caused the page fault.

Figure 4-24   Page Fault Error Code Format
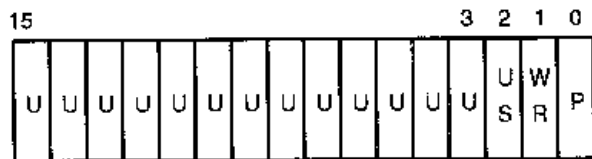
17852A-045

Table 4-5   Type of Access Causing Page Fault

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

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

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

4.5.7   Operating System Responsibilities

The Am486DX/DX2 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 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 0, the TLB must be flushed. Operating systems may want to take advantage of the fact the 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 Am486DX/DX2 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 Am486DX/DX2 microprocessor protection mechanism. In particular, the Am486DX/DX2 microprocessor allows the simultaneous execution of 8086 operating systems and its applications, as well as an Am486DX/DX2 microprocessor operating system and both 80286 and Am486DX/DX2 microprocessor applications. Thus, in a multiuser Am486DX/DX2 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 he had the computer completely to themselves. Figure 4-25 illustrates this concept.

4.6.2 Virtual 8086 Mode Addressing Mechanism

One of the major differences between the Am486DX/DX2 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 are shifting left four bits and are added to the offset to form the segment base linear address.

The Am486DX/DX2 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 1-Mbyte address space of the Virtual Mode task can be mapped to anywhere in the 4 Gbyte linear address space of the Am486DX/DX2 microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 K byte cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode are simply 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 1 Mbyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each page can be located anywhere within the maximum 4 Gbyte physical address space of the Am486DX/DX2 microprocessor. In

Figure 4-25   Virtual 8086 Environment Memory Management
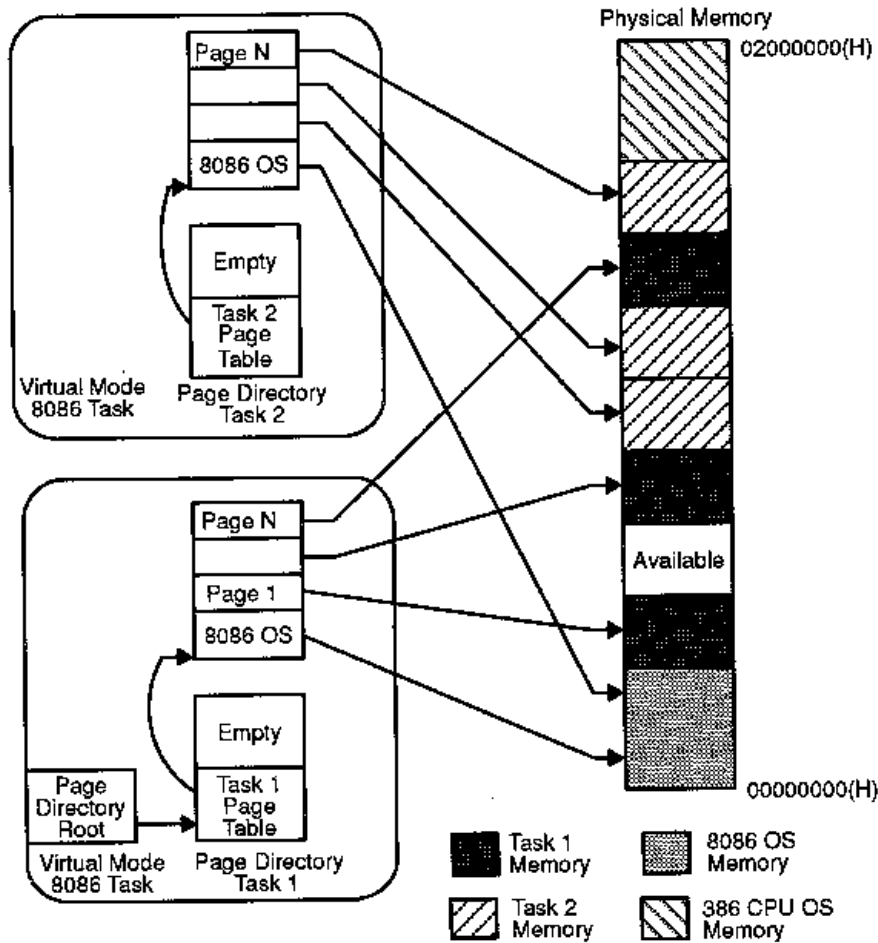
addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4-25 shows how the Am486DX/DX2 microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4   Protection and I/O Permission Bit-map

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 causes 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 that are IOPL-sensitive in Protected Mode are

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

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 are not 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 Bit-map contained in the Am486DX/DX2 microprocessor Task State Segment. The I/O Permission Bit-map, automatically used by the Am486DX/DX2 microprocessor in Virtual 8086 Mode, is illustrated by Figure 4-15 and Figure 4-16.

The I/O Permission Bit-map 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 that 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 4-16.

Each bit in the I/O Permission Bit-map corresponds to a single byte-wide I/O port, as illustrated in Figure 4-16. 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 is allowed. If any referenced bits are 1, the attempted I/O causes an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bit-map, the bit-map can be located anywhere within the TSS, or can 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 needs to have an associated map bit, by adjusting the TSS limit to truncate the bit-map. This eliminates the commitment of 8K of memory when a complete bit-map 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) allows a 32-byte bit-map for the I/O ports #0–255, plus a terminator byte of all 1's (** see note below). This allows the I/O bit-map to control I/O Permission to I/O ports 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 256 through 65,565.

Note: ** Beyond the last byte of I/O mapping information in the I/O Permission Bit-map must be a byte containing all 1's. The byte of all 1's must be within the limit of the Am486DX/DX2 microprocessor TSS segment (see Figure 4-15).

4.6.5   Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled uniquely. When running in Virtual Mode, all interrupts and exceptions involve a privilege change back to the host Am486DX/DX2 microprocessor operating system. The Am486DX/DX2 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 Am486DX/DX2 microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Am486DX/DX2 microprocessor operating system can 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 are intercepted by the Am486DX/DX2 microprocessor operating system. The Am486DX/DX2 microprocessor operating system can emulate the 8086 operating system's call. Figure 4-26 shows how the Am486DX/DX2 microprocessor operating system can intercept an 8086 operating system's call to "Open a File".

An Am486DX/DX2 microprocessor operating system can provide a Virtual 8086 environment that 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 an Am486DX/DX2 microprocessor task. The Am486DX/DX2 microprocessor TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 Mode is to switch to a task with an Am486DX/DX2 microprocessor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the

processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in Real Mode, or in Virtual 8086 Mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt that causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image); the VM bit 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 do not change the value in the VM bit.

The transition out of Virtual 8086 Mode to Am486DX/DX2 microprocessor Protected Mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 Mode, all interrupts and exceptions vector through the Protected Mode IDT and enter an interrupt handler in protected Am486DX/DX2 microprocessor mode. That is, as part of interrupt processing, the VM bit is cleared.

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

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks that can execute in Virtual 8086 Mode must be described by a TSS with the new Am486DX/DX2 microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of Virtual 8086 Mode operates exactly the same as any other task switch out of a task with an Am486DX/DX2 microprocessor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS contain 8086 segment base values rather than selectors.

A task switch into a task described by an Am486DX/DX2 microprocessor TSS has 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 an Am486DX/DX2 microprocessor TSS, the FLAGS image is loaded so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in Virtual 8086 Execution Mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit Virtual 8086 Mode. The other method is to exit through a trap or interrupt gate as part of handling an interrupt; and to enter as part of executing an IRET instruction. The transition out must use an Am486DX/DX2 microprocessor trap gate (TYPE 14), or Am486DX/DX2 microprocessor interrupt gate (TYPE 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. Am486DX/DX2 microprocessor gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit is not saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler pops only the lower 16 bits from FLAGS and does not affect the VM bit. The action taken for an Am486DX/DX2 microprocessor trap or interrupt gate, if an interrupt occurs while the task is executing in Virtual 8086 Mode, is given by the following sequence.

Figure 4-26   Virtual 8086 Environment Interrupt and Call Handling
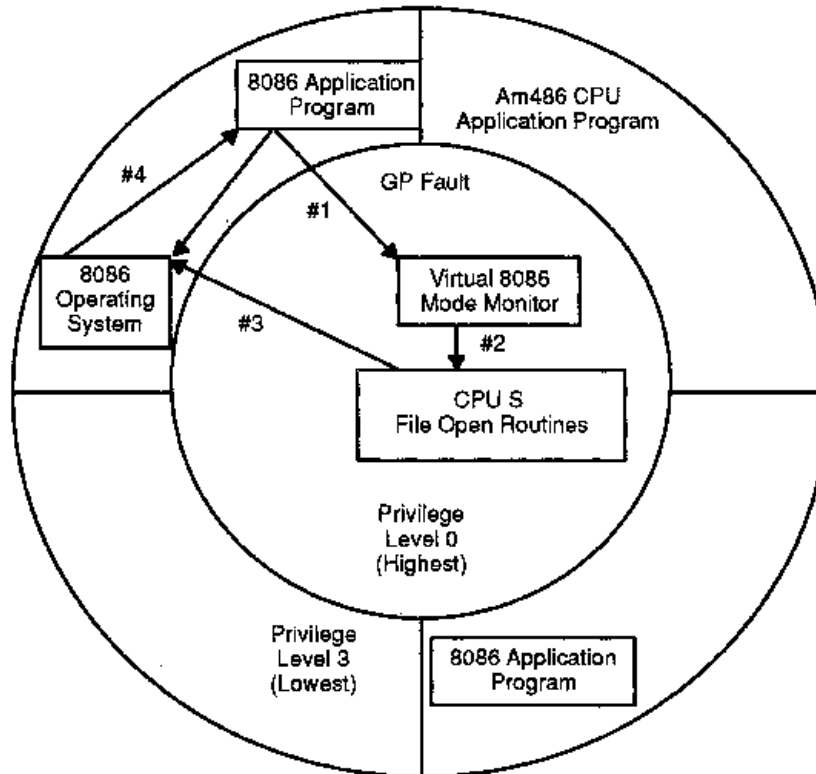
Notes:
8086 Application makes "Open File Call" → causes General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls Am486 CPU OS (Arrow #2)
Am486 CPU OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application (Arrow #4)
Transparent to Application

17852A–047

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 Virtual Mode 8086 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 is done as a Protected Mode segment load since the VM bit was turned off above.

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 four 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 Am486DX/DX2 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 permits the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers that do not care about the mode of the interrupted program can use the same prologue and epilogue code for state saving (i.e., push all registers in prologue pop all in epilogue) regardless of whether or not a "native" mode or Virtual 8086 Mode program is interrupted. Restoring null selectors to these registers before executing the IRET does not cause a trap in the interrupt handler. Interrupt routines that expect values in the segment registers, or return values in segment registers, have to obtain/return values from the 8086 register images pushed onto the new stack. They 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 performs the inverse of the above sequence. Only the extended Am486DX/DX2 microprocessor's 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 intertask 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 that is to be resumed. Otherwise, continue with the following sequence.

2. Read the FLAGS image from SS:[ESP] into the FLAGS register. This sets 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 that 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 is done as an 8086 segment load.

4. Increment the ESP register by 4 to bypass the FLAGS image that 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:[EXP + 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 any 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 or Virtual 8086 Mode.

5   ON-CHIP CACHE

To meet its performance goals, the Am486DX/DX2 microprocessor contains an 8-Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the Am386 CPU architectures.

The on-chip cache is 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 address.

The cache organization is four-way set associative and each line is 26-bytes wide. The 8-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.

The write strategy of on-chip cache is write-through. All writes 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 is only written to external memory. Cache allocations are not made on write misses.

Figure 5-1   On-Chip Cache Physical Organization
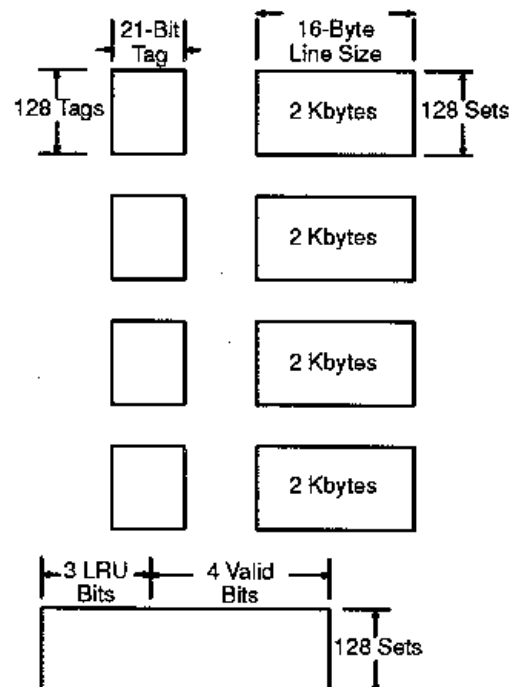
17852A–048

5.2   CACHE CONTROL

Cache control is provided by the CD and NW bits in Control Register 0 (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 (see Table 5-1). These modes provide flexible on-chip cache operation.

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode can be useful for debugging programs where it is important to see all memory cycles at the pins. Writes that hit in the cache do 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 carefully choosing memory references with the cache turned on, or by using 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 $\overline{\text{KEN}}$ pin was strapped High, disabling cache fills. Write-throughs and invalidates can 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 an error code of 0 is raised. Note that this mode implies a non-transparent write-back cache. A future processor might 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 still occur and data is read from the cache.

5.3   CACHE LINE FILLS

Any area of memory can be cached in the Am486DX/DX2 microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the Am486DX/DX2 microprocessor that a memory address is non-cacheable by returning the $\overline{\text{KEN}}$ 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.

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

A read request can be generated from program operation or by an instruction prefetch. The data is 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 Am486DX/DX2 microprocessor initiates a cache line fill. During a line fill, a 16-byte line is read into the Am486DX/DX2 microprocessor.

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

Cache line fills can be performed over 8- and 16-bit buses 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 Am486DX/DX2 microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the Am486DX/DX2 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 Am486DX/DX2 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 (LRU) mechanism is used to determine which line is 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 valid bits are checked during an invalidation in the order 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 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 the LRU bits and 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 first selects which of I0:I1 and I2:I3 was least recently used. Then the cache determines which of the two lines was least recently used and marks 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.

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.

Figure 5-2   On-Chip Cache Replacement Strategy

All four lines in the set valid? →**No**→ Replace non-valid line

Yes ↓

B0 = 0?

Yes: I0 or I1 least recently used   No: I2 or I3 least recently used

B1 = 0?   B2 = 0?

Yes   No   Yes   No

Replace   Replace   Replace   Replace
I0   I1   I2   I3
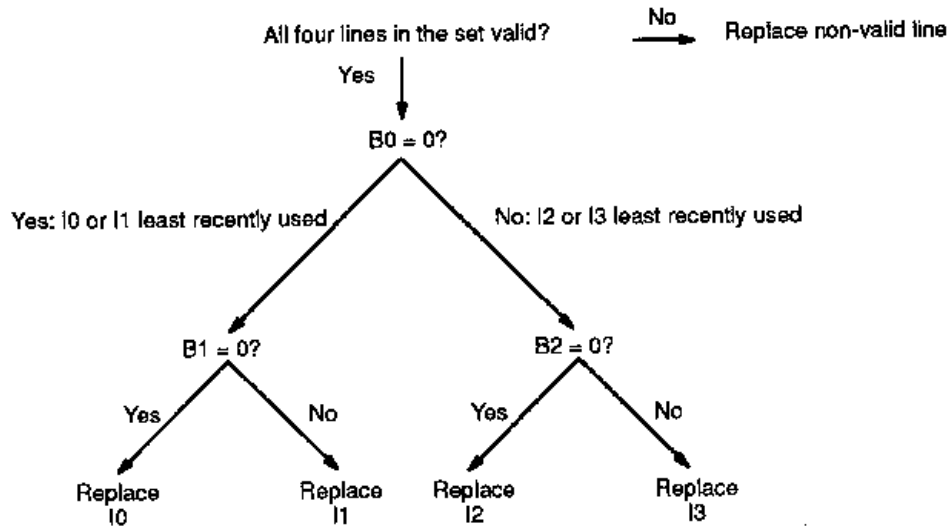
17852A—049

The PWT bit controls write policy for second level caches used with the Am486DX/DX2 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 Am486DX/DX2 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 $\overline{KEN}$ 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 $\overline{KEN}$ = 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 $\overline{KEN}$, telling the Am486DX/DX2 microprocessor if the area is cacheable. The Am486DX/DX2 microprocessor initiates a cache line fill if PCD and $\overline{KEN}$ indicate that the requested information is cacheable.

The PCD bit is masked with the CD bit in control register 0 to determine the state of the PCD pin. If CD = 1, the Am486DX/DX2 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 entry, or page table entry. These bits are assumed to be 0 during Real Mode, whenever paging is disabled, or for cycles that bypass paging (I/O references, interrupt acknowledge, and Halt cycles).

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.

Figure 5-3   Page Cacheability
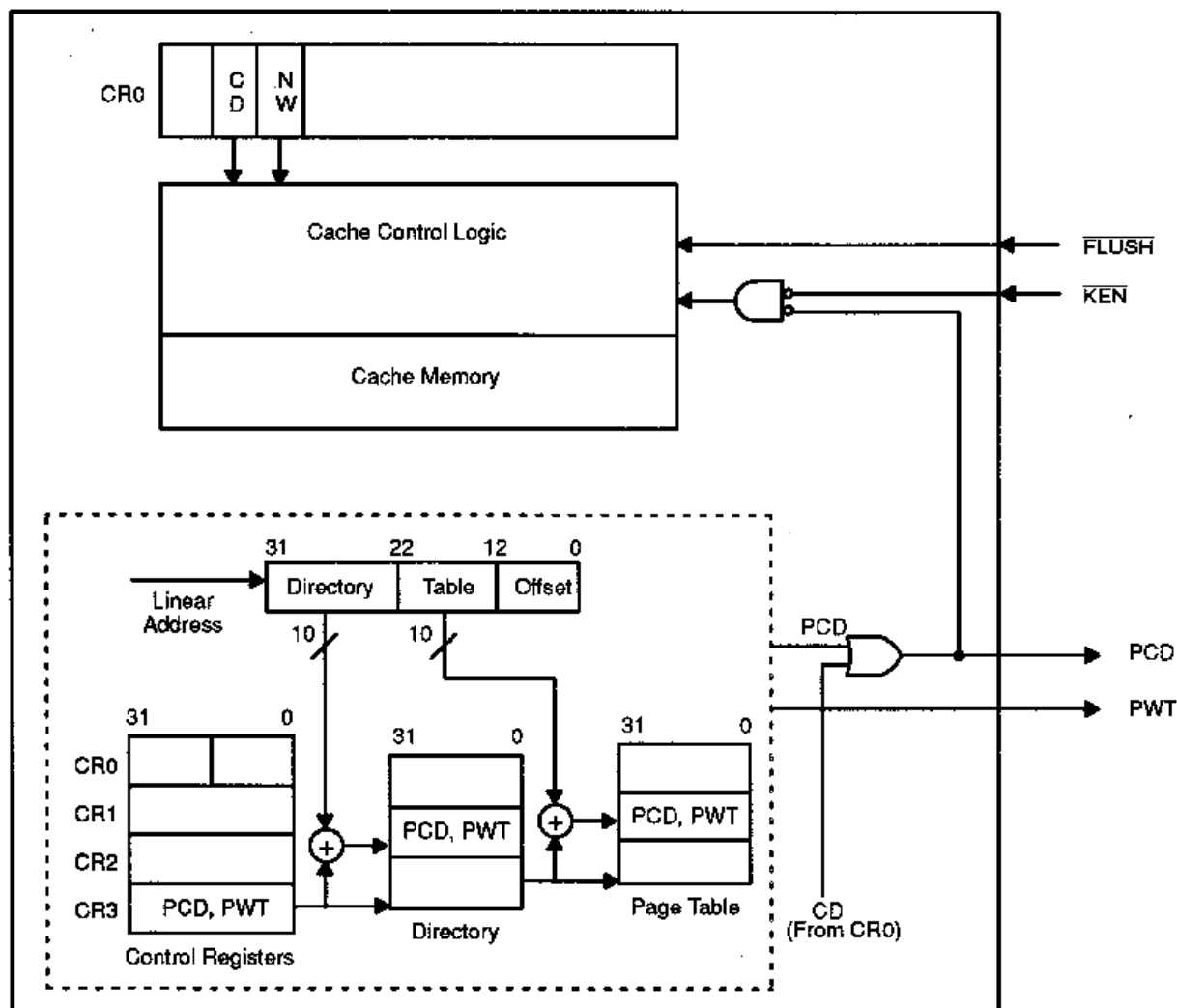
17852A–050

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 $\overline{FLUSH}$ 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 causes execution to stop as the processor repeatedly flushes 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 $\overline{EADS}$.

The instructions INVD and WBINVD cause the on-chip cache to be flushed. External caches connected to the Am486DX/DX2 microprocessor are signaled to flush their contents when these instructions are executed.

WBINVD causes an external write-back cache to write back dirty lines before flushing its contents. The external cache is signaled 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).

The results of the INVD and WBINVD instructions are identical for the operation of the Am486DX/DX2 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 Am486DX/DX2 microprocessor family might change the definition of this instruction.

5.8 CACHING TRANSLATION LOOKASIDE BUFFER ENTRIES

The Am486DX/DX2 microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the 386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU 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 looks 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 accesses the page tables in external memory if the linear address is not in the TLB. The required page table entry is read into the TLB and then the cache or bus cycle for the actual data takes 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 that 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 Am486DX/DX2 microprocessor allows 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 prevents the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, Page Cacheability).

6 HARDWARE INTERFACE

6.1 INTRODUCTION

The Am486DX/DX2 microprocessor bus is designed similar to the 386 microprocessor bus whenever possible. Several new features have been added to the Am486DX/DX2 microprocessor bus, resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a $\overline{BS8}$ bus sizing mechanism, and parity support.

The Am486DX/DX2 microprocessor is driven by a 1X clock as opposed to a 2X clock in the 386 microprocessor. A 25-MHz Am486DX/DX2 microprocessor uses a 25-MHz clock in contrast to a 25-MHz 386 microprocessor that requires a 50-MHz clock. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the 386 microprocessor, the Am486DX/DX2 microprocessor has separate parallel buses for data and addresses. The bidirectional data bus is 32-bits wide. The address bus consists of two components: 30 address lines (A31–A2) and four byte enable lines ($\overline{BE3}$–$\overline{BE0}$). The address bus addresses external memory in the same manner as the 386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4-byte location. The address lines are bidirectional for use in cache line invalidations.

The Am486DX/DX2 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 Am486DX/DX2 microprocessor has a bus hold feature similar to that of the 386 microprocessor. During bus hold, the Am486DX/DX2 microprocessor relinquishes control of the local bus by floating its address, data, and control buses.

The Am486DX/DX2 microprocessor has an address hold feature in addition to bus hold. During address hold, only the address bus is floated, the data and control buses can remain active. Address hold is used for cache line invalidations.

Section 6.2 has a brief description of the Am486DX/DX2 microprocessor input and output signals arranged by functional groups. Before beginning the signal descriptions, a few terms need to be defined. The overbarring of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When the signal name is not overbarred, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with $\overline{RDY}$ or $\overline{BRDY}$.

Chapters 6 and 7 discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with $\overline{ADS}$ active in the first clock and $\overline{RDY}$ active in the last clock. Data is transferred to or from the Am486DX/DX2 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 Am486DX/DX2 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

The Am486DX/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 two phases. CLK only needs TTL levels for proper operation. Figure 6-2 illustrates the CLK waveform.

Figure 6-1   Functional Signal Groupings
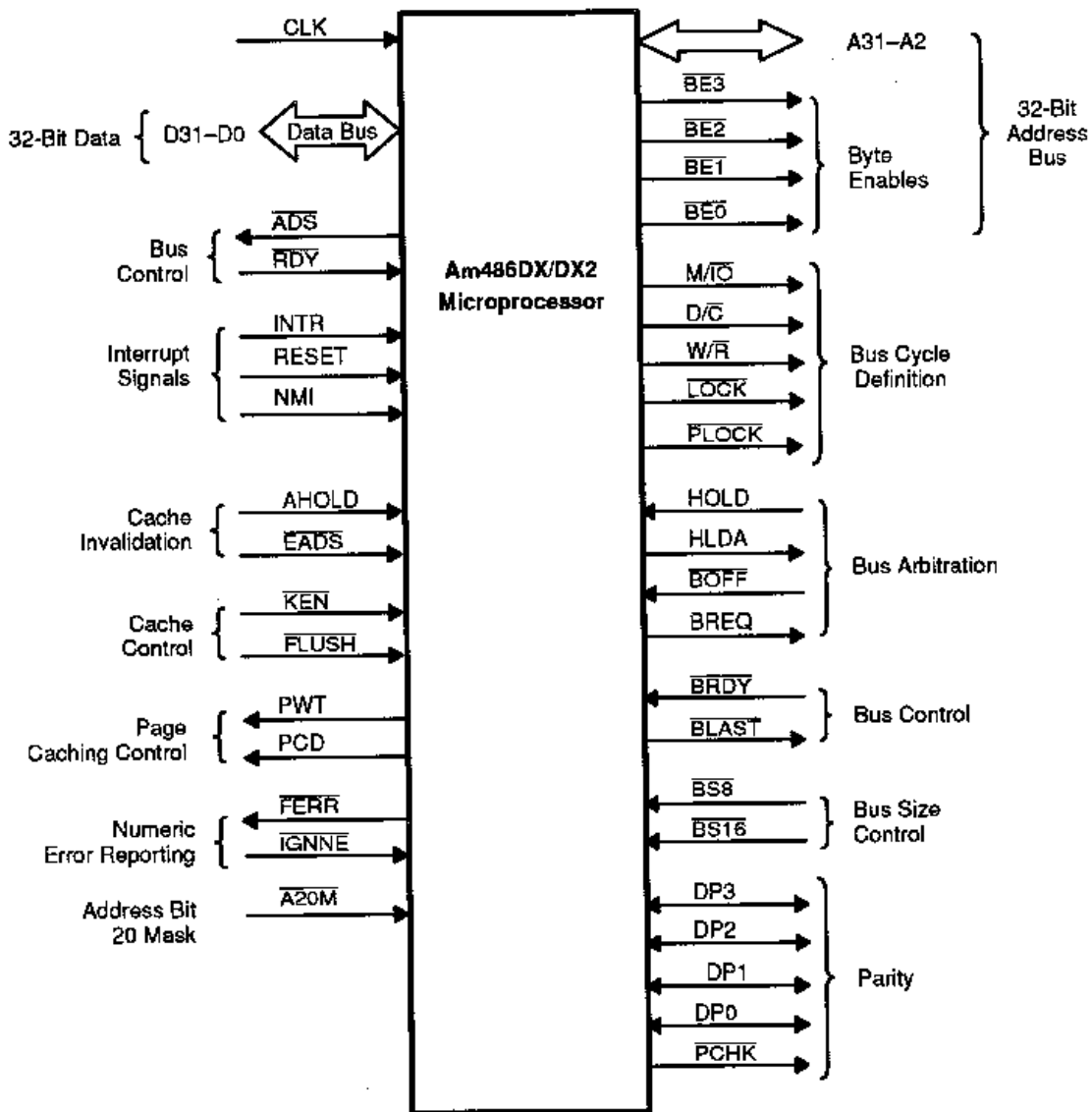
17852A—051

Figure 6-2   CLK Waveform

tx = input setup times
ty = input hold times, output float, valid and hold times
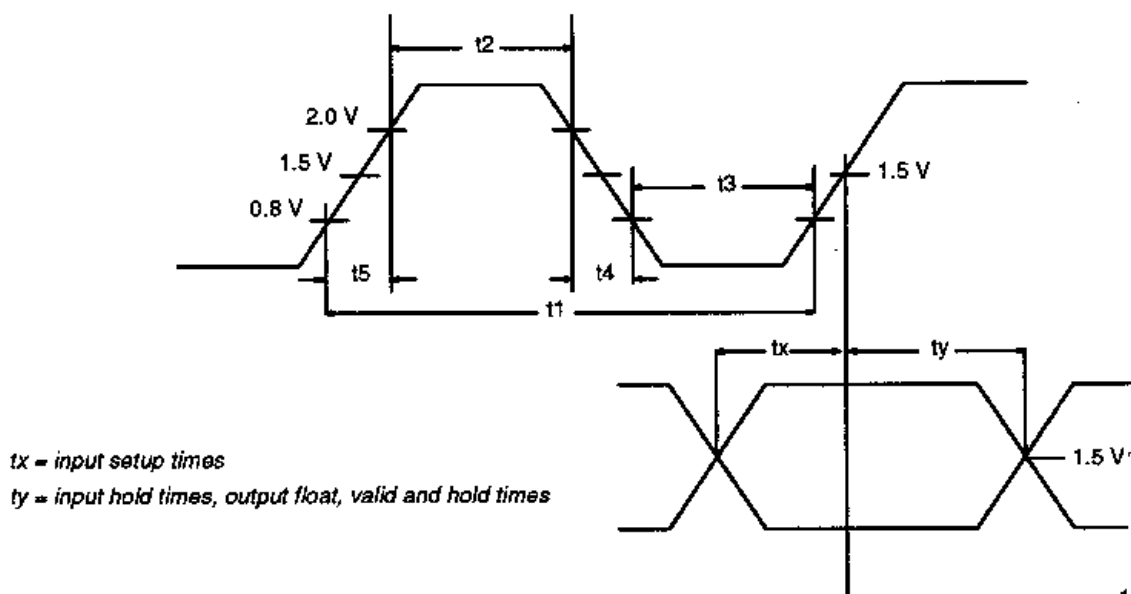
17852A–052

6.2.2   Address Bus (A31–A2, BE3–BE0)

A31–A2 and BE3–BE0 form the address bus and provide physical memory and I/O port addresses. The Am486DX/DX2 microprocessor is capable of addressing 4 Gbytes 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. BE3–BE0 identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the Am486DX/DX2 microprocessor over A31–A4 during cache line invalidations. The address lines are active High. When used as inputs into the processor, A31–A4 must meet the setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

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

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

BE3–BE0 can be decoded to generate A0, A1, and BHE signals used in 8- and 16-bit systems (see Table 7-5). BE3–BE0 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 Am486DX/DX2 microprocessor. D7–D0 define the least significant byte and D31–D24 the most significant byte. Data transfers to 8- or 16-bit devices are 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

6.2.4.1   Data Parity Input/Outputs (DP3–DP0)

DP3–DP0 are the data parity pins for the microprocessor. 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 there is 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 Am486DX/DX2 microprocessor. Even parity information must be driven back to the Am486DX/DX2 microprocessor on these pins with the same timing as read information. This ensures that the correct parity check status is indicated by the Am486DX/DX2 microprocessor.

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

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

6.2.4.2   Parity Status Output ($\overline{PCHK}$)

Parity status is driven on the $\overline{PCHK}$ pin and a parity error is indicated by this pin being Low. $\overline{PCHK}$ 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. $\overline{PCHK}$ 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 Am486DX/DX2 microprocessor. At all other times it is inactive High. $\overline{PCHK}$ is never floated.

Driving $\overline{PCHK}$ is the only effect bad input parity has on the Am486DX/DX2 microprocessor. The Am486DX/DX2 microprocessor does not vector to a bus error interrupt when bad data parity is returned. In systems that do not employ parity, $\overline{PCHK}$ can be ignored. In systems not using parity, DP3–DP0 should be connected to V_{cc} through a pull-up resistor.

6.2.5   Bus Cycle Definition

6.2.5.1   M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$ Outputs

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

Bus cycle definitions as a function of M/$\overline{IO}$, D/$\overline{C}$, and W/$\overline{R}$ are given in Table 6-1. Note there is a difference between the Am486DX/DX2 microprocessor and Am386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the Am486DX/DX2 microprocessor from location 101 in the Am386 microprocessor. Location 101 is now reserved and will never be generated by the Am486DX/DX2 microprocessor.

Table 6-1 $\overline{\text{ADS}}$ **Initiated Bus Cycle Definitions**

$\overline{\text{M/IO}}$	$\overline{\text{D/C}}$	$\overline{\text{W/R}}$	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

6.2.5.2   **Bus Lock Output ($\overline{\text{LOCK}}$)**

$\overline{\text{LOCK}}$ indicates that the Am486DX/DX2 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 $\overline{\text{LOCK}}$ is asserted, the current bus cycle is locked and the Am486DX/DX2 microprocessor should be allowed exclusive access to the system bus. $\overline{\text{LOCK}}$ 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 Am486DX/DX2 microprocessor does not acknowledge bus hold when $\overline{\text{LOCK}}$ is asserted (though it does allow an address hold). $\overline{\text{LOCK}}$ is active Low and is floated during bus hold. Locked read cycles are not transformed into cache fill cycles if $\overline{\text{KEN}}$ is returned active. Refer to Section 7.2.6 for a detailed discussion of locked bus cycles.

6.2.5.3   **Pseudo-Lock Output ($\overline{\text{PLOCK}}$)**

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 Am486DX/DX2 microprocessor asserts $\overline{\text{PLOCK}}$ during floating-point long reads and writes (64 bits), segment table descriptor reads (64 bits), and cache line fills (128 bits).

When $\overline{\text{PLOCK}}$ is asserted no other master is 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 $\overline{\text{PLOCK}}$ is asserted. The Am486DX/DX2 microprocessor drives $\overline{\text{PLOCK}}$ active until the addresses for the last bus cycle of the transaction have been driven, regardless of whether $\overline{\text{BRDY}}$ or $\overline{\text{RDY}}$ is 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 $\overline{\text{PLOCK}}$ and $\overline{\text{BLAST}}$ are inverse of each other. However, during the first cycle of a 64-bit floating-point write, both $\overline{\text{PLOCK}}$ and $\overline{\text{BLAST}}$ are asserted.

Since $\overline{\text{PLOCK}}$ is a function of the bus size and $\overline{\text{KEN}}$ inputs, $\overline{\text{PLOCK}}$ should be sampled only if 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.

6.2.6.1 Address Status Output ($\overline{ADS}$)

The $\overline{ADS}$ output indicates that the address and bus cycle definition signals are valid. This signal goes active in the first clock of a bus cycle and goes inactive in the second and subsequent clocks of the cycle. $\overline{ADS}$ is also inactive when the bus is idle.

$\overline{ADS}$ 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 $\overline{ADS}$ is driven active.

$\overline{ADS}$ is active Low and is not driven during bus hold.

6.2.6.2 Non-Burst Ready Input ($\overline{RDY}$)

$\overline{RDY}$ indicates that the current bus cycle is complete. In response to a read, $\overline{RDY}$ indicates that the external system has presented valid data on the data pins. In response to a write request, $\overline{RDY}$ indicates that the external system has accepted the Am486DX/DX2 microprocessor data. $\overline{RDY}$ is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since $\overline{RDY}$ is sampled during address hold, data can be returned to the processor when AHOLD is active.

$\overline{RDY}$ is active Low and is not provided with an internal pull-up resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 Burst Control

6.2.7.1 Burst Ready Input ($\overline{BRDY}$)

$\overline{BRDY}$ performs the same function during a burst cycle that $\overline{RDY}$ performs during a non-burst cycle. $\overline{BRDY}$ 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 Am486DX/DX2 microprocessor data in response to a write. $\overline{BRDY}$ is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, $\overline{BRDY}$ is sampled each clock, and if active, the data presented on the data bus pins is stored into the Am486DX/DX2 microprocessor. $\overline{ADS}$ is negated during the second through last data cycles in the burst, but address lines A3–A2 and byte enables change to reflect the next data item expected by the Am486DX/DX2 microprocessor.

If $\overline{RDY}$ is returned simultaneously with $\overline{BRDY}$, $\overline{BRDY}$ is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle is initiated after an aborted burst cycle if the cache line fill was not complete. $\overline{BRDY}$ 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.

$\overline{BRDY}$ is active Low and is provided with a small internal pull-up resistor. $\overline{BRDY}$ must satisfy the setup and hold times t_{16} and t_{17}.

6.2.7.2 Burst Last Output ($\overline{BLAST}$)

$\overline{BLAST}$ indicates that the next time $\overline{BRDY}$ is returned it is treated as a normal $\overline{RDY}$, terminating the line fill or other multiple-data-cycle transfer. $\overline{BLAST}$ 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.

6.2.8.1 Reset Input (RESET)

RESET forces the Am486DX/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 Am486DX/DX2 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper Am486DX/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 Chapter 8 for a description of the test modes during reset.)

6.2.8.2 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 Am486DX/DX2 microprocessor generates two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number is 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 ensure 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 pull-down resistor. INTR is asynchronous, but the INTR setup and hold times t_{20} and t_{21} must be met to ensure recognition on any specific clock.

6.2.8.3 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 is masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. For proper operation, NMI must be held Low for at least four CLK periods before this rising edge. NMI is not provided with an internal pull-down resistor. NMI is asynchronous, but setup and hold times t_{20} and t_{21} must be met to ensure 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.

6.2.9.1 Bus Request Output (BREQ)

The Am486DX/DX2 microprocessor asserts $\overline{BREQ}$ whenever a bus cycle is pending internally. Thus, $\overline{BREQ}$ is always asserted in the first clock of a bus cycle, along with $\overline{ADS}$. Furthermore, if the Am486DX/DX2 microprocessor is currently not driving the bus (due to HOLD, AHOLD, or $\overline{BOFF}$), $\overline{BREQ}$ is asserted in the same clock that $\overline{ADS}$ would have been asserted if the processor was driving the bus.

After the first clock of the bus cycle, $\overline{BREQ}$ can change state. It will be asserted if additional cycles are necessary to complete a transfer (via $\overline{BS8}$, $\overline{BS16}$, $\overline{KEN}$), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, $\overline{BREQ}$ can be negated before ready comes back for the current cycle. External logic can use the $\overline{BREQ}$ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. $\overline{BREQ}$ is active High and is never floated. During a hold state, internal events may cause $\overline{BREQ}$ to be deasserted prior to any bus cycles.

6.2.9.2 Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the Am486DX/DX2 microprocessor bus. The Am486DX/DX2 microprocessor responds 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 Am486DX/DX2 CPU receives a HOLD request while performing a non-cacheable, non-bursted code prefetch and that cycle is backed off ($\overline{BOFF}$), the Am486DX/DX2 CPU recognizes HOLD before restarting the cycle. The $\overline{BREQ}$, HLDA, and $\overline{PCHK}$ pins are not floated during bus hold. The Am486DX/DX2 microprocessor maintains its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for a bus hold cycle.

Unlike the 386 microprocessor, the Am486DX/DX2 microprocessor recognizes HOLD during reset. Pull-up 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.

6.2.9.3 Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the Am486DX/DX2 microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the Am486DX/DX2 microprocessor floats its bus.

HLDA is driven inactive when leaving bus hold and the Am486DX/DX2 microprocessor resumes driving the bus. The Am486DX/DX2 microprocessor does not cease internal activity during bus hold since the internal cache satisfies the majority of bus requests. HLDA is active High and remains driven during bus hold.

6.2.9.4 Backoff Input ($\overline{BOFF}$)

Asserting the $\overline{BOFF}$ input forces the Am486DX/DX2 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 $\overline{BOFF}$ differs from the response to HOLD in two ways: First, the bus is floated immediately in response to $\overline{BOFF}$ while the Am486DX/DX2 microprocessor completes the current bus cycle, before floating its bus in response to HOLD. Second, the Am486DX/DX2 microprocessor does not assert HLDA in response to $\overline{BOFF}$.

The processor remains in bus hold until $\overline{BOFF}$ is negated. Upon negation, the Am486DX/DX2 microprocessor restarts the bus cycle aborted when $\overline{BOFF}$ was asserted. To the internal execution engine, the effect of $\overline{BOFF}$ 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 $\overline{BOFF}$ is asserted is ignored. $\overline{BOFF}$ has higher priority than $\overline{RDY}$ or $\overline{BRDY}$. If both $\overline{BOFF}$ and ready are returned in the same clock, $\overline{BOFF}$ takes effect. If $\overline{BOFF}$ is asserted while the bus is idle, the Am486DX/DX2

microprocessor floats its bus in the next clock. $\overline{BOFF}$ 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 $\overline{EADS}$ inputs are used during cache invalidation cycles. AHOLD conditions the Am486DX/DX2 microprocessor's address lines, A31–A4, to accept an address input. $\overline{EADS}$ indicates that an external address is actually valid on the address inputs. Activating $\overline{EADS}$ causes the Am486DX/DX2 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.

6.2.10.1 Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the Am486DX/DX2 microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD forces the Am486DX/DX2 microprocessor to stop driving its address bus in the next clock. While AHOLD is active, only the address bus is floated and 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 Am486DX/DX2 microprocessor does not initiate another bus cycle during address hold. Since the Am486DX/DX2 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 while AHOLD is asserted, the Am486DX/DX2 CPU redrives 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 are 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 Am486DX/DX2 microprocessor are exercised on assertion of RESET.

6.2.10.2 External Address Valid Input ($\overline{EADS}$)

$\overline{EADS}$ indicates that a valid external address has been driven onto the Am486DX/DX2 CPU address pins. This address is used to perform an internal cache invalidation cycle. The external address is checked with the current cache contents. If the specified address matches any areas in the cache, that area is immediately invalidated.

An invalidation cycle can be run by asserting $\overline{EADS}$, regardless of the state of AHOLD, HOLD, and $\overline{BOFF}$. $\overline{EADS}$ is active Low and is provided with an internal pull-up resistor. $\overline{EADS}$ must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 Cache Control

6.2.11.1 Cache Enable Input ($\overline{KEN}$)

$\overline{KEN}$ is the cache enable pin. $\overline{KEN}$ is used to determine whether the data being returned by the current cycle is cacheable. When $\overline{KEN}$ is active and the Am486DX/DX2 microprocessor generates a cacheable cycle (most any memory read cycle), the cycle is 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 Am486DX/DX2 microprocessor runs between 4 and 16 contiguous bus cycles to fill the line, depending on the bus data width selected by $\overline{BS8}$ and $\overline{BS16}$. Refer to Section 7.2.3 for a description of cache line fill cycles.

The $\overline{\text{KEN}}$ input is active Low and is provided with a small internal pull-up resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.11.2 Cache Flush Input (FLUSH)

The $\overline{\text{FLUSH}}$ input forces the Am486DX/DX2 microprocessor to flush its entire internal cache. $\overline{\text{FLUSH}}$ is active Low and need only be asserted for one clock. $\overline{\text{FLUSH}}$ is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

$\overline{\text{FLUSH}}$ also determines whether or not the three-state test mode of the Am486DX/DX2 microprocessor is 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 Am486DX/DX2 CPU ignores the PCD and PWT bits and assumes they are 0 for the purpose of caching and driving PCD and PWT.

PCD is masked by the CD bit in control register 0 (CR0). When CD = 1 (cache line fills disabled), the Am486DX/DX2 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 Am486DX/DX2 microprocessor does 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 0 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/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, and W/$\overline{\text{R}}$). 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 Am486DX/DX2 microprocessor provides two pins, $\overline{\text{FERR}}$ and $\overline{\text{IGNNE}}$.

6.2.13.1 Floating-Point Error Output (FERR)

The Am486DX/DX2 microprocessor asserts $\overline{\text{FERR}}$ whenever an unmasked floating-point error is encountered. $\overline{\text{FERR}}$ is similar to the $\overline{\text{ERROR}}$ pin on the 387 math coprocessor. $\overline{\text{FERR}}$ can be used by external logic for PC-type floating-point error reporting in Am486DX/DX2 microprocessor systems. $\overline{\text{FERR}}$ is active Low, and is not floated during bus hold.

In some cases, $\overline{\text{FERR}}$ 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 $\overline{\text{FERR}}$ 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 $\overline{FERR}$ 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).

6.2.13.2   Ignore Numeric Error Input ($\overline{IGNNE}$)

The Am486DX/DX2 microprocessor ignores a numeric error and continues executing non-control floating-point instructions when $\overline{IGNNE}$ is asserted; $\overline{FERR}$ is still activated. When deasserted, the Am486DX/DX2 microprocessor freezes on a non-control floating-point instruction if a previous instruction caused an error. $\overline{IGNNE}$ has no effect when the NE bit in control register 0 is set.

The $\overline{IGNNE}$ input is active Low and is provided with a small internal pull-up resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to ensure recognition on any specific clock.

6.2.14   Bus Size Control ($\overline{BS16}$, $\overline{BS8}$)

The $\overline{BS16}$ and $\overline{BS8}$ inputs allow external 16- and 8-bit buses to be supported with a small number of external components. The Am486DX/DX2 CPU samples these pins every clock. The value sampled in the clock before ready determines the bus size. When asserting $\overline{BS16}$ or $\overline{BS8}$, only 16 or 8 bits of the data bus need be valid. If both $\overline{BS16}$ and $\overline{BS8}$ are asserted, an 8-bit bus width is selected.

When $\overline{BS16}$ or $\overline{BS8}$ is asserted, the Am486DX/DX2 microprocessor converts a larger data request to the appropriate number of smaller transfers. The byte enables are also modified appropriately for the bus size selected.

$\overline{BS16}$ and $\overline{BS8}$ are active Low and are provided with small internal pull-up resistors. $\overline{BS16}$ and $\overline{BS8}$ must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15   Address Bit 20 Mask ($\overline{A20M}$)

Asserting the $\overline{A20M}$ input causes the Am486DX/DX2 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 $\overline{A20M}$ is asserted, the Am486DX/DX2 microprocessor emulates the 1-Mbyte address wraparound that occurs on the 8086. $\overline{A20M}$ is active Low and must be asserted only when the processor is in Real Mode. The $\overline{A20M}$ is not defined in Protected Mode. $\overline{A20M}$ 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, $\overline{A20M}$ should be sampled High two clocks before and two clocks after RESET goes Low.

When $\overline{A20M}$ is asserted synchronously, $\overline{A20M}$ should be High (non-active) at the clock prior to the falling edge of RESET. $\overline{A20M}$ exhibits a minimum 4 clock latency, from time of assertion to masking of the A20 bit. $\overline{A20M}$ is ignored during cache invalidation cycles. I/O writes require $\overline{A20M}$ to be asserted a minimum of 2 clocks prior to $\overline{RDY}$ being returned for the I/O write. This ensures recognition of the address mask before the Am486DX/DX2 microprocessor begins execution of the instruction following OUT. If $\overline{A20M}$ is asserted after the $\overline{ADS}$ of a data cycle, the A20 address signal is not masked during this cycle but is masked in the next cycle. During a prefetch (cacheable or not), if

$\overline{\text{A20M}}$ is asserted after the first $\overline{\text{ADS}}$, A20 is not masked for the duration of the prefetch; even if $\overline{\text{BS16}}$ or $\overline{\text{BS8}}$ is asserted.

6.2.16 Boundary Scan Test Signals

6.2.16.1 Test Clock (TCK)

TCK is an input to the Am486DX/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, 0, 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 an NC (this is important during power up to avoid the possibility of glitches on the TCK that could prematurely initiate boundary scan operations). TCK is supplied with an internal pull-up 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.

6.2.16.2 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.

6.2.16.3 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".

6.2.16.4 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 Am486DX/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 propagates 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 is placed in the write buffers and propagates 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 are driven onto the external bus in the same order they are received by the write buffers. Under certain conditions, a memory read goes 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 is only 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 Am486DX/DX2 microprocessor does 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). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

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 6-4).

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 one 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

Figure 6-3   Internal Cache Example
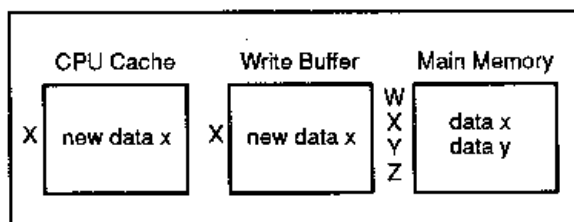
17852A–053

Figure 6-4   Internal Cache Example X No Longer Cached
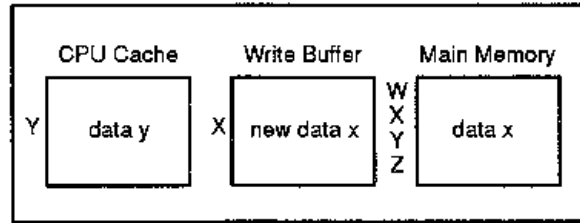
17852A–054

is allowed until all of these flagged writes propagate 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

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 ensures that the Am486DX/DX2 microprocessor updates all memory locations before reading status from an I/O devices.

The Am486DX/DX2 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 Am486DX/DX2 microprocessor or to mask interrupt's before the processor progresses to the instruction following OUT. REP OUTS instructions are buffered.

I/O device recovery time must be handled slightly differently with the Am486DX/DX2 microprocessor than with the 386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the 386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the 386 microprocessor to generate a prefetch bus cycle that cannot begin until the I/O writes complete.

Inserting a jump to the next write does not work with the Am486DX/DX2 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 is not 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 has 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 Am486DX/DX2 microprocessor, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the Am486DX/DX2 microprocessor always accesses external memory. It never looks 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 Am486DX/DX2 microprocessor's write buffers are written to memory before a locked cycle is allowed to proceed to the external bus.

The Am486DX/DX2 microprocessor asserts the $\overline{LOCK}$ pin after the write buffers are emptied during a locked bus cycle. With the $\overline{LOCK}$ pin asserted, the microprocessor reads the data, operates on the data, and places the results in a write buffer. The contents of the write buffer are then written to external memory. $\overline{LOCK}$ becomes inactive after the write part of the locked cycle.

6.4   INTERRUPT AND NON-MASKABLE INTERRUPT INTERFACE

The Am486DX/DX2 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 2.8. For interrupt timings, refer to Section 7.2.10.

6.4.1   Interrupt Logic

The Am486DX/DX2 microprocessor contains a two clock synchronizer on the interrupt line. An interrupt request reaches the internal instruction execution unit two clocks after the INTR pin is asserted, if proper set up 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 to be recognized by the instruction execution unit. The interrupt is not serviced by the Am486DX/DX2 microprocessor if the INTR signal does not remain active.

The instruction execution unit looks 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 are only accepted at these boundaries.

An interrupt must be presented to the Am486DX/DX2 microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt three 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 is accepted at the end of the next instruction, assuming INTR is still held active. The interrupt service microcode starts 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 operates 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 remains 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 is executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5   RESET AND INITIALIZATION

The Am486DX/DX2 microprocessor has a built-in self-test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted in the clock prior to RESET going from High to Low. RESET must be active for 15 clocks with or without BIST being enabled. Refer to Chapter 8 for information on Am486DX/DX2 microprocessor testability.

The Am486DX/DX2 microprocessor registers have the values shown in Table 6-2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) contains 04 and the lower byte (DL) contains a stepping identifier (see Table 6-4). 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.

The Am486DX/DX2 microprocessor starts executing instructions at location FFFFFFF0H and RESET. When the first Intersegment Jump or Call is executed, address lines A31–A20 drop Low for CS-relative memory cycles, and the Am486DX/DX2 microprocessor only executes instructions in the lower 1 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 Am486DX/DX2 microprocessor to terminate all execution and local bus activity. No instruction or bus activity occurs as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1   Pin State During RESET

The Am486DX/DX2 microprocessor recognizes and can respond to HOLD, AHOLD, and $\overline{BOFF}$ requests, regardless of the RESET state. 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 Am486DX/DX2 microprocessor bus is in the state shown in Figure 6-5 if the HOLD, AHOLD, and $\overline{BOFF}$ requests are inactive. Note that the address (A31–A2, $\overline{BE3}$–$\overline{BE0}$) and cycle definition ($M/\overline{IO}$, $D/\overline{C}$, and $W/\overline{R}$) pins are undefined from the time RESET is asserted up to the start of the first bus cycle. All undefined pins (except $\overline{FERR}$) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H.

$\overline{FERR}$ reflects the state of the error summary status (ES) 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 $\overline{FERR}$ and the numeric status word register bits 7–0 depend on whether or not BIST is performed. Table 6-3 shows the state of $\overline{FERR}$ signal after RESET and before the execution of the FINIT/FNINIT instruction.

After the first FINIT or FNINIT instruction, $\overline{FERR}$ pin and the FPU status word register bits (7–0) will be inactive irrespective of the BIST.

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   FERR Pin State

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

Table 6-4 **Am486DX/DX2 CPU Revision ID**

CPU Stepping Name	Revision ID
B3	· 01
B4	01
B5	01
B6	01
C0	02
D0	04
CA2	10
CA3	10
CB0	11

KEY TO SWITCHING WAVEFORMS

WAVEFORM	INPUTS	OUTPUTS
	Must be Steady	Will be Steady
	May Change from H to L	Will be Changing from H to L
	May Change from L to H	Will be Changing from L to H
	Don't Care, Any Change Permitted	Changing, State Unknown
	Does Not Apply	Center Line is High Impedance "Off" State

KS000010

Figure 6-5   Pin States During RESET
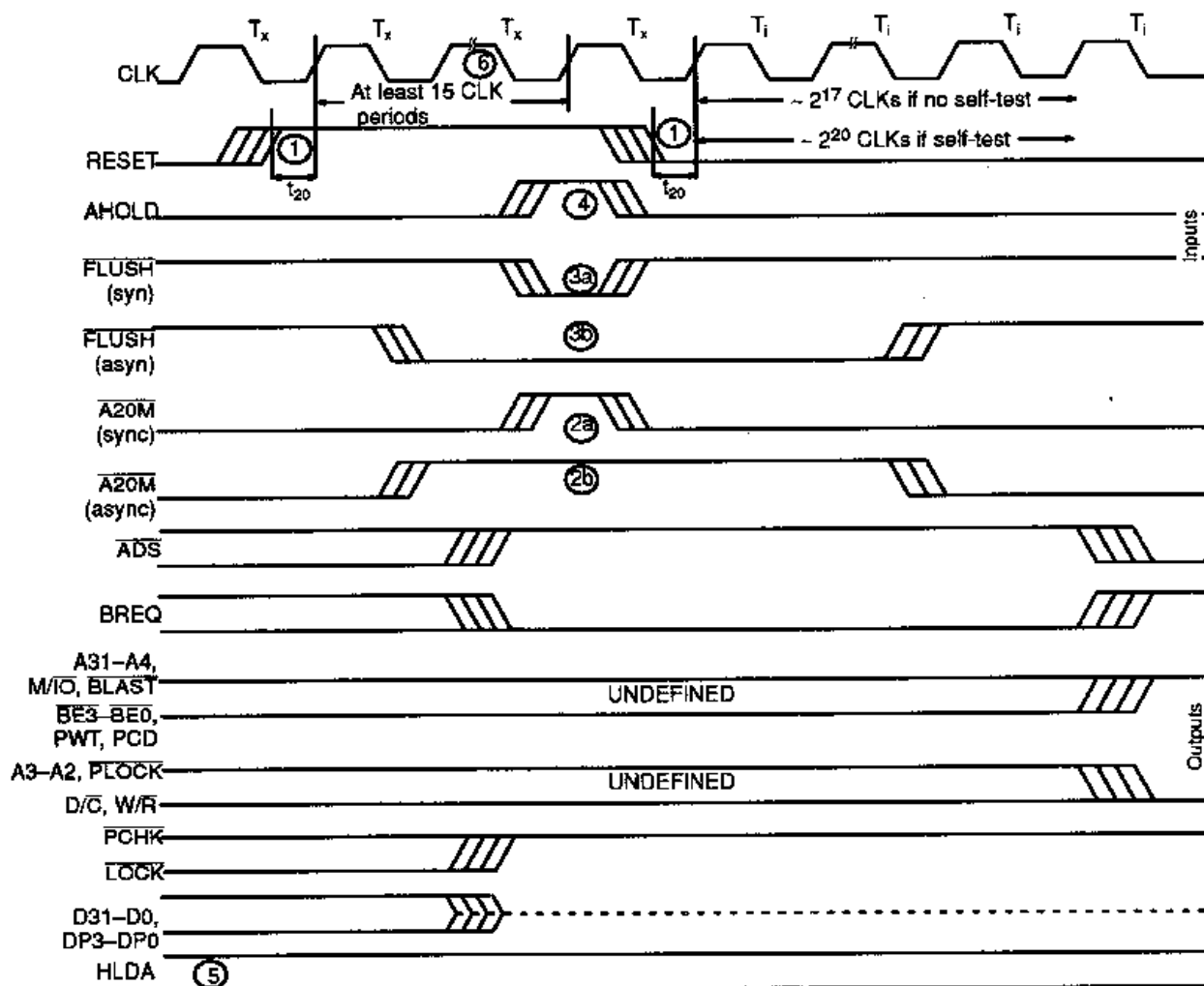
Notes:

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

2a. When $\overline{A20M}$ is driven synchronously, it must be driven High (inactive) for the CLK edge prior to the falling edge of RESET. This ensures proper operation. $\overline{A20M}$ setup and hold times must be met.

2b. When $\overline{A20M}$ 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 $\overline{FLUSH}$ is driven synchronously, it should be driven Low (active) for the CLK edge prior to the falling edge of RESET to invoke the Three-State Output Test Mode. All outputs are guaranteed three-stated within ten CLKs of RESET being deasserted. $\overline{FLUSH}$ setup and hold times must be met.

3b. When $\overline{FLUSH}$ 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 Three-State Output Test Mode. All outputs are guaranteed three-stated within ten 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.

17852A–055

7 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 can be transferred without restrictions on physical address alignment. Data can be accessed at any byte boundary, but two or three cycles can be required for unaligned data transfers. (See Section 7.1.3, Dynamic Data Bus Sizing, and Section 7.1.6, Operand Alignment.)

The Am486DX/DX2 microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A31–A2. The byte enables, $\overline{BE3}$–$\overline{BE0}$, 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, (see Table 7-1). Byte enable patterns that have a negated byte enable separating two or three asserted byte enables never occur (see Table 7-5). All other byte enable patterns are possible.

Address bits A0 and A1 of the physical operand's base address can be created when necessary. (Using the byte enables to create A0 and A1 is shown in Table 7-2). The byte enables can also be decoded to generate $\overline{BLE}$ (Byte Low Enable) and $\overline{BHE}$ (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 7-1   Byte Enables and Associated Data and Operand Bytes

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

Table 7-2   Generating A31–A0 from $\overline{BE3}$–$\overline{BE0}$ and A31–A2

CPU Address Signals							
A31–A2				$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
	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 can access physical memory space or I/O space. Peripheral devices in the system can either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 Gbytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O (see Figure 7-1).

7.1.2 Memory and I/O Space Organization

The Am486DX/DX2 microprocessor data path to memory and I/O spaces can be 32-, 16- or 8-bits wide. The byte enable signals, $\overline{BE3}$–$\overline{BE0}$, allow byte granularity when addressing any memory or I/O structure whether 8-,16-, or 32-bits wide.

The Am486DX/DX2 microprocessor includes bus control pins, $\overline{BS16}$ and $\overline{BS8}$, that allow direct connection to 16-, and 8-bit memories and I/O devices. Cycles to 32-, 16-, and 8-bit memories can occur in any sequence, since the $\overline{BS8}$ and $\overline{BS16}$ 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 D7–D0; the highest-addressed byte with D31–D24. Physical 4-byte words begin at addresses divisible by 4.

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

Figure 7-1   Physical Memory and I/O Spaces
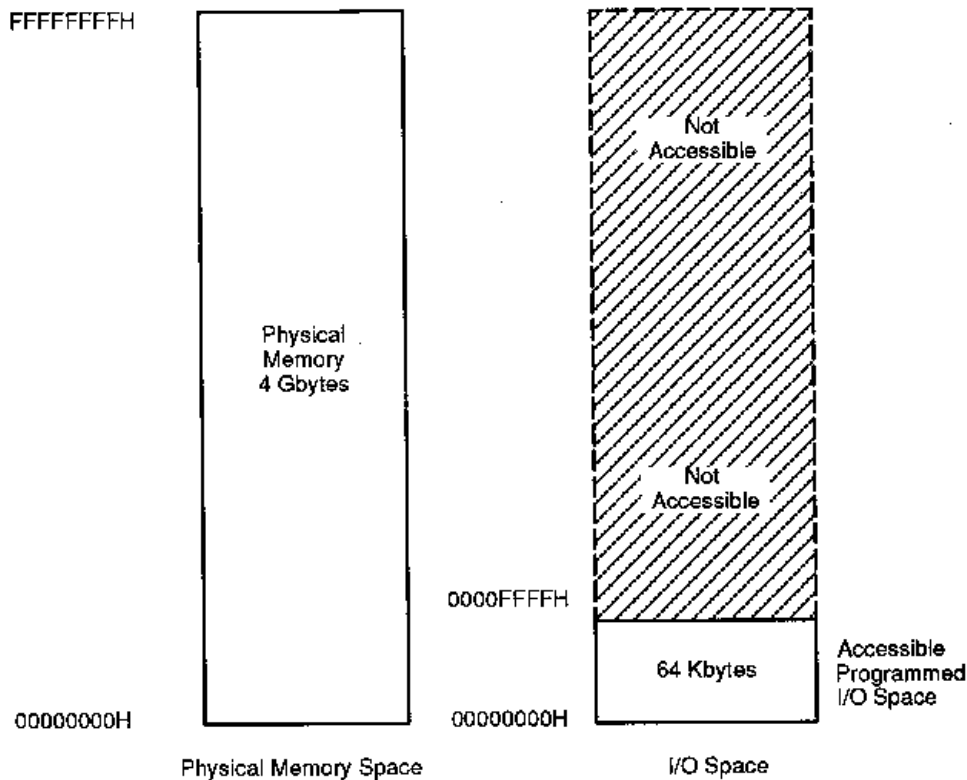
17852A–056

Figure 7-2   Physical Memory and I/O Space Organization

32-Bit Wide Organization

FFFFFFFFH   FFFFFFFCH

00000003H   00000000H

          BE3   BE2   BE1   BE0

16-Bit Wide Organization

FFFFFFFFH   FFFFFFFEH

00000001H   00000000H

          BHE   BLE
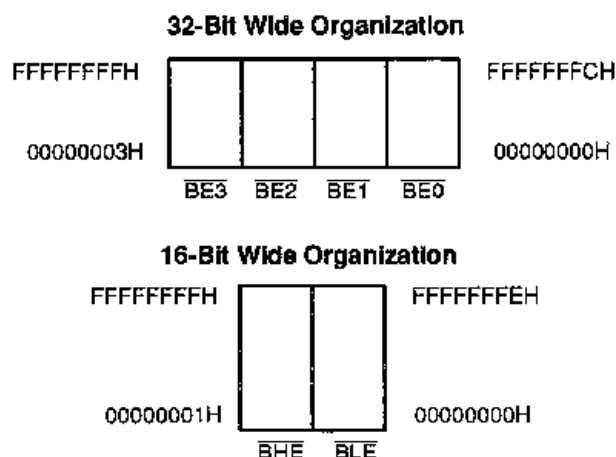
17852A–057

To address 8-bit memories, the two low-order address bits, A0 and A1, must be decoded from BE3–BE0. 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 can 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 can 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 theAm486DX/DX2 microprocessor to run additional bus cycles to complete requests larger than 16 or 8 bits. A 32-bit transfer is converted into two 16-bit transfers (or three transfers if the data is misaligned) when BS16 is asserted. Asserting BS8 converts 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 extra cycle unless the addressed device has the ability to change the number of bytes it can return between cycles.

The Am486DX/DX2 microprocessor drives the byte enables appropriately during extra cycles forced by BS8 and BS16. A31–A2 do not change if accesses are to a 32-bit aligned area. Table 7-3 shows the set of byte enables that are 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 Am486DX/DX2 microprocessor is significantly different than that of the 386 microprocessor. Unlike the 386 microprocessor, the Am486DX/DX2 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 Am486DX/DX2 microprocessor reads the two high-order bytes, they must be driven on the data bus pins D31–D16. The Am486DX/DX2 microprocessor expects the two low-order bytes on D15–D0. The 386 microprocessor expects both the high- and low-order bytes on D15–D0. The 386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16 is asserted.

The external system must contain buffers to enable the Am486DX/DX2 microprocessor to read and write data on the appropriate data bus pins. Table 7-4 shows the data bus

lines where the Am486DX/DX2 microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data is only driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus are driven but they do not contain valid data. Unlike the 386 microprocessor, the Am486DX/DX2 microprocessor does not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

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 4. A31–A2 are used as a 4-byte word select. $\overline{BE3}$–$\overline{BE0}$ select individual bytes within the 4-byte word. $\overline{BS8}$ and $\overline{BS16}$ are negated for all bus cycles involving the 32-bit array.

Table 7-3   Next Byte Enable Values for $\overline{BSn}$ Cycles

Current				Next with $\overline{BS8}$				Next with $\overline{BS16}$			
$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
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

Note:
"n" means another bus cycle is not required to satisfy the request.

Table 7-4   Data Pins Read with Different Bus Sizes

$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	w/o $\overline{BS8}$ / $\overline{BS16}$	w $\overline{BS8}$	w $\overline{BS16}$
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

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

Figure 7-4 shows the Am486DX/DX2 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, $\overline{BHE}$, and $\overline{BLE}$ (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 $\overline{BLE}$ is exactly the same as A0 (see Table 7-5).

$\overline{BE3}$–$\overline{BE0}$ can be decoded as shown in Table 7-5 to generate A1, $\overline{BHE}$, and $\overline{BLE}$. The byte select logic necessary to generate $\overline{BHE}$ and $\overline{BLE}$ is shown in Figure 7-5.

Figure 7-3   Am486 Microprocessor with 32-Bit Memory
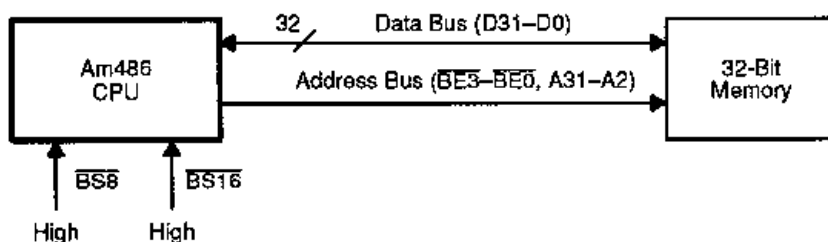
17852A–058

Figure 7-4   Addressing 16- and 8-Bit Memories
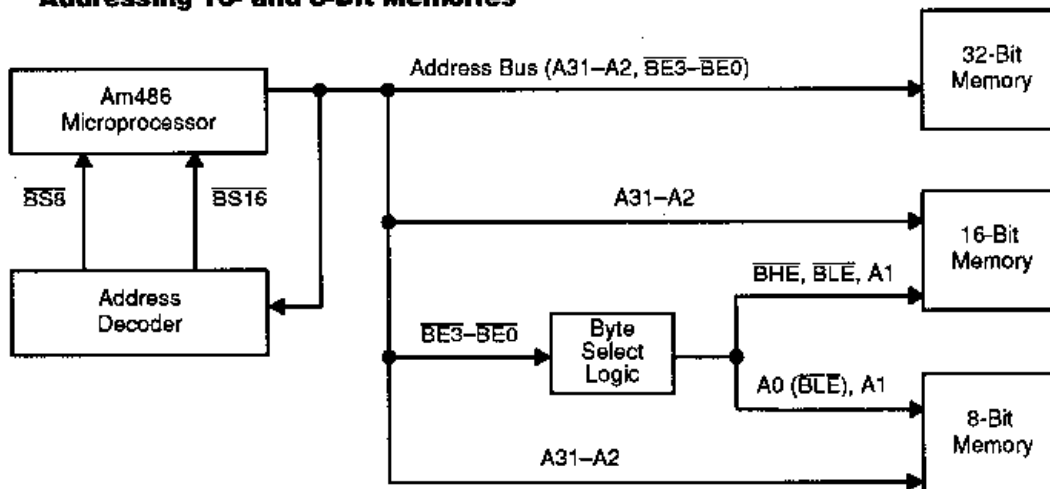
17852A–059

Figure 7-5   Logic to Generate A1, $\overline{BHE}$, and $\overline{BLE}$ for 16-Bit Buses
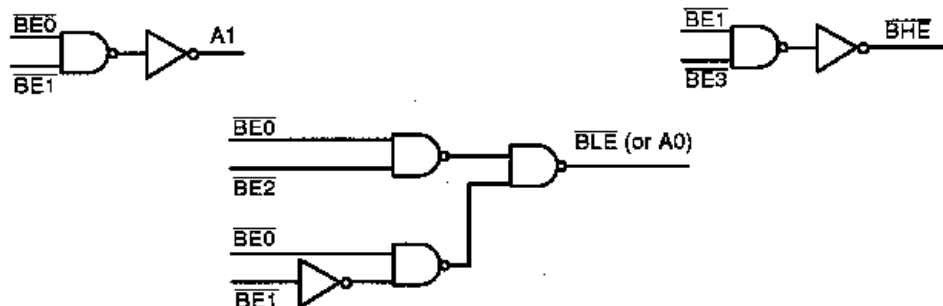
17852A–060

Combinations of BE3–BE0 that 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 BE3–BE0 combinations to its best advantage.

Figure 7-6 shows an Am486DX/DX2 microprocessor data bus interface to 32-, 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 Am486DX/DX2 microprocessor on the correct data pins (see Table 7-4).

Table 7-5   Generating A1, BHE, and BLE for Address for 16-Bit Devices

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-non-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-non-contiguous bytes
L*	H*	L*	H*	x	x	x	x-non-contiguous bytes
L*	H*	L*	L*	x	x	x	x-non-contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x-non-contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

Note:
BLE asserted when D7–D0 of 16-bit bus is active.
BHE asserted when D15–D8 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 reasserted, or the pattern has Byte Enables asserted for non-contiguous bytes

7.1.5   Dynamic Bus Sizing During Cache Line Fills

BS8 and BS16 can be driven during cache line fills. The Am486DX/DX2 microprocessor generates 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 Am486DX/DX2 microprocessor generates proper byte enables for

subsequent cycles in the line. Table 7-6 shows the appropriate A0 ($\overline{BLE}$), A1, and $\overline{BHE}$ for the various combinations of the Am486DX/DX2 microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that are generated by the Am486DX/DX2 microprocessor during a cache line fill.

7.1.6 Operand Alignment

Physical 4-byte words begin at addresses that are multiples of 4. 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 multibyte 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.

Figure 7-6 Data Bus Interface to 16- and 8-bit Memories
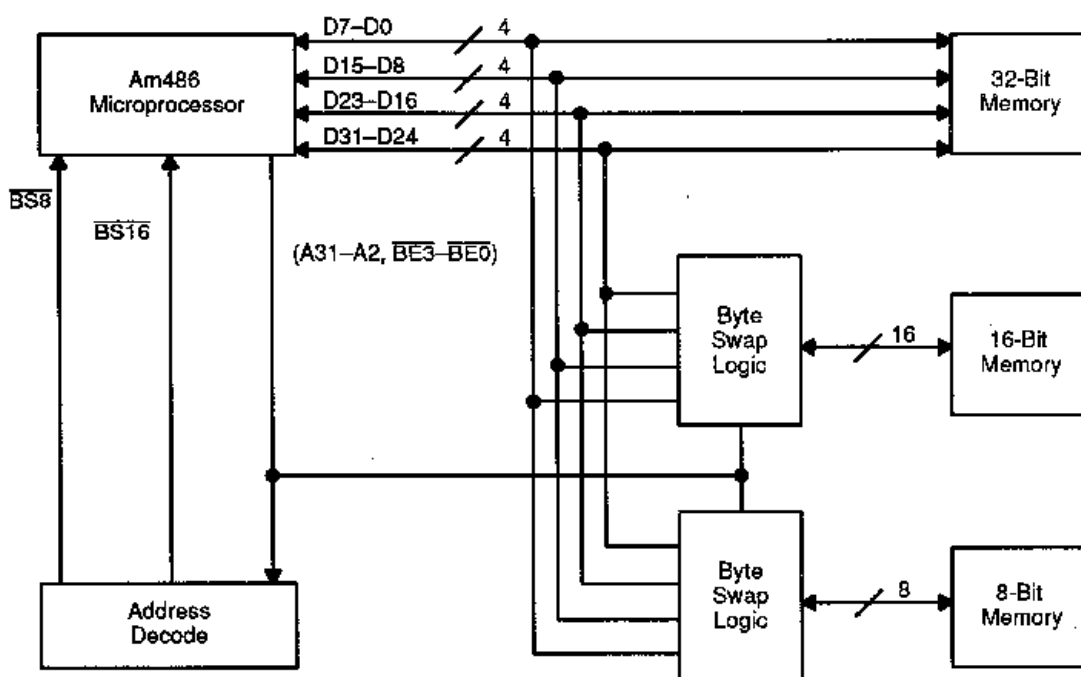
17852A–061

Table 7-6   Generating A0, A1, and $\overline{BHE}$ from the Am486DX/DX2 Microprocessor Byte Enables

$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	$\overline{BHE}$	A0	A1	$\overline{BHE}$
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

Note:
* All combination of byte enables that are generated by the CPU during a cache line fill.

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 = $\overline{BS16}$ 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 = $\overline{BS8}$ 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

4-byte Operand

| lb | mlb | mhb | hb |
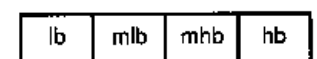
↑ byte with lowest address   ↑ byte with highest address

The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems assert $\overline{BS16}$ or $\overline{BS8}$ and force extra cycles, low-order bytes or words are transferred first (opposite to the example in Table 7-7). When the Am486DX/DX2 microprocessor requests a 4-byte read and the external system asserts $\overline{BS16}$, the lower two bytes are read first followed by the upper two bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted $\overline{BS16}$ 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 in Table 7-7.

7.2 BUS FUNCTIONAL DESCRIPTION

The Am486DX/DX2 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 $\overline{ADS}$ active in the first clock and ready active in the last clock. Data is transferred to or from the Am486DX/DX2 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 Am486DX/DX2 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 Am486DX/DX2 microprocessor initiates a cycle by asserting the address status signal ($\overline{ADS}$) at the rising edge of the first clock. The $\overline{ADS}$ 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 ($\overline{RDY}$) is returned by the external system in the second clock. $\overline{RDY}$ 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 Am486DX/DX2 microprocessor samples $\overline{RDY}$ at the end of the second clock. The cycle is complete if $\overline{RDY}$ is active (Low) when sampled. Note that $\overline{RDY}$ is ignored at the end of the first clock of the bus cycle.

The burst last signal ($\overline{BLAST}$) is asserted (Low) by the Am486DX/DX2 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 Am486DX/DX2 microprocessor asserts $\overline{BLAST}$ in the last cycle of a bus transfer.

The timing of the parity check output ($\overline{PCHK}$) is shown in Figure 7-7. The Am486DX/DX2 microprocessor drives the $\overline{PCHK}$ output one clock after ready terminates a read cycle.

Figure 7-7   Basic 2-2 Bus Cycle
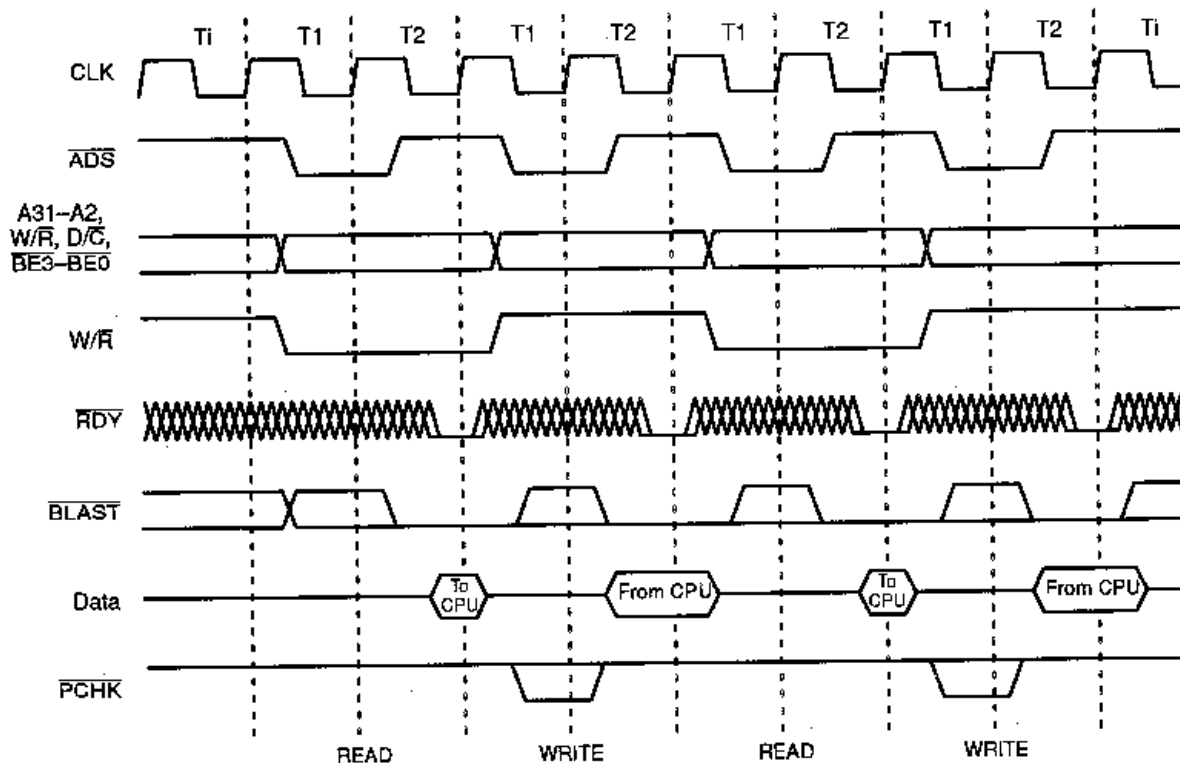
17852A–062

$\overline{PCHK}$ indicates the parity status for the data sampled at the end of the previous clock. The $\overline{PCHK}$ signal can be used by the external system. The Am486DX/DX2 microprocessor does nothing in response to the $\overline{PCHK}$ output.

7.2.1.2   Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving $\overline{RDY}$ inactive at the end of the second clock. $\overline{RDY}$ 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 Am486DX/DX2 microprocessor bus cycle by maintaining $\overline{RDY}$ inactive.

The burst ready input ($\overline{BRDY}$) must be driven inactive on all clock edges where $\overline{RDY}$ 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 Am486DX/DX2 microprocessor or by the external memory system. An internal request for a 64-bit floating-point load or a 128-bit prefetch must take more than one cycle. Internal requests for unaligned data can 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.

Figure 7-8   Basic 3-3 Bus Cycle
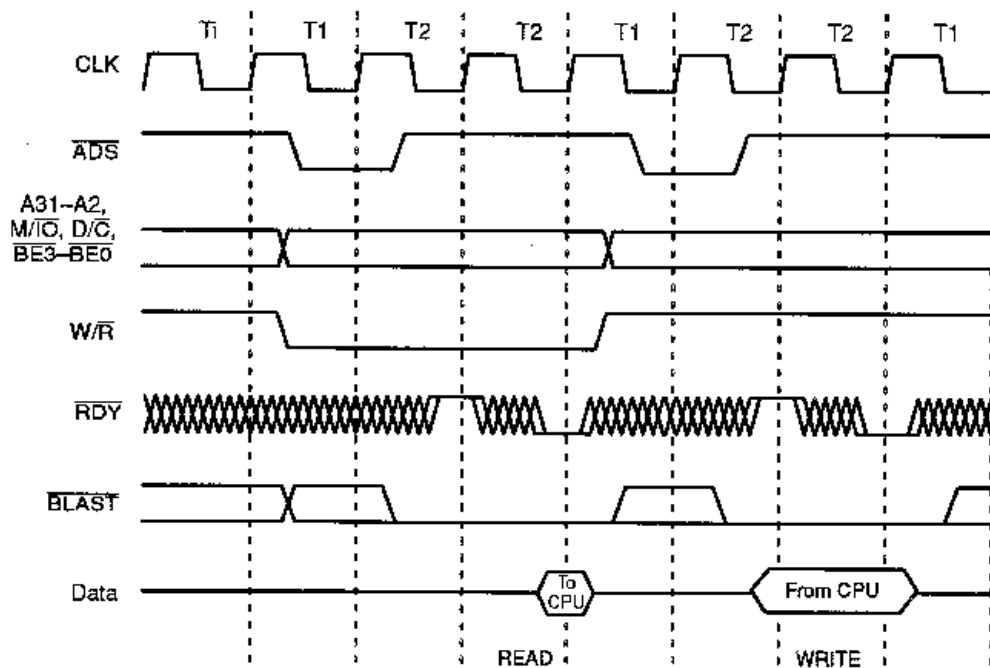
7.2.2.1   Burst Cycles

The Am486DX/DX2 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 Am486DX/DX2 microprocessor every clock, rather than every other clock as in non-burst cycles. The fastest burst cycle requires two clocks for the first data item with subsequent data items returned every clock.

The Am486DX/DX2 microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if $\overline{BS8}$ or $\overline{BS16}$ is asserted. For example, the Am486DX/DX2 microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the Am486DX/DX2 microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the Am486DX/DX2 microprocessor driving out an address and asserting $\overline{ADS}$ in the same manner as non-burst cycles. The Am486DX/DX2 microprocessor indicates it is willing to perform a burst cycle by holding the burst last signal ($\overline{BLAST}$) 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 ($\overline{BRDY}$) active.

The addresses of the data items in a burst cycle all fall within the same 16-byte aligned area (corresponding to an internal Am486DX/DX2 microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0H and ends at location XXXXXXXFH. During a burst cycle, only $\overline{BE3}$–$\overline{BE0}$, A2, and A3 can change. A31–A4, M/$\overline{IO}$, D/$\overline{C}$, and W/$\overline{R}$ 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 Am486DX/DX2 microprocessor internal cache lines.

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the Am486DX/DX2 microprocessor can be converted into a burst cycle. The Am486DX/DX2 microprocessor only bursts the number of bytes needed to complete a transfer. For example, eight bytes are 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 $\overline{BRDY}$ active rather than $\overline{RDY}$ (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted, such as interrupt acknowledge and halt, $\overline{BRDY}$ has the same effect as $\overline{RDY}$. $\overline{BRDY}$ is ignored if both $\overline{BRDY}$ and $\overline{RDY}$ are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with $\overline{RDY}$.

7.2.2.2   Terminating Multiple and Burst Cycle Transfers

The Am486DX/DX2 microprocessor drives $\overline{BLAST}$ inactive for all but the last cycle in a multiple cycle transfer. $\overline{BLAST}$ is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. $\overline{BLAST}$ is driven active in the last cycle of the transfer, indicating that the next time $\overline{BRDY}$ or $\overline{RDY}$ is returned, the transfer is complete.

$\overline{BLAST}$ is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when $\overline{RDY}$ or $\overline{BRDY}$ 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 ($\overline{BS8}$ and $\overline{BS16}$), the state of the cache enable input ($\overline{KEN}$), and alignment of the data to be transferred.

When the Am486DX/DX2 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 $\overline{KEN}$, $\overline{BS8}$, and $\overline{BS16}$ inputs one clock before $\overline{RDY}$ or $\overline{BRDY}$ is returned. The Am486DX/DX2 microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

$\overline{BLAST}$ is not valid in the first clock of a bus cycle because the Am486DX/DX2 microprocessor cannot determine the number of cycles a transfer will take until the external system returns $\overline{KEN}$, $\overline{BS8}$, and $\overline{BS16}$. $\overline{BLAST}$ should only be sampled in the second and subsequent clocks of a cycle when the external system returns $\overline{RDY}$ or $\overline{BRDY}$.

The system can terminate a burst cycle by returning $\overline{RDY}$ instead of $\overline{BRDY}$. $\overline{BLAST}$ remains deasserted until the last transfer. However, any transfers required to complete a cache line fill follow the burst order (e.g., if burst order was 4, 0, C, 8 and $\overline{RDY}$ was returned at after 0, the next transfers are 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 Am486DX/DX2 microprocessor indicates to the external system that this is a multiple cycle transfer by driving $\overline{BLAST}$ inactive during the second clock of the first cycle. The external system returns $\overline{RDY}$ active, indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning $\overline{KEN}$ inactive one clock before it returns $\overline{RDY}$ active. When the Am486DX/DX2 microprocessor samples $\overline{RDY}$ active, it ignores $\overline{BRDY}$.

Each cycle in the transfer begins when $\overline{ADS}$ is driven active, and the cycle is complete when the external system returns $\overline{RDY}$ active.

The Am486DX/DX2 microprocessor indicates the last cycle of the transfer by driving $\overline{BLAST}$ active. The next $\overline{RDY}$ returned by the external system terminates the transfer.

Figure 7-9   Non-Cacheable, Non-Burst, Multiple Cycle Transfers

1st Data   2nd Data
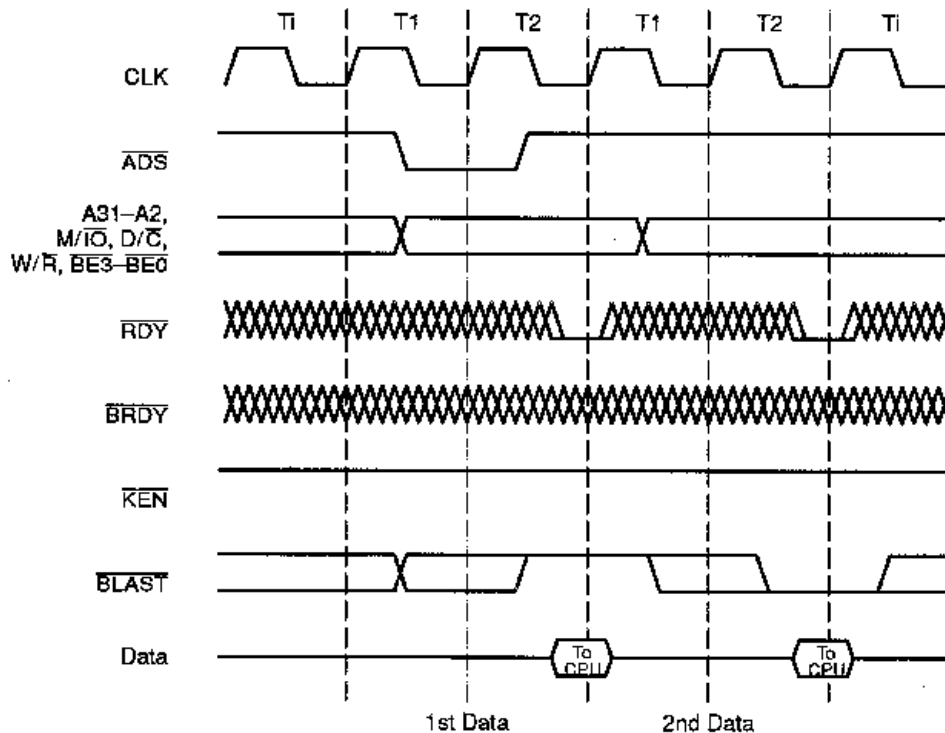
17852A–064

7.2.2.4   Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning $\overline{BRDY}$ active rather than $\overline{RDY}$ in the first cycle of the transfer (see Figure 7-10).

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

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

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 $\overline{KEN}$ active one clock before $\overline{RDY}$ or $\overline{BRDY}$ during the first cycle of the transfer on the external bus. Once $\overline{KEN}$ is asserted and the remaining three requirements described below are met, the Am486DX/DX2 microprocessor fetches an entire cache line, regardless of the state of $\overline{KEN}$. $\overline{KEN}$ must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The Am486DX/DX2 microprocessor only converts memory reads or prefetches into a cache fill. $\overline{KEN}$ is ignored during write or I/O cycles. Memory writes are only 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 $\overline{KEN}$ pin must be asserted one clock prior to $\overline{RDY}$ or $\overline{BRDY}$ being returned for the first data cycle.

Figure 7-10   Non-Cacheable, Burst Cycle
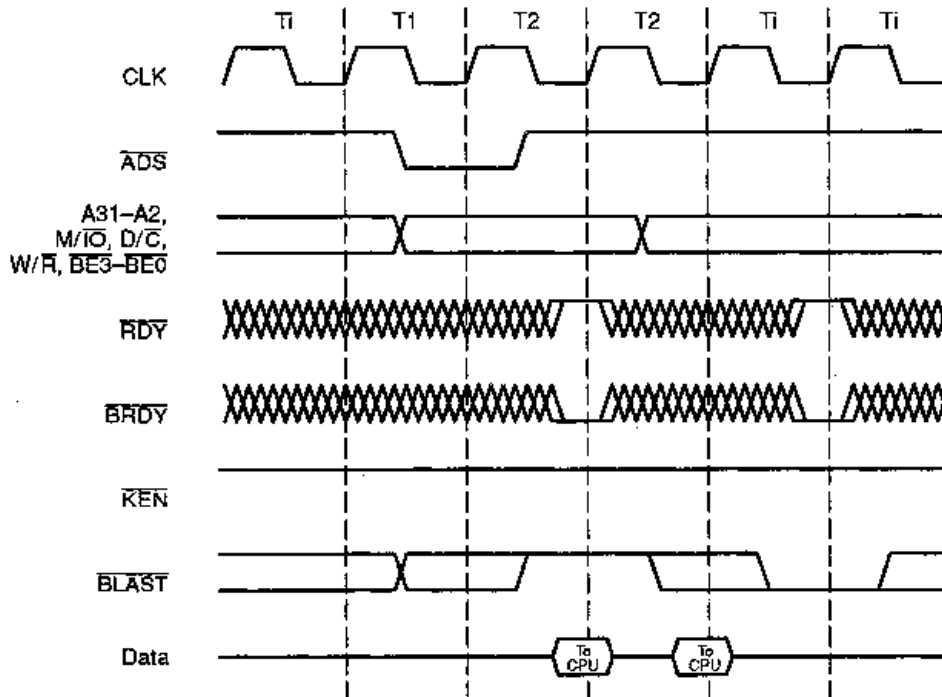
17852A–065

2. The cycle must be 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 PCD bit 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 CD bit in control register 0 (CR0) must be clear.

External hardware can determine when the Am486DX/DX2 microprocessor has transformed a read or prefetch into a cache fill by examining the $\overline{KEN}$, M/$\overline{IO}$, D/$\overline{C}$, W/$\overline{R}$, $\overline{LOCK}$, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the Am486DX/DX2 microprocessor 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 Am486DX/DX2 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 $\overline{BS8}$ is asserted, only one byte need be returned on data lines D7–D0. Similarly, if $\overline{BS16}$ is asserted, two bytes should be returned on D15–D0.

The Am486DX/DX2 microprocessor generates 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 Am486DX/DX2 microprocessor samples $\overline{KEN}$ active at the end of the first clock. The Am486DX/DX2 microprocessor drives $\overline{BLAST}$ inactive in the second clock in response to $\overline{KEN}$. $\overline{BLAST}$ is driven inactive because a cache fill requires three additional cycles to complete. $\overline{BLAST}$ remains inactive until the last transfer in the cache line fill. $\overline{KEN}$ 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 $\overline{KEN}$ 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 $\overline{BLAST}$ output is invalid in the first clock of a cycle. $\overline{BLAST}$ can be active during the first clock due to earlier inputs. Ignore $\overline{BLAST}$ 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 Am486DX/DX2 microprocessor drives the address lines and byte enables (see Section 7.2.4.2, Burst and Cache Line Fill Order).

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 $\overline{KEN}$ active at the end of the first clock in the cycle.

The external system informs the Am486DX/DX2 microprocessor that it will burst the line in by driving $\overline{BRDY}$ active at the end of the first cycle in the transfer.

Figure 7-11   Non-Burst, Cacheable Cycles
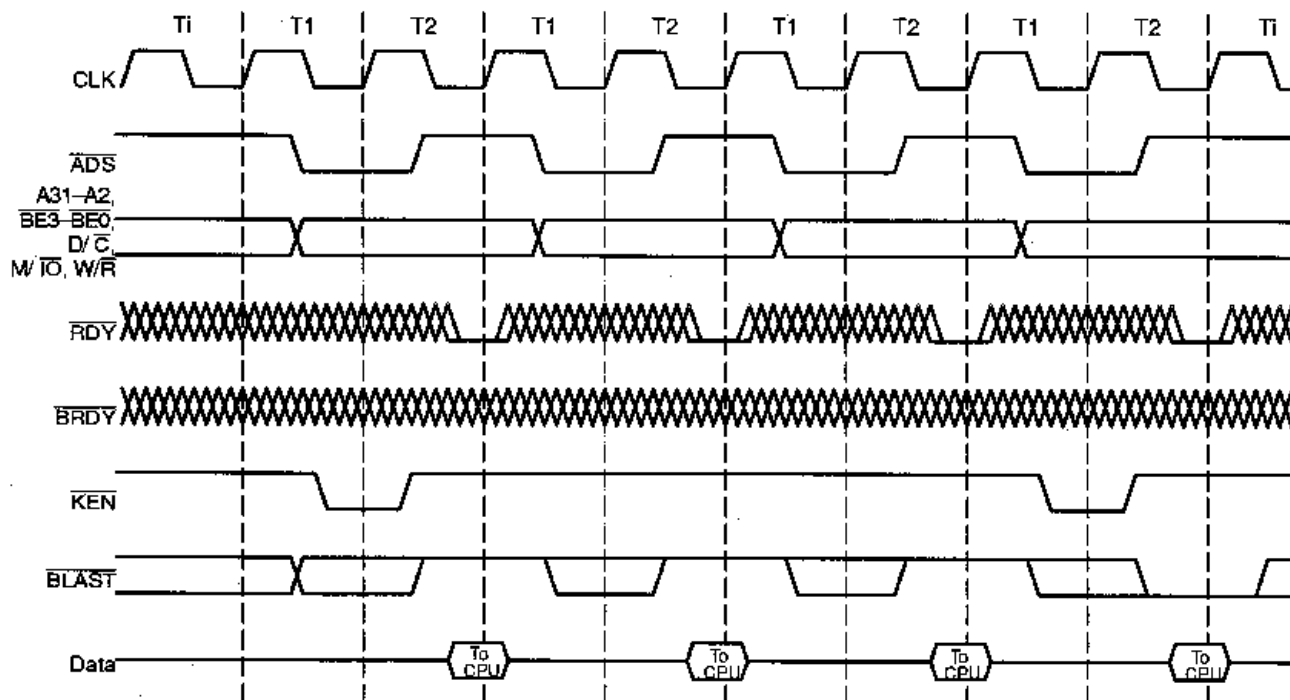
17852A-066

Note that during a burst cycle, $\overline{ADS}$ is only driven with the first address.

7.2.3.4 Effect of Changing KEN during a Cache Line Fill

$\overline{KEN}$ can change multiple times as long as it arrives at its final value in the clock before $\overline{RDY}$ or $\overline{BRDY}$ is returned (see Figure 7-13). Note that the timing of $\overline{BLAST}$ follows that of $\overline{KEN}$ by one clock. The Am486DX/DX2 microprocessor samples $\overline{KEN}$ every clock and uses the value returned in the clock before $\overline{RDY}$ to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of $\overline{KEN}$ in the last cycle before early $\overline{RDY}$ to load the line just retrieved from the memory into the cache. $\overline{KEN}$ is sampled every clock and must satisfy setup and hold time.

$\overline{KEN}$ 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.

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 Am486DX/DX2 microprocessor only strobes data into the chip when either $\overline{RDY}$ or $\overline{BRDY}$ are active.

Driving $\overline{BRDY}$ and $\overline{RDY}$ 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 7-12 Burst Cacheable Cycle
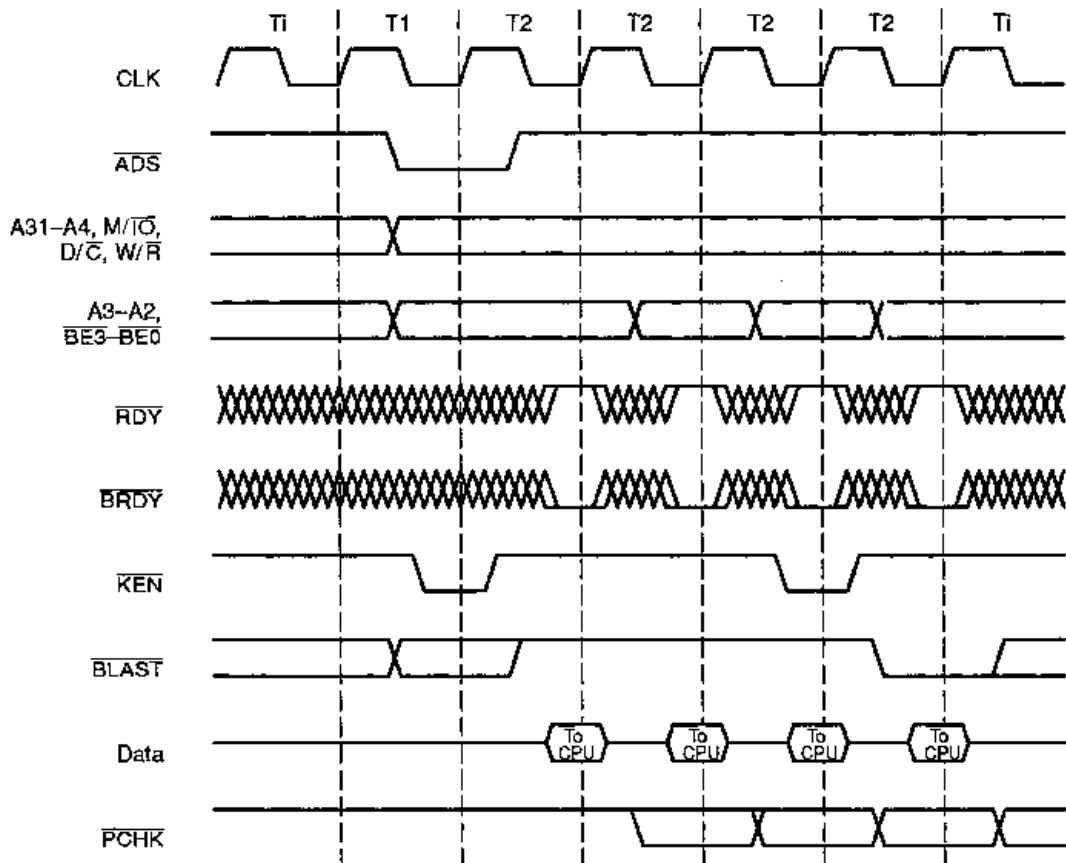
17852A–067

Figure 7-13   Effect of Changing KEN
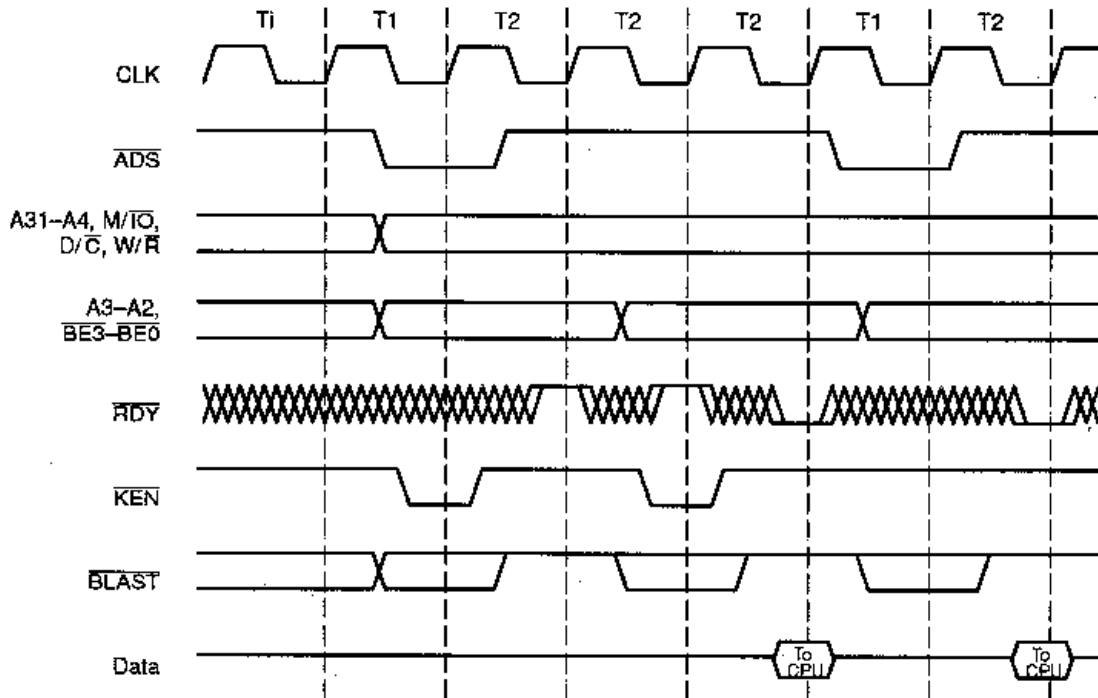
17852A-068

Figure 7-14   Slow Burst Cycle
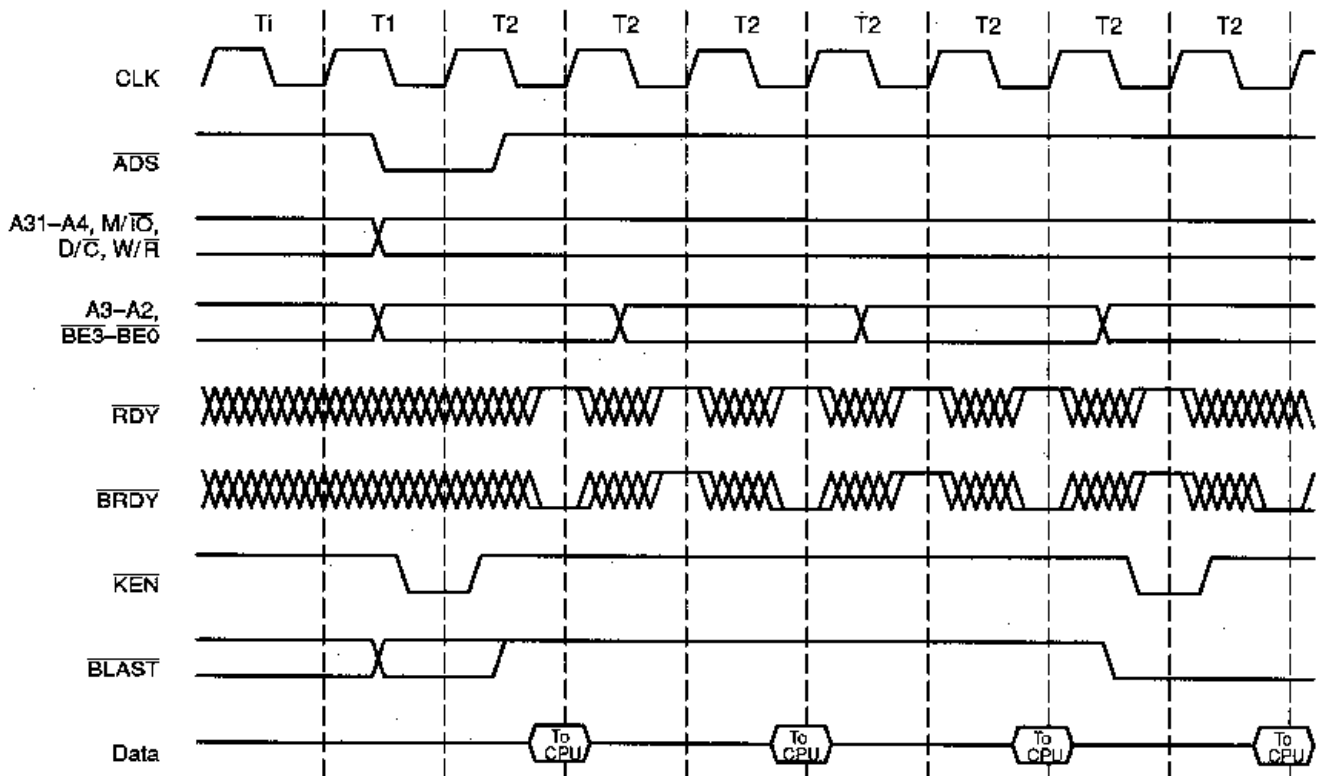
17852A-069

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the Am486DX/DX2 microprocessor is shown in Table 7-8. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not), or code prefetch.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address is 104, the next three addresses in the burst will be 100, 10C, and 108. An example of burst address sequencing is shown in Figure 7-15.

The sequences shown in Table 7-8 accommodate systems with 64-bit buses, as well as systems with 32-bit data buses. 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 prefetch. If either $\overline{BS8}$ or $\overline{BS16}$ is returned active, the Am486DX/DX2 microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a $\overline{BS16}$ burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

Table 7-8   Burst Order

First Address	Second Address	Third Address	Fourth Address
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

Figure 7-15   Burst Cycle Showing Order of Addresses
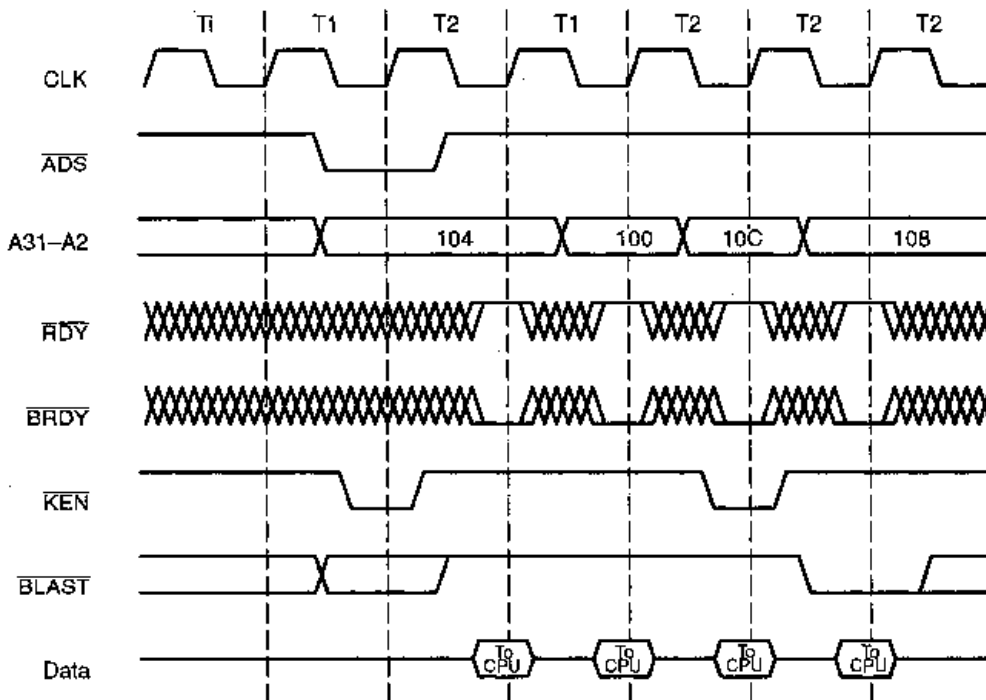
17852A–070

7.2.4.3   Interrupted Burst Cycles

Some memory systems might not be able to respond with burst cycles in the order defined in Table 7-8. To support these systems, the Am486DX/DX2 microprocessor allows a burst cycle to be interrupted at any time.

The Am486DX/DX2 microprocessor automatically generates 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 $\overline{RDY}$ instead of $\overline{BRDY}$. $\overline{RDY}$ can be returned after any number of data cycles terminated with $\overline{BRDY}$.

An example of an interrupted burst cycle is shown in Figure 7-16. The Am486DX/DX2 microprocessor immediately drives $\overline{ADS}$ active to initiate a new bus cycle after $\overline{RDY}$ is returned active. $\overline{BLAST}$ is driven inactive one clock after $\overline{ADS}$ begins the second bus cycle, indicating that the transfer is incomplete.

$\overline{KEN}$ need not be returned active in the first data cycle of the second part of the transfer in Figure 7-16. The cycle was converted to a cache fill in the first part of the transfer and the Am486DX/DX2 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 Am486DX/DX2 microprocessor requests operands during an interrupted burst transfer is determined in Table 7-8. Mixing $\overline{RDY}$ and $\overline{BRDY}$ does not change the order in which the Am486DX/DX2 microprocessor requests operand addresses.

Figure 7-16   Interrupted Burst Cycle
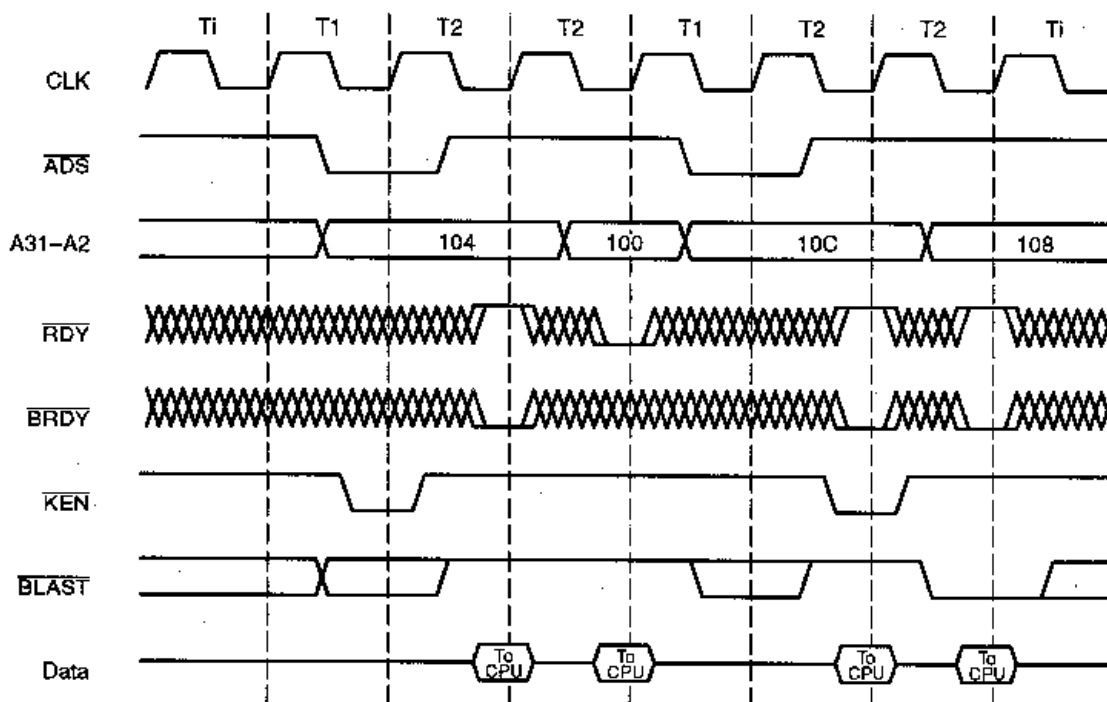
17852A–071

An example of the order in which the Am486DX/DX2 microprocessor requests operands during a cycle, in which the external system mixes $\overline{RDY}$ and $\overline{BRDY}$, is shown in Figure 7-17. The Am486DX/DX2 microprocessor initially requests a transfer beginning at Location 104. The transfer becomes a cache line fill when the external system returns $\overline{KEN}$ active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with $\overline{RDY}$. The Am486DX/DX2 microprocessor drives out a new request (by asserting $\overline{ADS}$) to address 100. If the external system terminates the second cycle with $\overline{BRDY}$, the Am486DX/DX2 microprocessor then requests/expects address 10C. The correct order is determined by the first cycle in the transfer, which might not be the first cycle in the burst if the system mixes $\overline{RDY}$ with $\overline{BRDY}$.

7.2.5   8- and 16-Bit Cycles

The Am486DX/DX2 microprocessor supports both 16- and 8-bit external buses through the $\overline{BS16}$ and $\overline{BS8}$ inputs. $\overline{BS16}$ and $\overline{BS8}$ allow the external system to specify, on a cycle-by-cycle basis, whether the addressed component can supply 8, 16, or 32 bits. $\overline{BS16}$ and $\overline{BS8}$ can be used in burst cycles as well as non-burst cycles. If both $\overline{BS16}$ and $\overline{BS8}$ are returned active for any bus cycle, the Am486DX/DX2 microprocessor responds as if only $\overline{BS8}$ were active.

The timing of $\overline{BS16}$ and $\overline{BS8}$ is the same as that of $\overline{KEN}$. $\overline{BS16}$ and $\overline{BS8}$ must be driven active before the first $\overline{RDY}$ or $\overline{BRDY}$ is driven active.

Driving the $\overline{BS16}$ and $\overline{BS8}$ active can force the Am486DX/DX2 microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. $\overline{BS8}$ and $\overline{BS16}$ can change the state of $\overline{BLAST}$ when they force subsequent cycles from the transfer.

Figure 7-18 shows an example in which $\overline{BS8}$ forces the Am486DX/DX2 microprocessor to run two extra cycles to complete a transfer. The Am486DX/DX2 microprocessor issues a

Figure 7-17   Interrupted Burst Cycle with Unobvious Order of Addresses
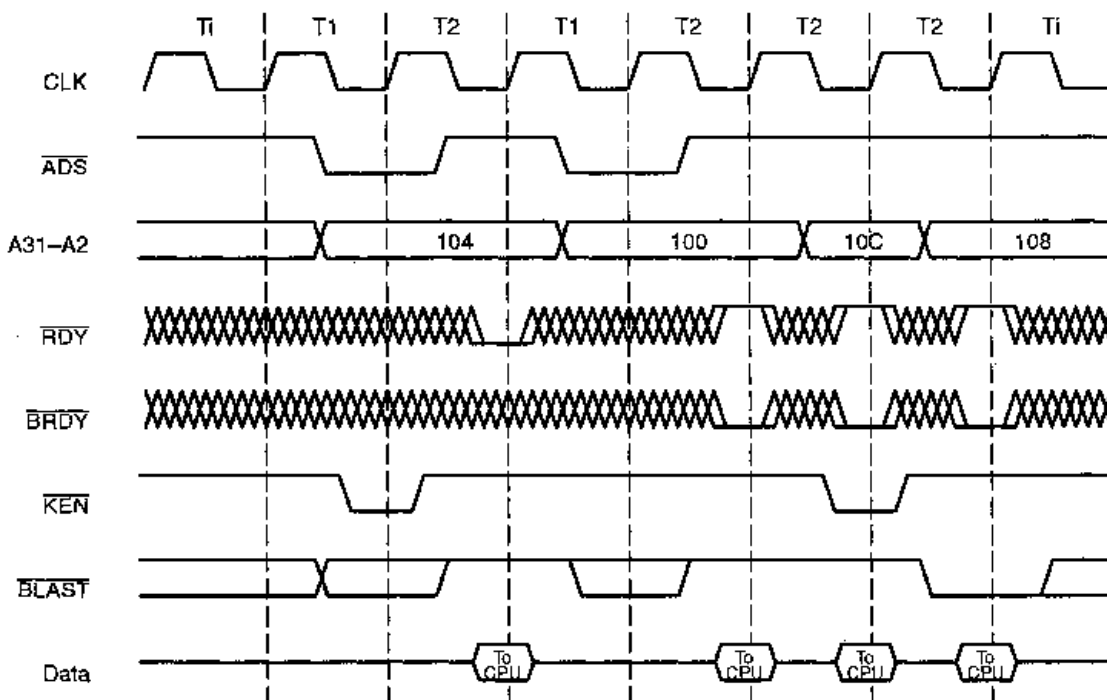
17852A–072

request for 24 bits of information. The external system drives $\overline{BS8}$ active, indicating that only eight bits of data can be supplied per cycle. The Am486DX/DX2 microprocessor issues two extra cycles to complete the transfer.

Extra cycles forced by the $\overline{BS16}$ and $\overline{BS8}$ should be viewed as independent bus cycles. $\overline{BS16}$ and $\overline{BS8}$ 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 Am486DX/DX2 microprocessor drives $\overline{BLAST}$ inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while $\overline{BS8}$ or $\overline{BS16}$ are active.

$\overline{BS8}$ and $\overline{BS16}$ operate during burst cycles exactly the same as non-burst cycles. For example, a single non-cacheable read can be transferred by the Am486DX/DX2 microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write can 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 $\overline{BS8}$ or $\overline{BS16}$ is asserted.

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 exchange (xchg) 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 $\overline{LOCK}$ instruction prefix is used with selected instructions.

Figure 7-18   8-Bit Bus Size Cycle
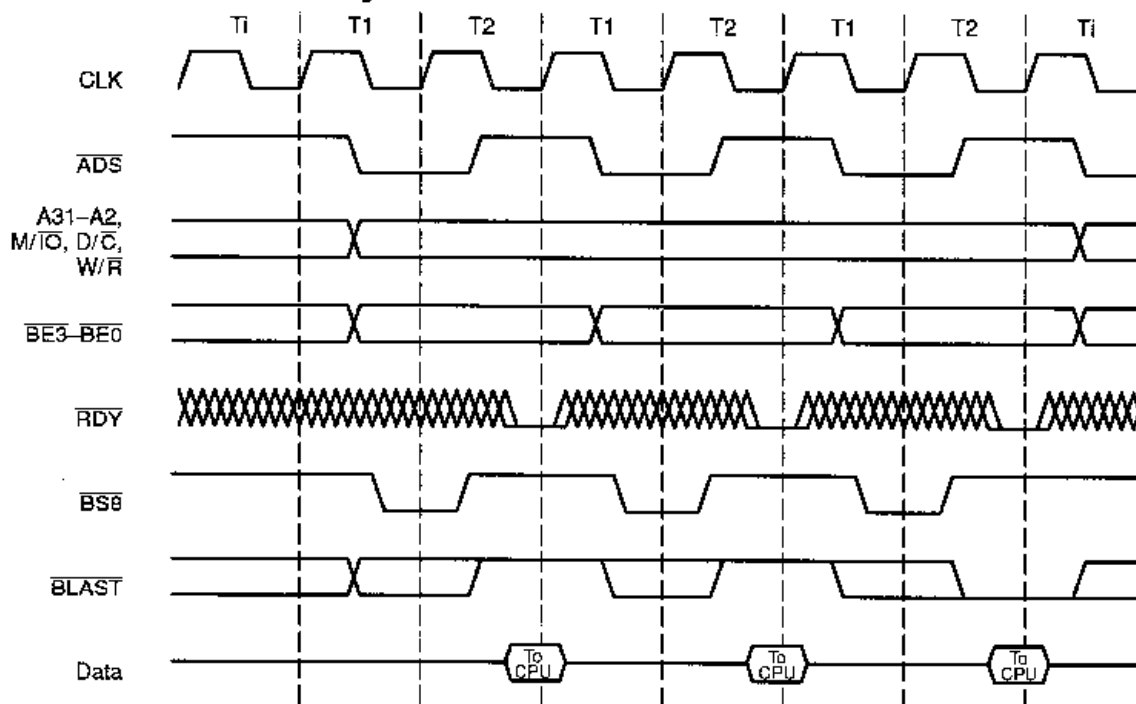
17852A-073

Figure 7-19 Burst Write as a Result of BS8 or BS16
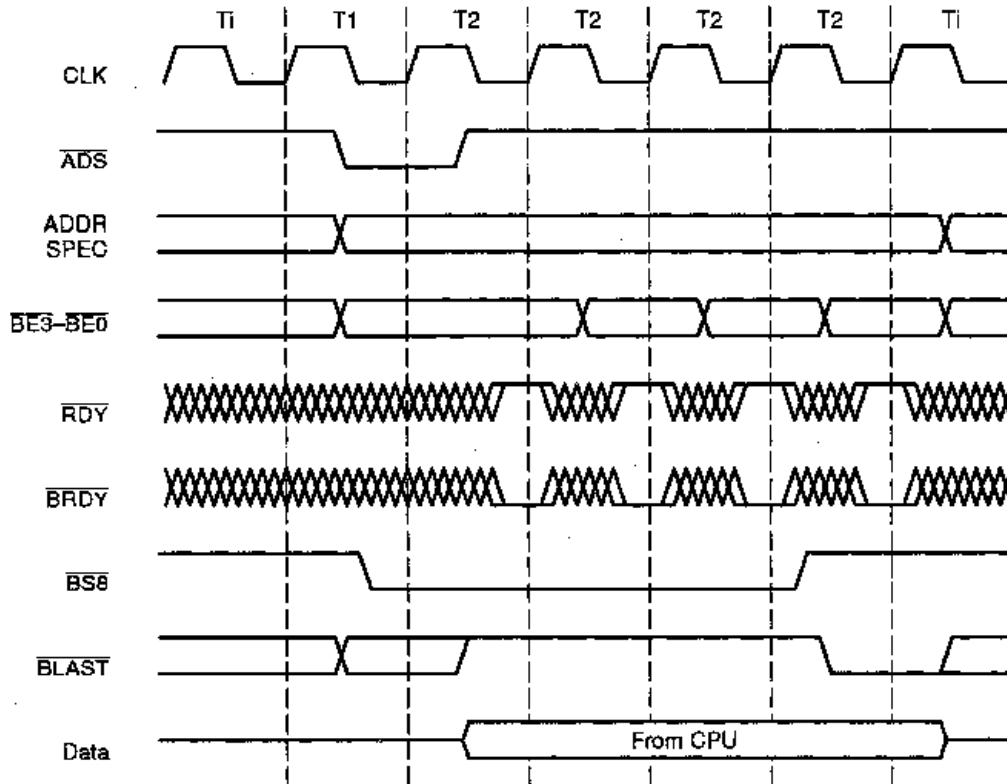
17852A–074

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-bit read-modify-write operation, the LOCK remains active for the entire duration of the multiple cycle. It goes inactive when RDY is returned for the last write cycle.

When LOCK is active, the Am486DX/DX2 microprocessor recognizes address hold and backoff but does not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the Am486DX/DX2 microprocessor generates LOCK.

7.2.7   Pseudo-Locked Cycles

Pseudo-locked cycles ensure that no other master is given control of the bus during operand transfers that take more than one bus cycle.

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 Am486DX/DX2 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 is deasserted in response to BS8 or BS16. A 64-bit write is driven out as two non-burst bus cycles. BLAST is asserted during both writes since a burst is impossible. PLOCK is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7-21.

Figure 7-20   Locked Bus Cycle
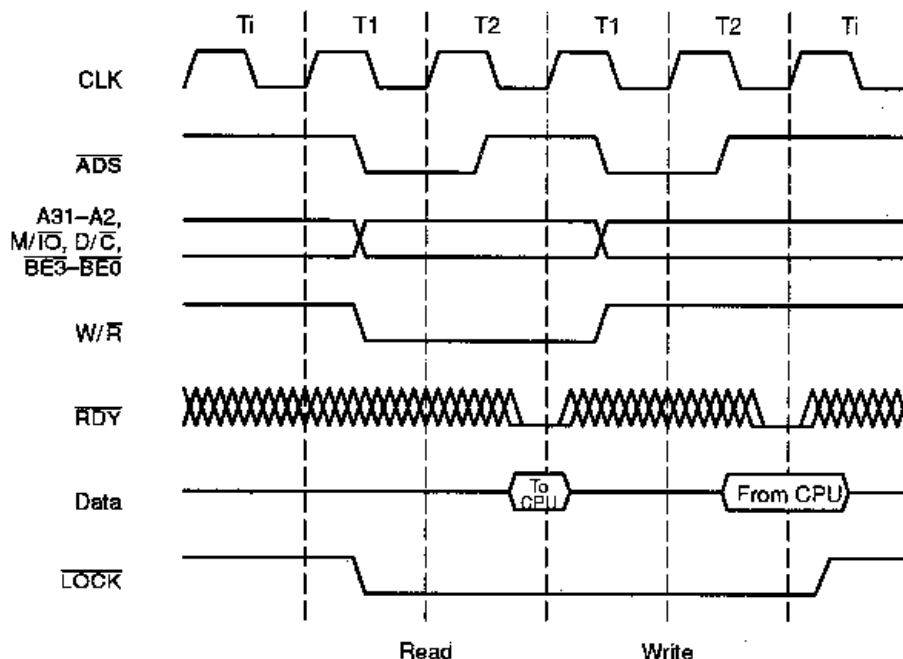
17852A–075

Figure 7-21   Pseudo Lock Timing
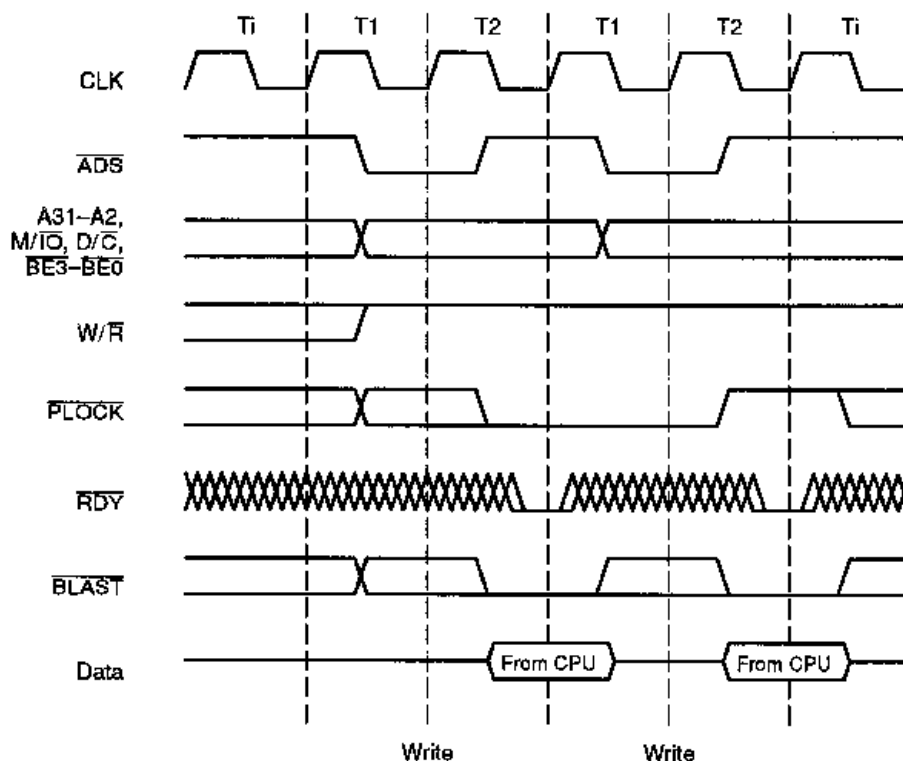
17852A–076

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.

$\overline{PLOCK}$ is asserted since $\overline{BLAST}$ is negated, but it is ignored and HOLD is recognized during the prefetch.

$\overline{PLOCK}$ can change several times during a cycle, settling to its final value when the clock $\overline{RDY}$ is returned.

7.2.8 Invalidate Cycles

Invalidate cycles are needed to keep the Am486DX/DX2 microprocessor's internal cache contents consistent with external memory. The Am486DX/DX2 microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a section of external memory contained in its internal cache, the processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address ($\overline{EADS}$). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input, forcing the Am486DX/DX2 microprocessor to immediately relinquish its address bus. Next, the external system asserts $\overline{EADS}$, indicating that a valid address is on the Am486DX/DX2 microprocessor's address bus. Figure 7-22 shows the fastest possible invalidation cycle. The Am486DX/DX2 CPU cycle recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, $\overline{EADS}$ 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 Am486DX/DX2 microprocessor's address bus is input/output, unlike the 386 microprocessor's bus which is output only.

The Am486DX/DX2 microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be three wait states into a read cycle. If AHOLD is activated, the Am486DX/DX2 microprocessor immediately floats 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 Figure 7-22 and Figure 7-23).

$\overline{EADS}$ is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for $\overline{EADS}$ to generate an internal invalidate. If $\overline{EADS}$ alone is asserted while the Am486DX/DX2 microprocessor is driving the address bus, it is possible that the invalidation address comes from the Am486DX/DX2 microprocessor itself.

Note that it is also possible to run an invalidation cycle by asserting $\overline{EADS}$ when HOLD or $\overline{BOFF}$ is asserted.

Running an invalidation cycle prevents the Am486DX/DX2 microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidation cycle is shown in Figure 7-22, while a more realistic invalidation cycle is shown in Figure 7-23. Both examples take one clock of cache access from the rest of the Am486DX/DX2 microprocessor.

7.2.8.1 Rate of Invalidate Cycles

The Am486DX/DX2 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 $\overline{EADS}$ is negated in ONE or BOTH of the following cases:

1. In the clock, $\overline{RDY}$ or $\overline{BRDY}$ is returned for the last time.
2. In the clock following, $\overline{RDY}$ or $\overline{BRDY}$ is being returned for the last time.

Figure 7-22   Fast Internal Cache Invalidation Cycle
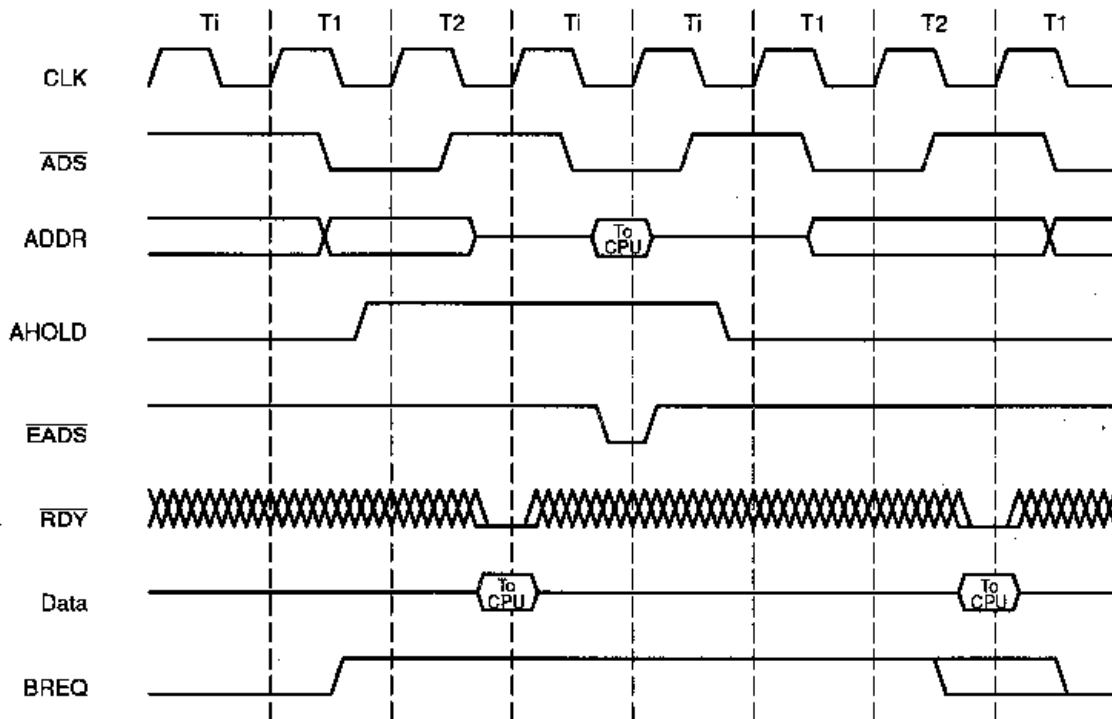
17852A–077

Figure 7-23   Typical Internal Cache Invalidation Cycle
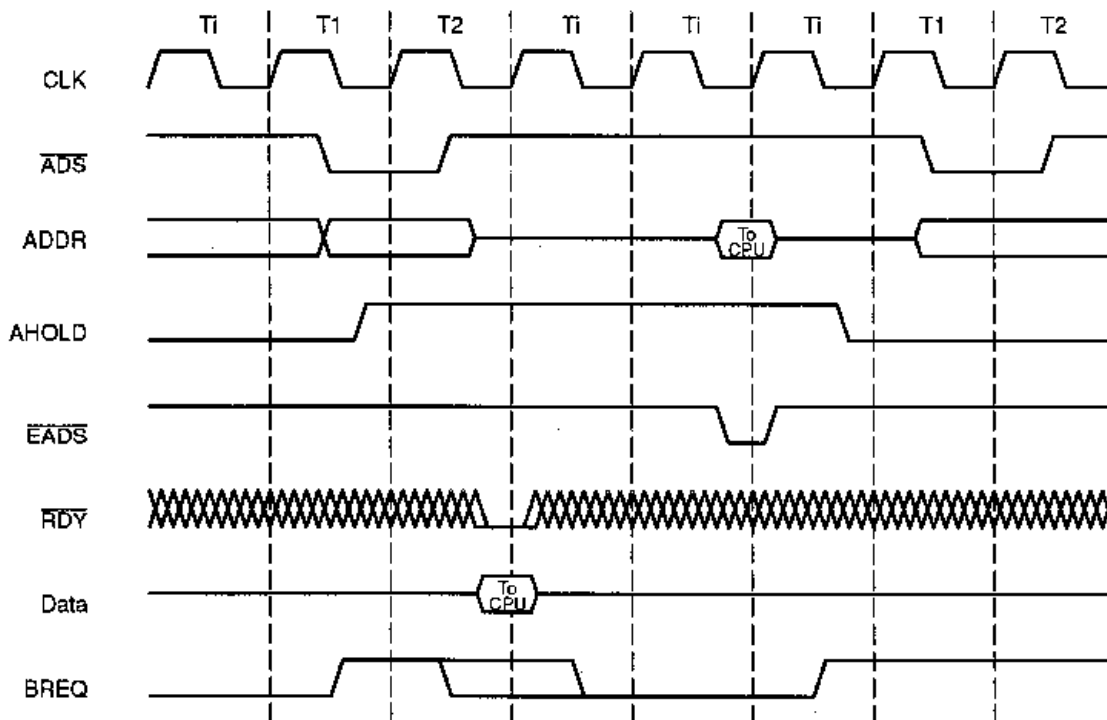
17852A–078

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 if 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 Am486DX/DX2 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 Am486DX/DX2 microprocessor is retrieving data from the second level cache. The Am486DX/DX2 microprocessor needs to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the Am486DX/DX2 microprocessor's cache.

A potential problem exists if the external device is writing to an address in external memory, and at the same time the Am486DX/DX2 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 Am486DX/DX2 microprocessor has requested during the line fill.

If the system asserts $\overline{\text{EADS}}$ before the first data in the line fill is returned to the Am486DX/DX2 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 $\overline{\text{EADS}}$ signal labeled in Figure 7-25.

If the system asserts $\overline{\text{EADS}}$ at the same time or after the first data in the line fill is returned (in the same clock that the first $\overline{\text{RDY}}$ or $\overline{\text{BRDY}}$ is returned or any subsequent clock in the line fill), the data is read into the Am486DX/DX2 microprocessor's input buffers but is not stored in the on-chip cache. This is illustrated by the asserted $\overline{\text{EADS}}$ signal labeled 2 in Figure 7-25. The stale data is used to satisfy the request that initiated the cache fill cycle.

Figure 7-24   System with Second Level Cache
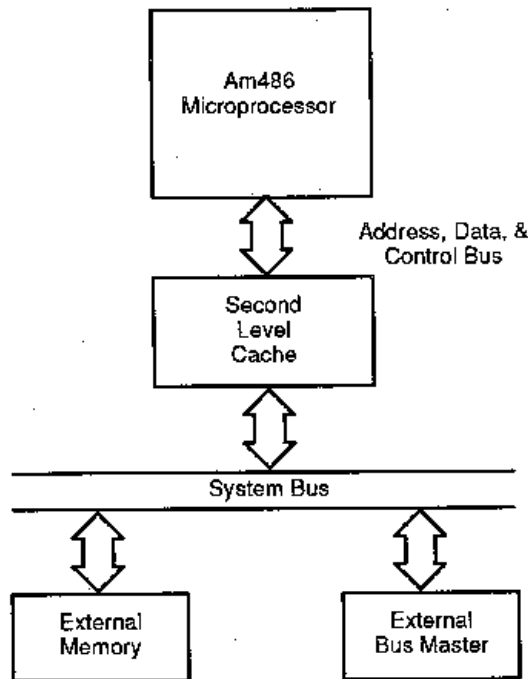
17852A–079

Figure 7-25   Cache Invalidation Cycle Concurrent with Line Fill
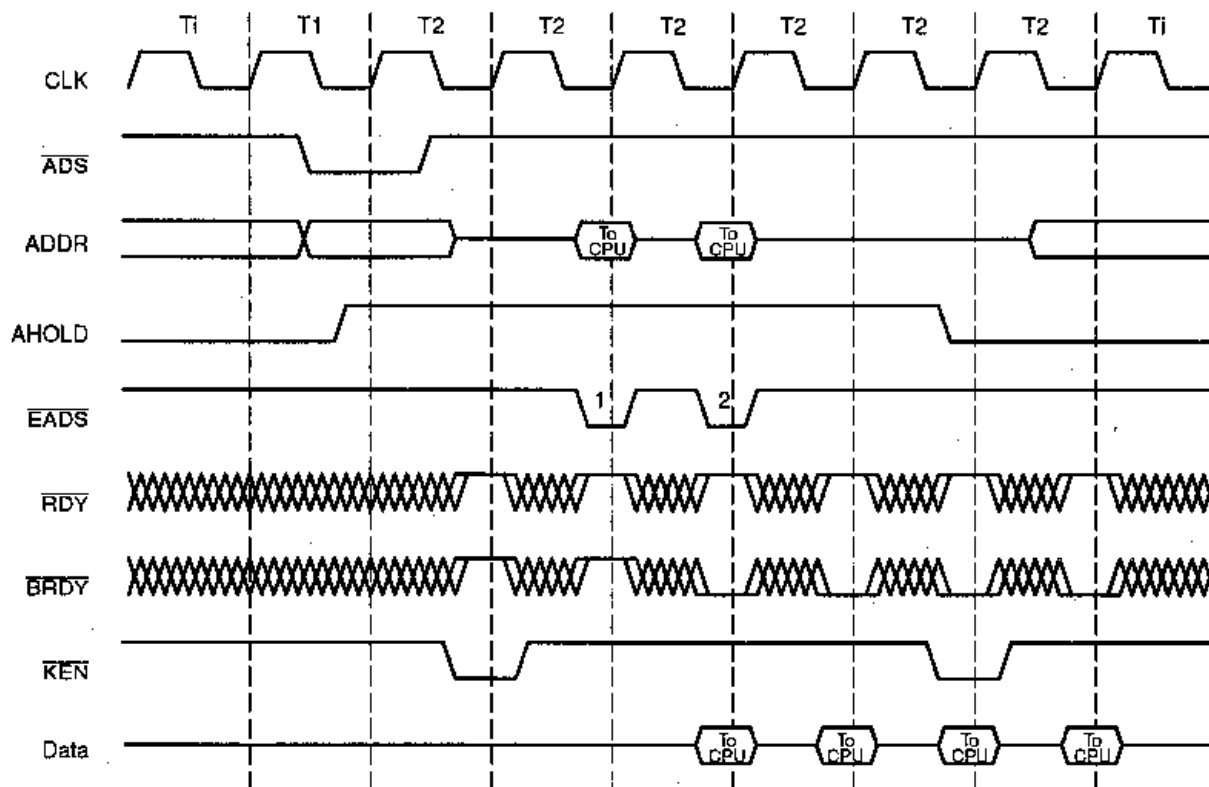
Notes:

1. Data returned must be consistent if its address equals the invalidation address in this clock.
2. Data returned is not cached if its address equals the invalidation address in this clock.

17852A—080

7.2.9   Bus Hold

The Am486DX/DX2 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 Am486DX/DX2 microprocessor's bus. The processor responds by floating its bus and driving HLDA active when the current bus cycle or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7-26. Unlike the 386 microprocessor, the Am486DX/DX2 microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD is recognized during unaligned writes (less than or equal to 32 bits) with $\overline{BLAST}$ being active for each write. For greater than 32-bit or unaligned write, HOLD recognition is prevented by $\overline{PLOCK}$ getting asserted.

The pins floated during bus hold are: $\overline{BE3}$–$\overline{BE0}$, PCD, PWT, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{PLOCK}$, $\overline{ADS}$, $\overline{BLAST}$, D31–D0, A31–A2, and DP3–DP0.

7.2.10   Interrupt Acknowledge

The Am486DX/DX2 microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests. These requests are generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

An example interrupt acknowledge transaction is shown in Figure 7-27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower eight bits of the data bus. The Am486DX/DX2 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, and BE0 Low).

Figure 7-26   HOLD/HLDA Cycles
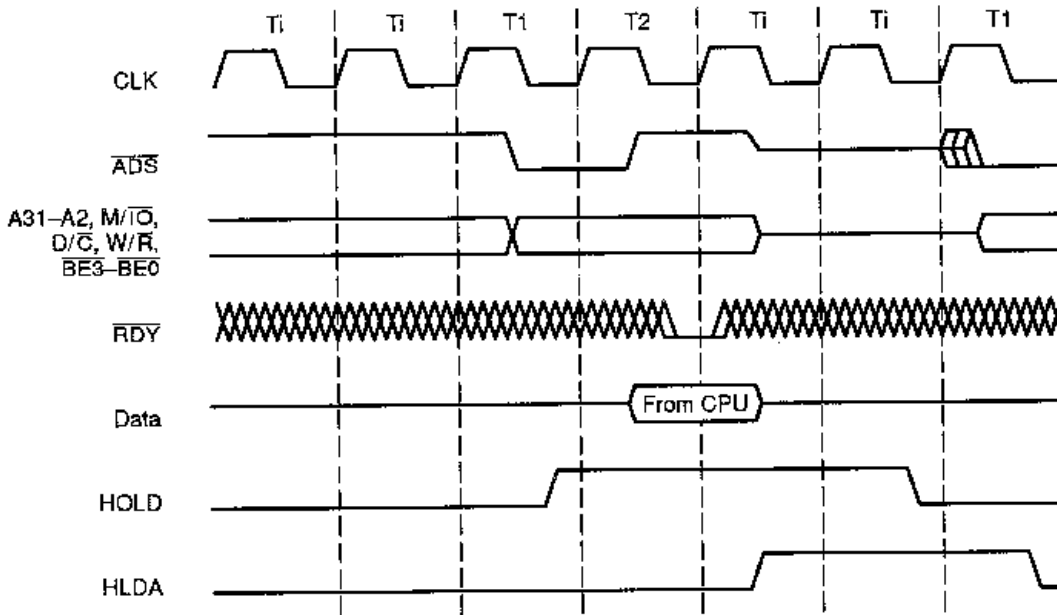
17852A–080

Figure 7-27   Interrupt Acknowledge Cycles
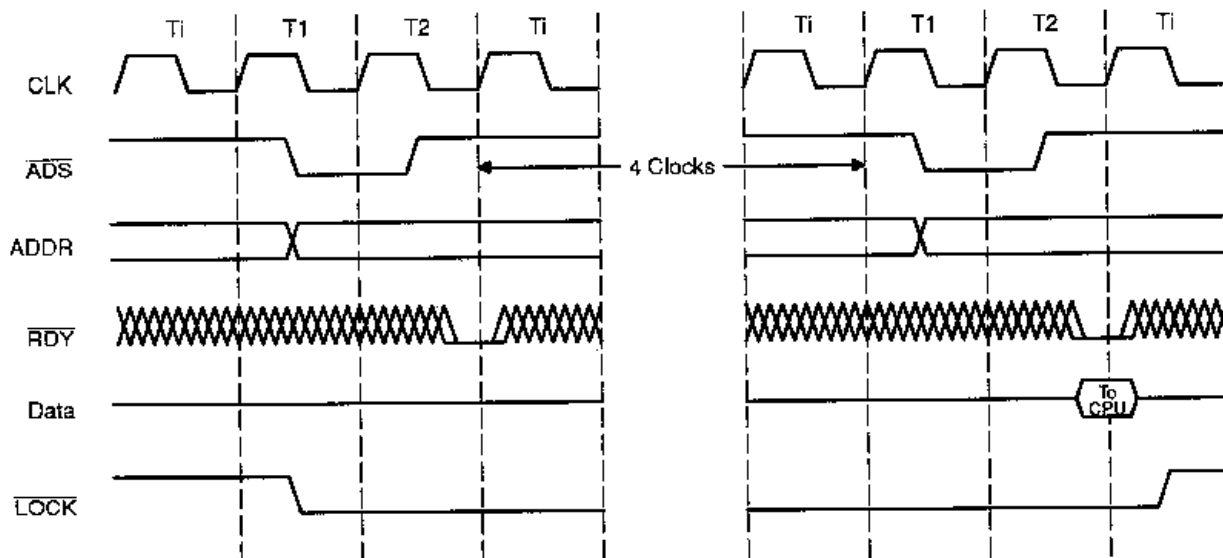
17852A–082

Table 7-9   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

Each of the interrupt acknowledge cycles are terminated when the external system returns $\overline{RDY}$ or $\overline{BRDY}$. Wait states can be added by withholding $\overline{RDY}$ or $\overline{BRDY}$. The Am486DX/DX2 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 Am486DX/DX2 microprocessor provides four special bus cycles to indicate that certain instructions were executed or certain conditions have occurred internally. The special bus cycles in Table 7-9 are defined when the bus cycle definition pins are in the following state:

$M/\overline{IO} = 0$, $D/\overline{C} = 0$ and $W/\overline{R} = 1$. 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 Am486DX/DX2 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 Am486DX/DX2 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 $\overline{RDY}$ or $\overline{BRDY}$.

7.2.11.1   Halt Indication Cycle

The Am486DX/DX2 microprocessor halts as the result of 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. $\overline{BE0}$ and $\overline{BE2}$ 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 D31–D0. The halt indication cycle must be acknowledged by $\overline{RDY}$ or $\overline{BRDY}$ asserted.

A halted Am486DX/DX2 microprocessor resumes execution when INTR (if interrupts are enabled), NMI, or RESET is asserted.

7.2.11.2   Shutdown Indication Cycle

The Am486DX/DX2 microprocessor shuts down as the 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 multimaster system, another bus master can require the use of the bus to enable the Am486DX/DX2 microprocessor to complete its current bus request. In this situation the

Am486DX/DX2 microprocessor needs to restart its bus cycle after the other bus master completes its bus transaction.

A bus cycle can be restarted if the external system asserts the backoff (BOFF) input. The Am486DX/DX2 microprocessor samples the BOFF pin every clock. The Am486DX/DX2 microprocessor immediately (in the next clock) floats 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 that are floated in response to HOLD are floated in response to BOFF. HLDA is not generated in response to BOFF. BOFF has higher priority than RDY or BRDY. If either RDY or BRDY is returned in the same clock as BOFF, BOFF takes effect.

The device asserting BOFF is free to run any cycles it wants while the Am486DX/DX2 microprocessor bus is in its high impedance state. If backoff is requested after the Am486DX/DX2 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 ensure 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 Am486DX/DX2 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 forces the Am486DX/DX2 microprocessor to ignore data returned for that cycle only. Data from previous cycles is still valid. For example, if BOFF is asserted on the third BRDY of a burst, the Am486DX/DX2 microprocessor assumes the data returned with the first and second BRDYs 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 causes the Am486DX/DX2 microprocessor to float its bus in the next clock and leaves ADS floating Low. Since ADS is floating Low, a peripheral might think a new bus cycle has begun, even though the cycle is aborted. There are two possible solutions to this problem. The first is for all devices to recognize this condition and ignore ADS until RDY 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 is not floating Low. This is only necessary in systems where BOFF can 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 7-10.

7.2.14   Floating-Point Error Handling

The Am486DX/DX2 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 can be enabled by setting the NE bit in control register 0 (CR0).

The Am486DX/DX2 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 Am486DX/DX2 microprocessor asserts the FERR output to indicate that a floating-point error has occurred. FERR corresponds to the ERROR pin on the 387 math coprocessor. 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, 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).

Figure 7-28   Restarted Read Cycle
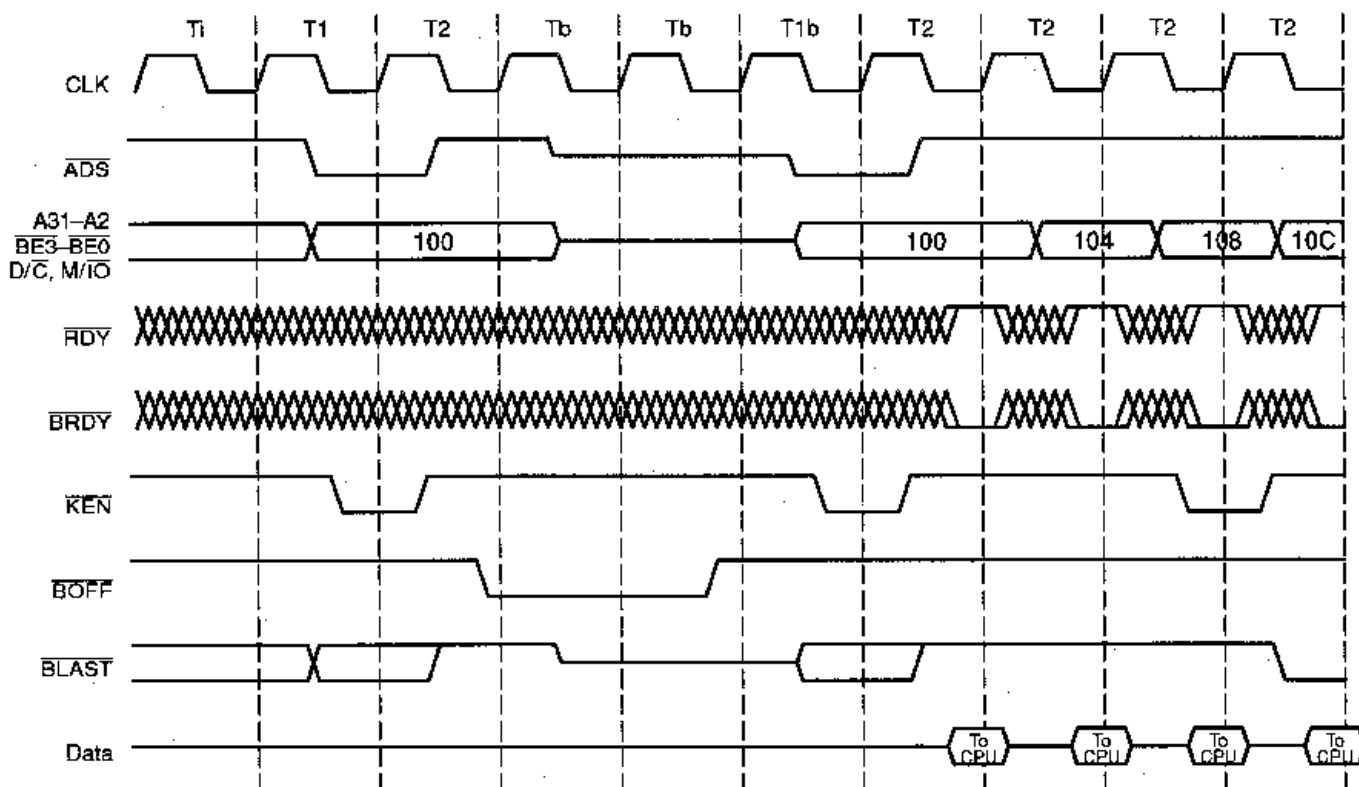
17852A–083

Figure 7-29   Restarted Write Cycle

17852A–084
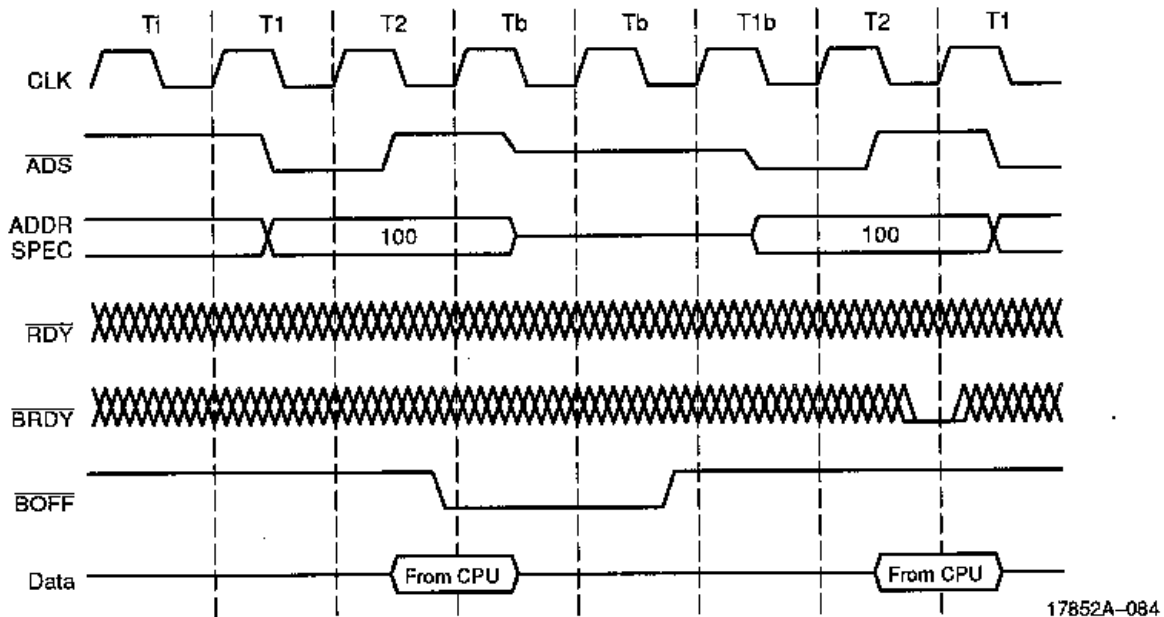
Table 7-10   Bus State Description

State	Means
Ti	Bus is idle. Address and status signals can be driven to undefined values, or the bus can be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and $\overline{ADS}$ 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. $\overline{RDY}$ and $\overline{BRDY}$ are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and $\overline{ADS}$ is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

For both sets of exceptions above, the 387 math coprocessor asserts $\overline{ERROR}$ when the error occurs and does not wait for the next floating-point instruction to be encountered.

$\overline{IGNNE}$ is an input to the Am486DX/DX2 microprocessor.

When the NE bit in CR0 is cleared and $\overline{IGNNE}$ is asserted, the Am486DX/DX2 microprocessor ignores a user floating-point error and continues executing floating-point instructions. When $\overline{IGNNE}$ is negated, the Am486DX/DX2 microprocessor freezes on floating-point instructions that get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI, and FNSETPM). $\overline{IGNNE}$ can be asynchronous to the Am486DX/DX2 microprocessor clock.

In systems with user-defined error reporting, the $\overline{FERR}$ pin is connected to the interrupt controller. When an unmasked floating-point error occurs, an interrupt is raised. If $\overline{IGNNE}$ is High at the time of this interrupt, the Am486DX/DX2 microprocessor freezes (disallowing execution of a subsequent floating-point instruction) until the interrupt handler is invoked. By driving the $\overline{IGNNE}$ pin Low (when clearing the interrupt request), the interrupt handler allows 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

AMD ▯

Figure 7-30   Bus State Diagram

$(\overline{RDY}\ Asserted + (\overline{BRDY} \bullet \overline{BLAST})\ Asserted) \bullet$
$(HOLD + AHOLD + No\ Request) \bullet$
$\overline{BOFF}\ Negated$

Request Pending •
$(\overline{RDY}\ Asserted + (\overline{BRDY} \bullet \overline{BLAST})\ Asserted) \bullet$
HOLD Negated•
AHOLD Negated •
$\overline{BOFF}$ Negated

Request Pending •
HOLD Negated•
AHOLD Negated•
$\overline{BOFF}$ Negated

Ti   T1   T2

$\overline{BOFF}$ Negated

$\overline{BOFF}$ Asserted

$\overline{BOFF}$ Asserted $\overline{BOFF}$ Negated

$\overline{BOFF}$ Asserted

T$_b$ T1b
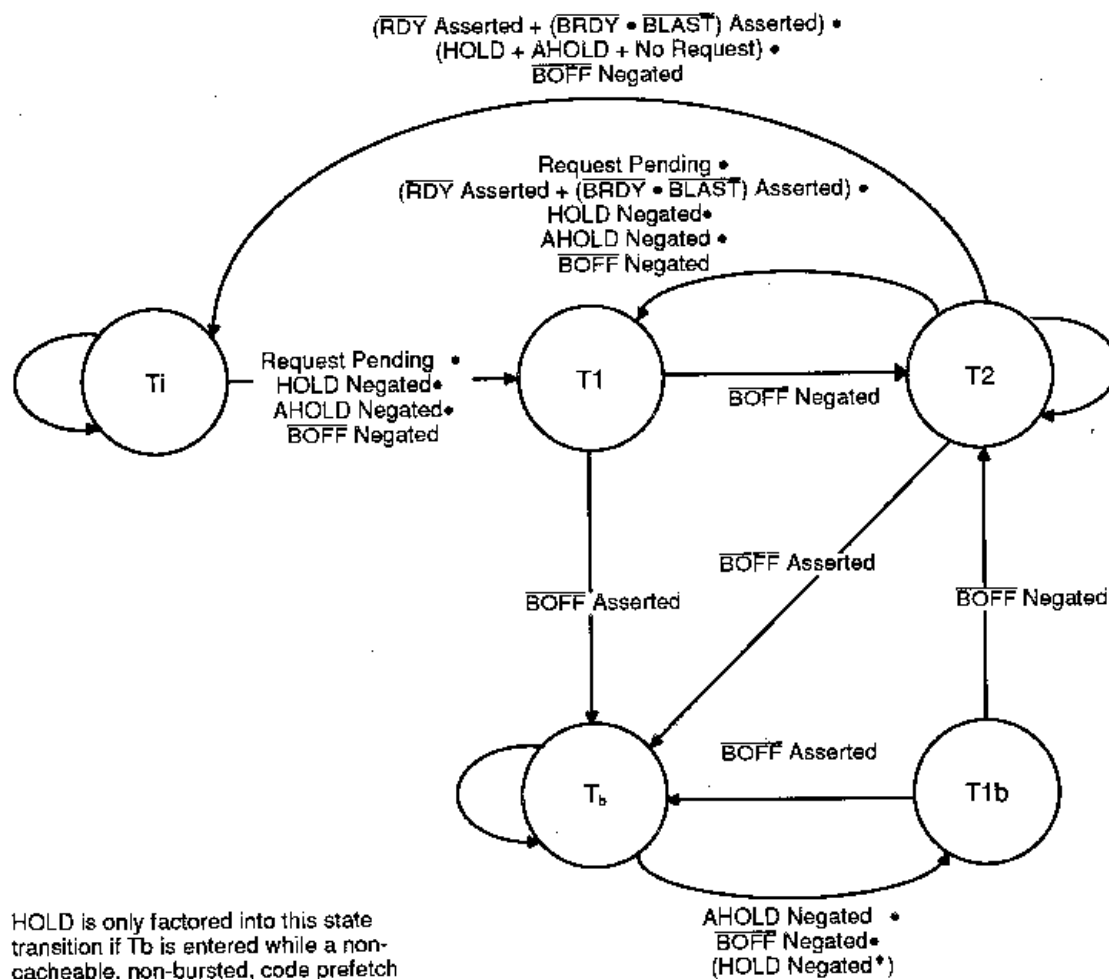
* HOLD is only factored into this state
  transition if Tb is entered while a non-
  cacheable, non-bursted, code prefetch
  is in progress. Otherwise, ignore
  HOLD.

AHOLD Negated •
$\overline{BOFF}$ Negated•
(HOLD Negated*)

17852A–085

floating-point instruction within the floating-point interrupt handler is unnecessary, the $\overline{IGNNE}$ pin can be tied High.

7.2.15   Floating-Point Error Handling In AT Compatible Systems

The Am486DX microprocessor provides special features to allow the implementation of an AT compatible numerics error reporting scheme. These features DO NOT replace the external circuit. Logic is still required that decodes the OUT F0 instruction and latches the $\overline{FERR}$ signal. What follows is a description of the use of these Am486DX/DX2 microprocessor features.

The features provided by the Am486DX/DX2 microprocessor are the NE bit in the Machine Status Register, the $\overline{IGNNE}$ pin, and the $\overline{FERR}$ pin.

The NE bit determines the action taken by the Am486DX/DX2 microprocessor when a numerics error is detected. When set, this bit signals that non-DOS compatible error handling will be implemented. In this mode the Am486DX/DX2 microprocessor takes a software exception (16) if a numerics error is detected.

If the NE bit is reset, the Am486DX/DX2 microprocessor uses the $\overline{IGNNE}$ pin to allow an external circuit to control the time at which non-control numerics instructions are allowed

to execute. Note that floating-point control instructions such as FNINIT and FNSAVE can be executed during a floating-point error condition regardless of the state of $\overline{\text{IGNNE}}$.

To process a floating-point error in the DOS environment, the following sequence must take place:

1. The error is detected by the Am486DX/DX2 microprocessor which activates the $\overline{\text{FERR}}$ pin.

2. $\overline{\text{FERR}}$ is latched so that it can be cleared by the OUT F0 instruction.

3. The latched $\overline{\text{FERR}}$ signal activates an interrupt at the interrupt controller. This interrupt is usually handled on IRQ13.

4. The Interrupt Service Routine (ISR) handles the error and then clears the interrupt by executing an OUT instruction to port F0. The address F0 is decoded externally to clear the $\overline{\text{FERR}}$ latch. The $\overline{\text{IGNNE}}$ signal is also activated by the decoder output.

5. Usually the ISR then executes an FNINIT instruction or other control instruction before restarting the program. FNINIT clears the $\overline{\text{FERR}}$ output.

Figure 84 illustrates the circuit required to perform this function. Note that this circuit has not been tested. It is included as an example of the required error handling logic.

Note that the $\overline{\text{IGNNE}}$ input allows non-control instructions to be executed prior to the time the $\overline{\text{FERR}}$ signal is reset by the Am486DX/DX2 microprocessor. This function is implemented to allow exact compatibility with the AT implementation. Most programs reinitialize the floating-point unit before continuing after an error is detected. The floating point unit can be reinitialized using one of the following four instructions: FCLEX, FINIT, FSAVE, FSTENV.

Figure 7-31   DOS Compatible Numerics Error Circuit

8 Am486DX/DX2 CPU TESTABILITY
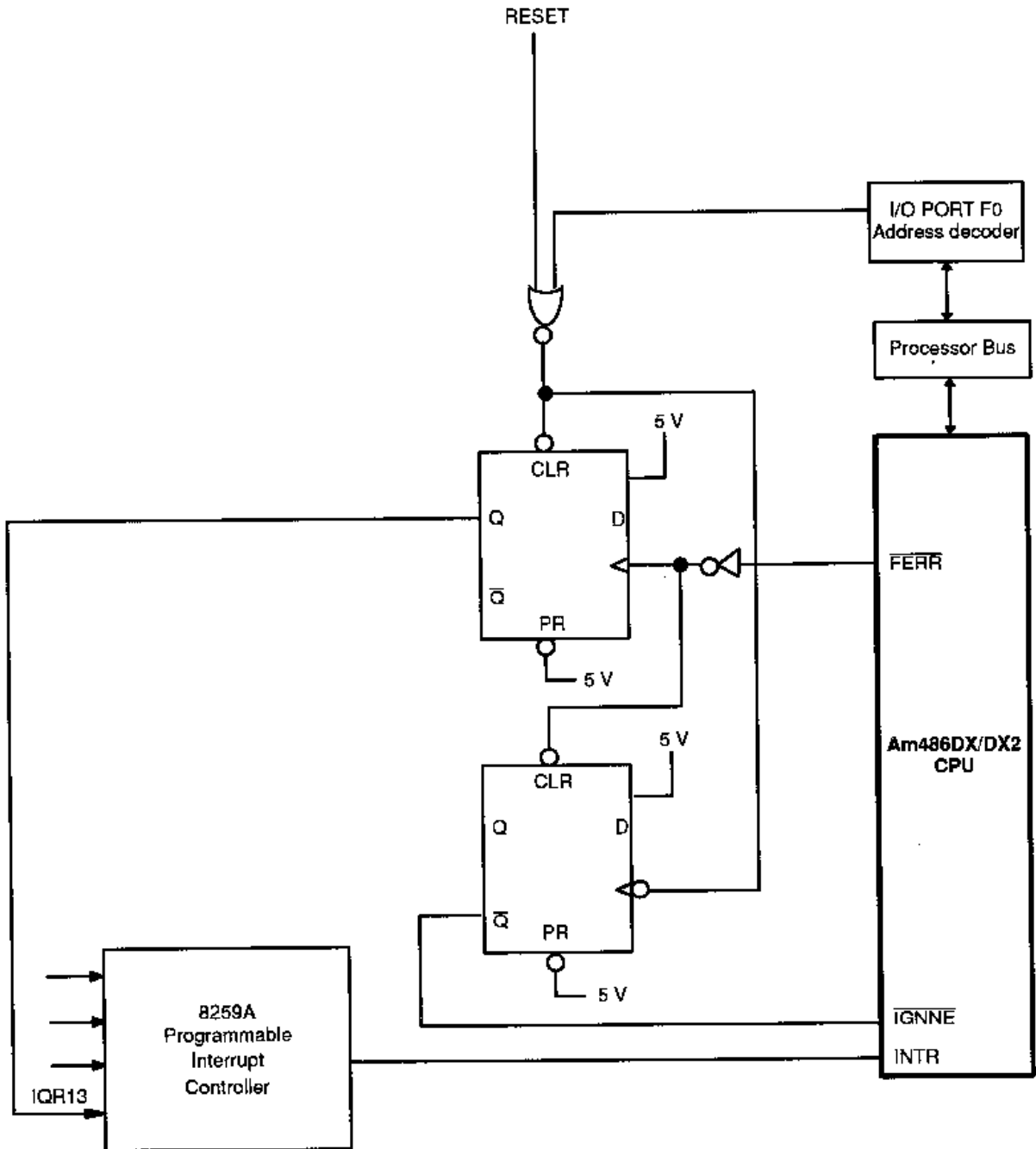
Testing the Am486DX/DX2 microprocessor can be divided into three categories: Built-In Self Test (BIST), Boundary Scan, and external testing. BIST performs basic device testing on the Am486DX/DX2 CPU, including the non-random logic, control ROM (CROM), translation lookaside buffer (TLB), and on-chip cache 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 Am486DX/DX2 microprocessor also has a test mode in which all of its outputs are three-stated. Additional testing can be performed by using the test registers within the Am486DX/DX2 CPU.

8.1 BUILT-IN SELF TEST (BIST)

The BIST is initiated by asserting AHOLD (address hold) on the falling edge of RESET. AHOLD is a synchronous signal only. It should be asserted in the clock prior to RESET going from High to Low to start BIST. FLUSH must also be asserted (driven Low) prior to the falling edge of RESET to start BIST. FLUSH must be deasserted (driven High) during BIST. A20M must be deasserted (driven High) during the falling edge of RESET to start BIST. The BIST takes approximately $2^{**}20$ clocks, or approximately 42 milliseconds with a 25-MHz Am486DX/DX2 microprocessor. No bus cycles are run by the Am486DX/DX2 microprocessor until the BIST is concluded. Note that for the Am486DX/DX2 microprocessor, the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST are stored in the EAX register. The Am486DX/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 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 Am486DX/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.

8.2.1.1 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 accesses in the buffers.

8.2.1.2 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, as well as the LRU bits and four valid bits from the accessed set.

8.2.1.3 Cache Control Test Register: TR5

TR5 specifies which testability operation is performed and the set and entry within the set that is accessed.

The 7-bit set select field determines which of the 128 sets is accessed.

The functionality of the two entry select bits depends 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 (see Table 8-1).

Figure 8-1   Cache Test Registers

= unused
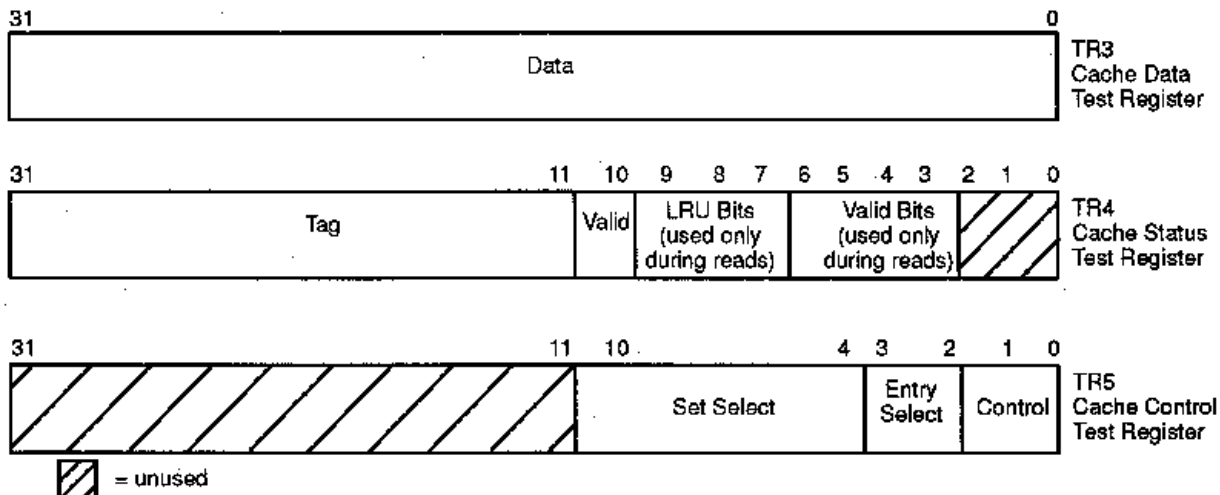
Table 8-1 **Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality**

Control Bits		Operation	Entry Select Bits Functions	Set Select Bits
Bit 1	**Bit 0**	Enable ⎡ Fill Buffer Write Read	Select 32-bit location in fill/ read buffer.	–
0	0	⎣ Buffer Read		
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	–	–

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 the 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. 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.

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 that 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 indicates 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 CO bit in 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 that 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 be 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 initiates 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 are corrupted. This occurs because the testability operations use hardware that is used in normal memory accesses for the Am486DX/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 resets the LRU bits and the valid bits to 0, but does 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 INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 TLB TESTING

The Am486DX/DX2 microprocessor TLB testability hooks are similar to those in the 386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the Am486DX/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 Am486DX/DX2 microprocessor's TLB is four-way set associative and has space for eight entries. The TLB is logically split into three blocks (see 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

Figure 8-2   Sample Assembly Code for Cache Testing

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
```

17852A–087

PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

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 LRU replacement algorithm. These bits are the LRU bits. The LRU replacement algorithm used in the TLB

is the same as that 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.

8.3.2.1   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.

Figure 8-3   TLB Organization
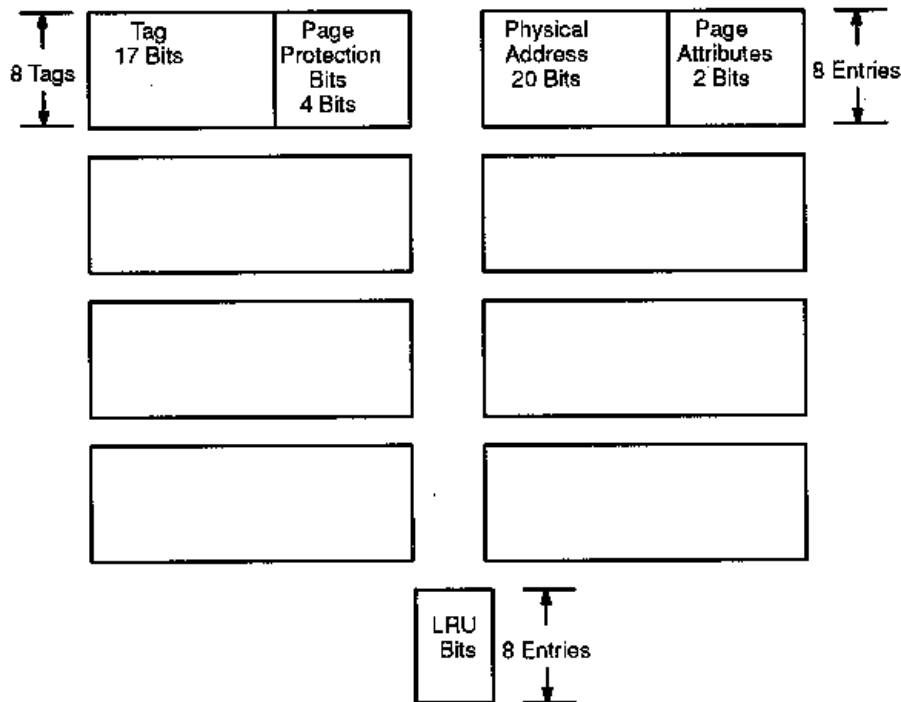
17852A–088

Figure 8-4   TLB Test Registers
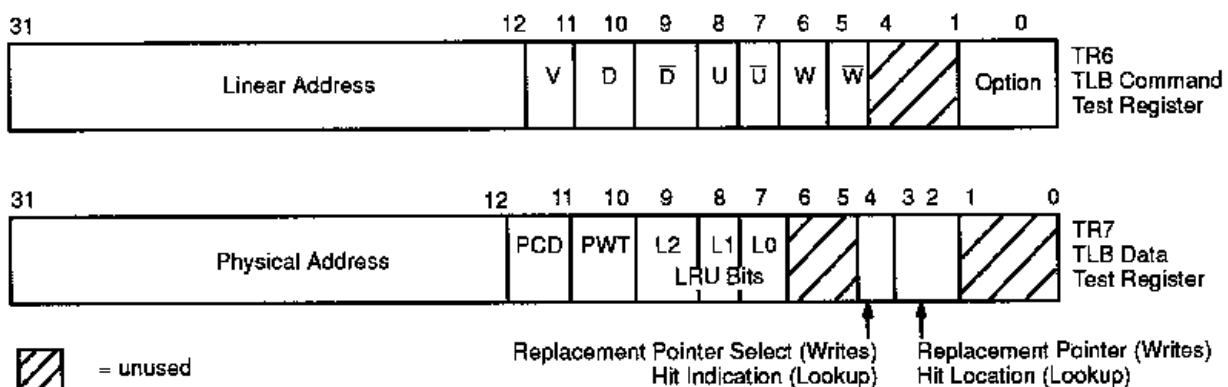
17852A–089

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, $\overline{D}$: The dirty bit for/from the TLB entry

U, $\overline{U}$: The user/supervisor bit for/from the TLB entry

W, $\overline{W}$: 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 occurs 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 is a write or a lookup. The function of the operation bit is given in Table 8-3.

8.3.2.2 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 31–12), two bits for PCD (bit 11) and PWT (bit 10), and three bits for the LRU bits (bits 9–7). 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 Table 8-4 and Table 8-5. Finally, TR7 contains two bits (bits 3–2) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8-2 Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit# (B#)	Meaning of TLB Write Operation	Meaning of 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

Table 8-3 TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

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 in 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.

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the 386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address does 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 0 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 are 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 indicates 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 contains the following information if bit 4 indicates that the lookup test resulted in a hit. Bits 2–3 indicate in which set the match occurred. The 22 most significant bits in TR7 contain the physical address and page attributes contained in the entry.

Bits 9–7 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 are updated if a hit occurred. The entry that was hit becomes the most recently used.

8.4 THREE-STATE OUTPUT TEST MODE

The Am486DX/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 that are never floated in normal operation of the chip (HLDA, $\overline{BREQ}$, $\overline{FERR}$, and $\overline{PCHK}$). When the Am486DX/DX2 microprocessor is in the three-state output test mode, external testing can be used to test board connections.

The three-state test mode is invoked by driving $\overline{FLUSH}$ Low for two clocks before and two clocks after RESET goes Low. The outputs are guaranteed to three-state no later than ten clocks after RESET goes Low (see Figure 6-4). The Am486DX/DX2 microprocessor remains in the three-state test mode until the next RESET.

8.5 Am486DX/DX2 MICROPROCESSOR BOUNDARY SCAN (JTAG)

The Am486DX/DX2 microprocessor provides additional 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 ensure 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:

■ 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 Am486DX/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 that 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 Am486DX/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 Am486DX/DX2 CPU contains a RUNBIST register to support the RUNBIST boundary scan instruction.

8.5.2.1 Bypass Register (BPR)

The BPR 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 BPR is selected, data is transferred from TDI to TDO without inversion.

8.5.2.2 Boundary Scan Register (BSR)

The BSR is a single shift register path containing the boundary scan cells that are connected to all input and output pins of the Am486DX/DX2 CPU. Figure 8-5 shows the logical structure of the BSR. 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 BSR during scanning. The BSR can be operated by the EXTEST and SAMPLE instructions. The boundary scan register order is described in Section 8.5.5.

8.5.2.3 Device Identification Register (DID)

The DID contains the manufacturer's identification code, part number code, and version code in the format shown in Figure 8-6. Table 8-6 lists the codes corresponding to the Am486DX/DX2 CPU.

8.5.2.4 RUNBIST Register

The RUNBIST register is a one-bit register used to report the results of the Am486DX/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.

Table 8-6   Component Codes

Component Code	Version Code	Part Number Code	Manufacturer Identity
Am486 CPU (Ax)	00H	0410H	09H
Am486 CPU (Bx)	00H	0411H	09H

Figure 8-5   Logical Structure of Boundary Scan Register
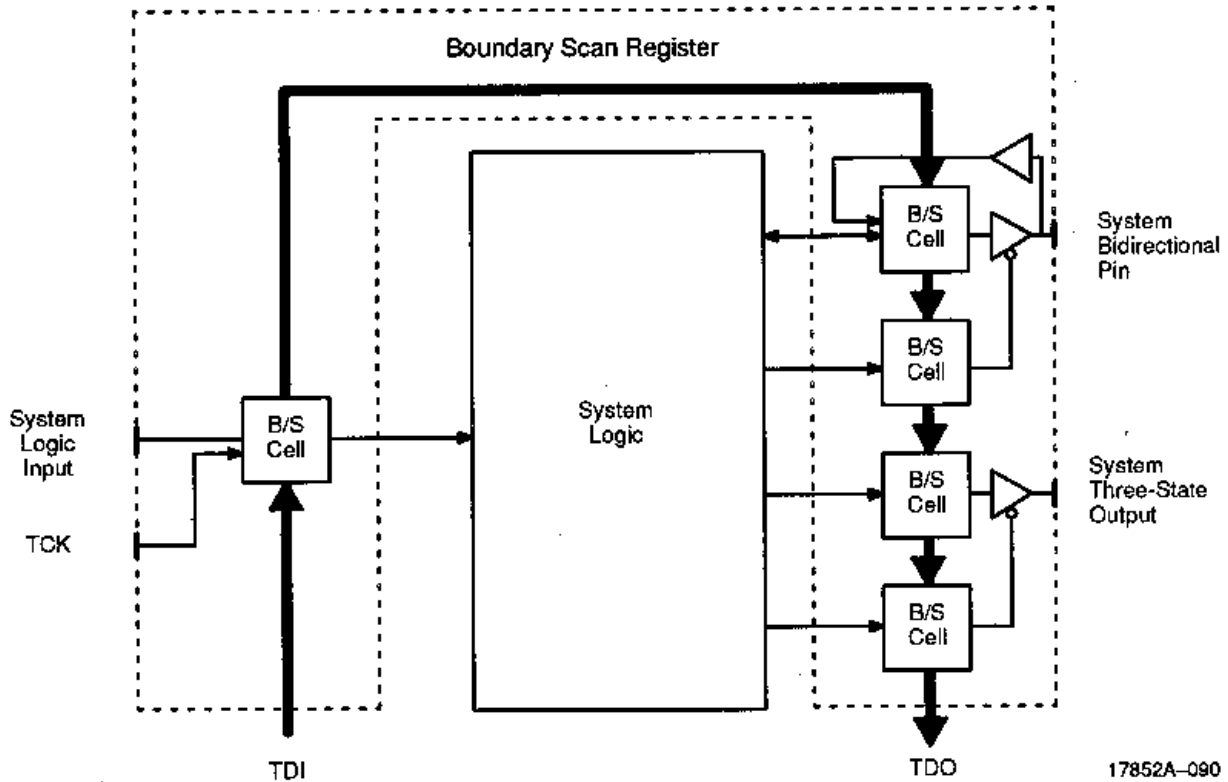
17852A–090

Figure 8-6   Format of Device Identification Register

31 30 29 28	27 26 25 24 23 22 21 20 19 18 17 16 15 14 13	12 11 10 9 8 7 6 5 4 3 2 1	0
Version	Part Number	Manufacturer Identity	1
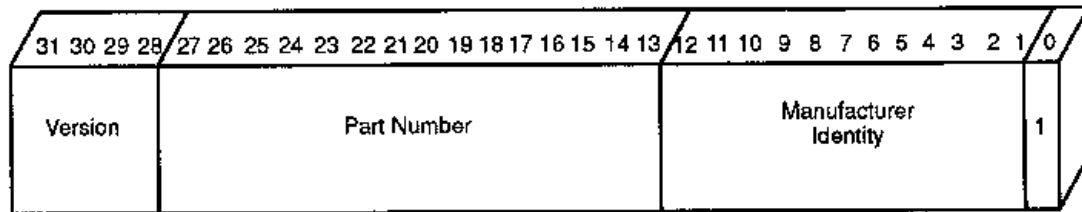
17852A–091

8.5.3   Instruction Register

The 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 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 IR. 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   Am486DX/DX2 CPU Boundary Scan Instruction Set

The Am486DX/DX2 CPU supports all three mandatory boundary scan instructions (BYPASS, SAMPLE/PRELOAD, and EXTEST) along with two optional instructions (IDCODE and RUNBIST). Table 8-7 lists the Am486DX/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 does not cause hazardous operation of the Am486DX/DX2 CPU.

Table 8-7 **Boundary Scan Instruction Codes**

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 Am486DX/DX2 CPU's BSR out on the output pins corresponding to each boundary scan cell. It then captures the values on Am486DX/DX2 CPU input pins to be loaded into their corresponding BSR locations. I/O pins are selected as input RUNBIST or output depending on the value loaded into their control setting locations in the BSR. Values shifted into input latches in the BSR are never used by the internal logic of the Am486DX/DX2 CPU.

Note:

After using the EXTEST instruction, the Am486DX/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 "snapshot" of the normal operation of the component without interfering with that normal operation. The instruction causes BSR cells associated with outputs to sample the value being driven by the Am486DX/DX2 CPU. It causes the cells associated with inputs to sample the value being driven into the Am486DX/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 BSR on the falling edge of TCK.

IDCODE

The instruction code is "0010". The IDCODE instruction selects the DID to be connected to TDI and TDO, allowing the device identification code to be shifted out of the device on TDO. Note that the DID 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 Am486DX/DX2

microprocessor by reducing the shift length of the device to one bit. Note that an open circuit fault in the board level test data path causes 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 Am486DX/DX2 CPU, which is able to detect approximately 60% of the stuck-at faults on the Am486DX/DX2 CPU. The Am486DX/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 CLK cycles to complete BIST and report the result to the runbist register. After completing the 1.2 million 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. After executing the RUNBIST instruction, the Am486DX/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 8-7. 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

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

Figure 8-7 TAP Controller State Diagram
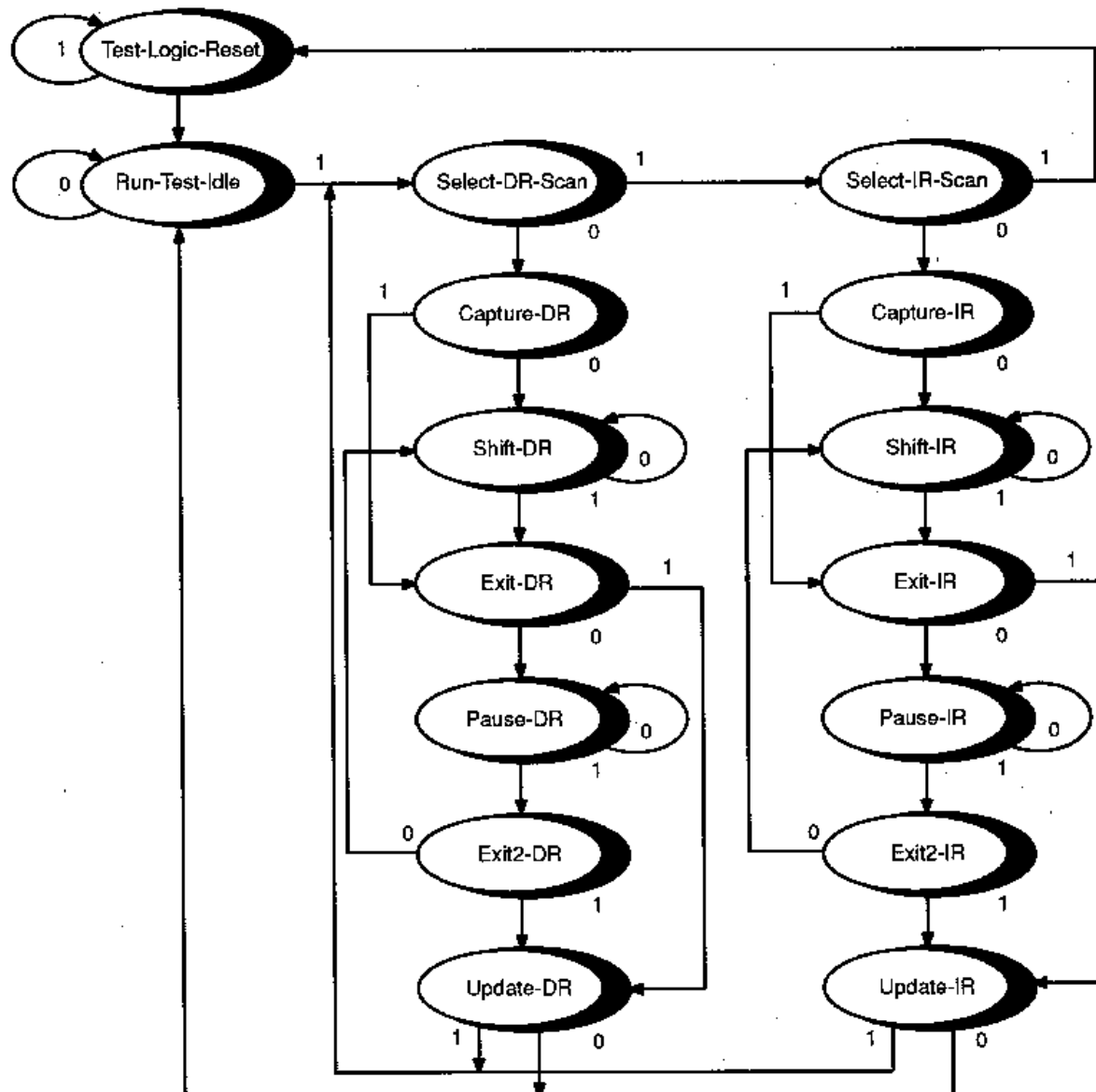
17852A–092

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 BSR 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 BSR 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 BSR 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 retain their previous values 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 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 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 BSR contains a cell for each pin, as well as cells for control of I/O and three-state pins.

The following is the bit order of the Am486DX/DX2 CPU BSR: (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 W/R BREQ BE0
BE1 BE2 BE3 M/IO D/C PWT PCD
EADS A20M RESET FLUSH INTR NMI
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 Am486DX/DX2 CPU.

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

1. WRCTL controls the D31–D0 and DP3–DP0 pins.

2. ABUSCTL controls the A31–A2 pins.

3. BUSCTL controls the $\overline{ADS}$, $\overline{BLAST}$, $\overline{PLOCK}$, $\overline{LOCK}$, W/R, $\overline{BE0}$, $\overline{BE1}$, $\overline{BE2}$, $\overline{BE3}$, M/IO, D/C, PWT, and PCD pins.

4. MISCCTL controls the $\overline{PCHK}$, HLDA, and BREQ 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.

9 DEBUGGING SUPPORT

The Am486DX/DX2 microprocessor provides several features that 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

3. The code and data breakpoint capability provided by the Debug Registers DR3–DR0, 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 EFLAGS 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 that 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 Am486DX/DX2 microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The Am486DX/DX2 microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoint addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoint are in the disabled state. Therefore, no breakpoints 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 (DR3–DR0)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR3–DR0, shown in Figure 9-1. The breakpoint addresses specified are 32-bit linear addresses. Am486DX/DX2 microprocessor hardware continuously compares the linear breakpoint addresses in DR3–DR0 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. However, regardless of whether paging is enabled or not, the breakpoint registers hold linear addresses.

9.3.2 Debug Control Register (DR7)

A Debug Control Register, DR7, (see 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 DR7 are as follows:

9.3.2.1 Breakpoint Length Specification Bits (LENi)

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 shown in Figure 9-1.

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 (see Table 9-1).

Figure 9-2 is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, Figure 9-2 indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.

Figure 9-1   Debug Registers
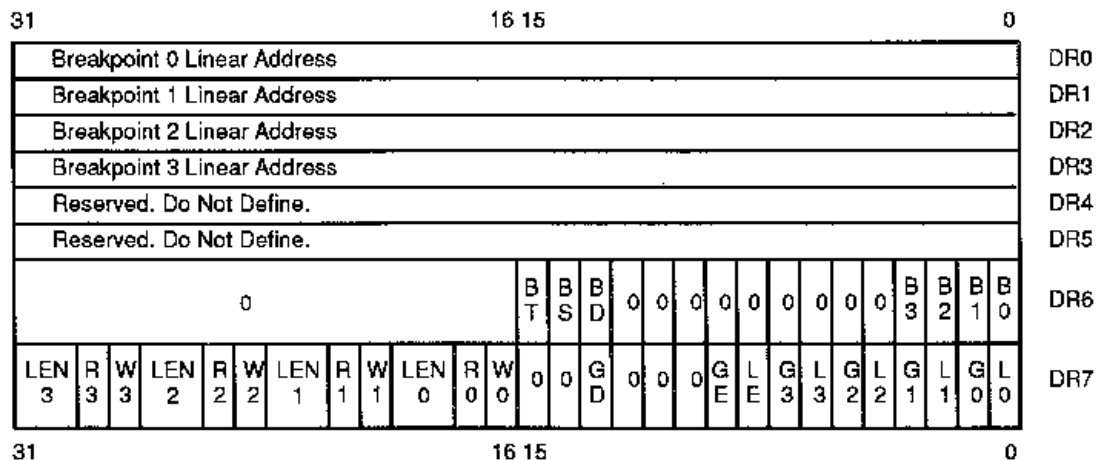
Note:
0 indicates AMD reserved: Do not define: see Section 2.3.10

17852A–093

Table 9-1   Debug Registers LENi Encoding

LENi Encoding	Breakpoint Field Width	Use 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	A31–A1 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	A31–A2 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

Figure 9-2   Debug Registers Breakpoint Fields
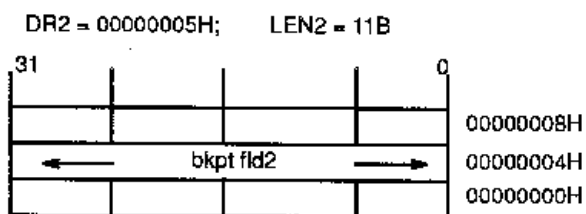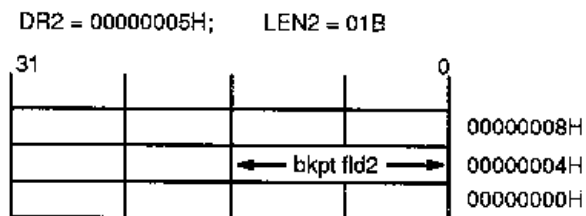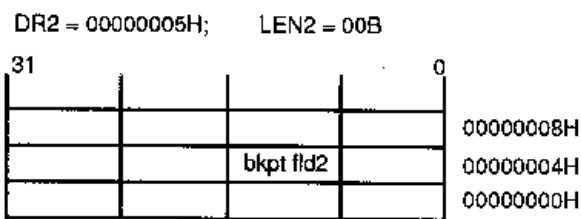
17852A–093

9.3.2.2   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 that must occur in order to activate the associated breakpoint (see Table 9-2).

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: 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).

Table 9-2   Debug Registers RW Encoding

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

9.3.2.3   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 falls entirely or partly within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap then occurs.

9.3.2.4   Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing the 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 occurs before the instruction is executed.

Note: An instruction execution breakpoint address must equal the beginning byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

9.3.2.5   Global Debug Register Access Detect (GD)

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 guarantees 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.

9.3.2.6   Exact Data Breakpoint Match Global (GE) and Exact Data Breakpoint Match Local (LE)

The breakpoint mechanism of the Am486DX/DX2 microprocessor differs from that of the Am386 CPU. The Am486DX/DX2 microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap is reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Am486DX/DX2 microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the Am486DX/DX2 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 Am486DX/DX2 microprocessor GE bit is unaffected during a task switch. The GE bit supports the exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note: Instruction execution breakpoints are always reported exactly.

9.3.2.7   Breakpoint Enable Global (Gi) and Breakpoint Enable Local (Li)

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 Am486DX/DX2 microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the Am486DX/DX2 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 Am486DX/DX2 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, (see 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 (exceptions taken before the instruction is executed), while other events are traps (exceptions 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 DR6 are as follows:

9.3.3.1   Debug Fault/Trap Due to Breakpoint 0–3 (Bi)

Four breakpoint indicator flags, B3–B0, correspond one-to-one with the breakpoint registers in DR3–DR0. 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 invokes 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.

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 can 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.

9.3.3.2 Debug Fault Due to Attempted Register Access when GD Bit Set (BD)

This bit is set if the exception 1 handler is 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.

9.3.3.3 Debug Trap Due to Single-Step (BS)

This bit is set if the exception 1 handler is invoked due to the TF bit in the flag register being set (for single-stepping).

9.3.3.4 Debug Trap Due to Task Switch (BT)

This bit is set if the exception 1 handler is invoked due to a task switch occurring to a task having an Am486DX/DX2 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. This occurs when the exception 1 handler returns to a user program at a user address that is also an instruction execution breakpoint.

10 INSTRUCTION SET SUMMARY

This section describes the Am486DX/DX2 microprocessor instruction set. Table 10-1 through Table 10-5 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 Am486DX/DX2 microprocessor instructions.

10.1 MICROPROCESSOR INSTRUCTION ENCODING AND CLOCK COUNT SUMMARY

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 10-1 through Table 10-5, by the processor clock period.

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 the exact encodings of all fields contained within the instruction.

10.1.1 Instruction Clock Count

The Am486DX/DX2 microprocessor instruction clock count tables give clock counts, assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss forces the Am486DX/DX2 microprocessor to run an external bus cycle. The Am486DX/DX2 microprocessor 32-bit burst bus is defined as r-b-w.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

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

w = The number of clocks for a write.

The fastest bus the Am486DX/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 buses, add r-2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

10.1.2 Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else, add clocks to reads until the bus is available.

2. Accesses are aligned. Add three clocks to each misaligned access.

3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or 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 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, one 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 clocks. If the destination instruction is not completely contained in the first 16-byte burst, add a maximum of another r+3b clocks.

7. If no write buffer delay, w clocks are added only in the case in which all write buffers are full. This case rarely occurs.

8. Displacement and immediate are not used together. If displacement and immediate are used together, one clock can be added to the clock count shown.

9. No invalidate cycles. Add a delay of one clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the Am486DX/DX2 CPU needs to use it.

10. Page translation hits in TLB. A TLB miss adds 13, 21, or 28 clocks to the instruction, depending on whether the accessed and/or dirty bit in neither, one, or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.

11. No exceptions are detected during instruction execution. Refer to Table 10-3 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 might be necessary and might add up to (r+3b) clocks to the cache miss penalty.

Table 10-1 Am486DX/DX2 Microprocessor Integer Clock Count Summary

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
INTEGER OPERATIONS						
MOV = Move:						
reg1 to reg2	1 0 0 0 1 0 0 W	1 1 reg1 reg2		1		
reg2 to reg1	1 0 0 0 1 0 1 w	1 1 reg1 reg2		1		
memory to reg	1 0 0 0 1 0 1 w	mod reg r/m		1	2	
reg to memory	1 0 0 0 1 0 0 w	mod reg r/m		1		
Immediate to reg	1 1 0 0 0 1 1 w	1 1 0 0 0 reg	immediate data	1		
or	1 0 1 1 w reg	immediate data		1		
Immediate to Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	displacement immediate	1		
Memory to Accumulator	1 0 1 0 0 0 0 w	full displacement		1	2	
Accumulator to Memory	1 0 1 0 0 0 1 w	full displacement		1		
MOVSX/MOVZX = Move with Sign/Zero Extension						
reg2 to reg1	0 0 0 0 1 1 1 1	1 0 1 1 z 1 1 w	1 1 reg1 reg2	3		
Memory to reg	0 0 0 0 1 1 1 1	1 0 1 1 z 1 1 w	mod reg r/m	3	2	
z Instruction						
0 MOVZX						
1 MOVSX						
PUSH = Push						
reg	1 1 1 1 1 1 1 1	1 1 1 1 0 reg		4		
or	0 1 0 1 0 reg			1		
memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m		4	1	1
immediate	0 1 1 0 1 0 s 0	immediate data		1		
PUSHA = Push All	0 1 1 0 0 0 0 0			11		
POP = Pop						
reg	1 0 0 0 1 1 1 1	1 1 0 0 0 reg		4	1	
or	0 1 0 1 1 reg			1	2	
memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m		5	2	1
POPA = Pop All	0 1 1 0 0 0 0 1			9	7/15	16/32
XCHG = Exchange						
reg1 with reg2	1 0 0 0 0 1 1 w	1 1 reg1 reg2		3		2
Accumulator with reg	1 0 0 1 0 reg			3		2
Memory with reg	1 0 0 0 0 1 1 w	mod reg r/m		5		2
NOP = No Operation	1 0 0 1 0 0 0 0			1		
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m				
No index register				1		
With index register				2		

Instruction	T T T
ADD=Add	0 0 0
ADC=Add with Carry	0 1 0
AND=Logical AND	1 0 0
OR=Logical OR	0 0 1
SUB=Subtract	1 0 1
SBB=Subtract with Borrow	0 1 1
XOR=Logical Exclusive OR	1 1 0

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
INTEGER OPERATIONS (continued)						
reg1 to reg2	0 0 T T T 0 0 w	1 1 reg1 reg2		1		
reg2 to reg1	0 0 T T T 0 1 w	1 1 reg1 reg2		1		
memory to register	0 0 T T T 0 1 w	mod reg r/m		2	2	
register to memory	0 0 T T T 0 0 w	mod reg r/m		3	6/2	U/L
immediate to register	1 0 0 0 0 0 s w	1 1 T T T reg	immediate data	1		
immediate to accumulator	0 0 T T T 1 0 w	immediate data		1		
immediate to memory	1 0 0 0 0 0 s w	mod T T T r/m	immediate data	3	6/2	U/L

Instruction	TTT
INC = Increment	0 0 0
DEC = Decrement	0 0 1

reg	1 1 1 1 1 1 1 w	1 1 T T T reg		1		
or	0 1 T T T reg			1		
memory	1 1 1 1 1 1 1 w	mod T T T r/m		3	6/2	U/L

Instruction	TTT
NOT = Logical Complement	0 1 0
NEG = Negate	0 1 1

reg	1 1 1 1 0 1 1 w	1 1 T T T reg		1		
memory	1 1 1 1 0 1 1 w	mod T T T r/m		3	6/2	U/L
CMP = Compare						
reg1 with reg2	0 0 1 1 1 0 0 w	1 1 reg1 reg2		1		
reg2 with reg1	0 0 1 1 1 0 1 w	1 1 reg1 reg2		1		
memory with register	0 0 1 1 1 0 0 w	mod reg r/m		2	2	
register with memory	0 0 1 1 1 0 1 w	mod reg r/m		2	2	
immediate with register	1 0 0 0 0 0 s w	1 1 1 1 1 reg	immediate data	1		
immediate with acc.	0 0 1 1 1 1 0 w	immediate data		1		
immediate with memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	immediate data	2	2	
TEST = Logical Compare						
reg1 and reg2	1 0 0 0 0 1 0 w	1 1 reg1 reg2		1		
memory and register	1 0 0 0 0 1 0 w	mod reg r/m		2	2	
immediate and register	1 1 1 1 0 1 1 w	1 1 0 0 0 reg	immediate data	1		
immediate and acc.	1 0 1 0 1 0 0 w	immediate data		1		
immediate and memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	immediate data	2	2	
MUL = Multiply (unsigned)						
acc. with register	1 1 1 1 0 1 1 w	1 1 1 0 0 reg				
Multiplier—Byte				13/18		MN/MX,3
Word				13/26		MN/MX,3
Dword				13/42		MN/MX,3
acc. with memory	1 1 1 1 0 1 1 w	mod 1 0 0 r/m				
Multiplier—Byte				13/18	1	MN/MX,3
Word				13/26	1	MN/MX,3

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
INTEGER OPERATIONS (continued)							
Dword					13/42	1	MN/MX,3
IMUL = Integer Multiply (signed)							
acc. with register	1 1 1 1 0 1 1 w	11 1 0 1 reg					
Multiplier—Byte					13/18		MN/MX,3
Word					13/26		MN/MX,3
Dword					13/42		MN/MX,3
acc. with memory	1 1 1 1 0 1 1 w	mod 1 0 1 r/m					
Multiplier—Byte					13/18		MN/MX,3
Word					13/26		MN/MX,3
Dword					13/42		MN/MX,3
reg1 with reg2	0 0 0 0 1 1 1 1	1 0 1 0 1 1 1 1	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	0 0 0 0 1 1 1 1	1 0 1 0 1 1 1 1	mod reg r/m				
Multiplier—Byte					13/18	1	MN/MX,3
Word					13/26	1	MN/MX,3
Dword					13/42	1	MN/MX,3
reg1 with imm. to reg2	0 1 1 0 1 0 s 1	1 1 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.	0 1 1 0 1 0 s 1	mod reg r/m	immediate data				
Multiplier—Byte					13/18	2	MN/MX,3
Word					13/26	2	MN/MX,3
Dword					13/42	2	MN/MX,3
DIV = Divide (unsigned)							
acc. by register	1 1 1 1 0 1 1 w	11 1 1 0 reg					
Divisor— Byte					16		
Word					24		
Dword					40		
acc. by memory	1 1 1 1 0 1 1 w	mod 1 1 0 r/m					
Divisor— Byte					16		
Word					24		
Dword					40		
IDIV = Integer Divide (signed)							
acc. by register	1 1 1 1 0 1 1 w	11 1 1 1 reg					
Divisor— Byte					19		
Word					27		
Dword					43		
acc. by memory	1 1 1 1 0 1 1 w	mod 1 1 1 r/m					
Divisor— Byte					20		
Word					28		
Dword					44		

Table 10-1 Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
INTEGER OPERATIONS (continued)							
CBW = Convert Byte to Word/							
Convert Word to Dword	`10011000`				3		
CWD = Convert Word to Dword/	`10011001`				3		
Convert Dword to Quad Word							

Instruction	Instruction	T T T
ROL = Rotate Left		0 0 0
ROR = Rotate Right		0 0 1
RCL = Rotate through Carry Left		0 1 0
RCR = Rotate through Carry Right		0 1 1
SHL/SAL = Shift Logical/Arithmetic Left		1 0 0
SHR = Shift Logical Right		1 0 1
SAR = Shift Arithmetic Right		1 1 1

Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
reg by 1	`1101000w`	`11 TTT reg`			3		
memory by 1	`1101000w`	`mod TTT r/m`			4	6	
reg by CL	`1101001w`	`11 TTT reg`			3		
memory by CL	`1101001w`	`mod TTT r/m`			4	6	
reg by immediate count	`1100000w`	`11 TTT reg`	imm. 8-bit data		2		
mem by immediate count	`1100000w`	`mod TTT r/m`	imm. 8-bit data		4	6	

Through Carry (RCL and RCR)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
reg by 1	`1101000w`	`11 TTT reg`			3		
memory by 1	`1101000w`	`mod TTT r/m`			4	6	
reg by CL	`1101001w`	`11 TTT reg`			8/30		MN/MX,4
memory by CL	`1101001w`	`mod TTT r/m`			9/31		MN/MX,5
reg by immediate count	`1100000w`	`11 TTT reg`	immediate 8-bit data		8/30		MN/MX,4
mem by immediate count	`1100000w`	`mod TTT r/m`	immediate 8-bit data		9/31		MN/MX,5

Instruction	T T T
SHLD = Shift Left Double	1 0 0
SHRD = Shift Right Double	1 0 1

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
register with immediate	`00001111`	`10TTT100`	`11 reg2 reg1`	immed. 8-bit data	2		
memory by immediate	`00001111`	`10TTT100`	`11 reg2 reg1`	immed. 8-bit data	3	6	
register by CL	`00001111`	`10TTT101`	`11 reg2 reg1`		3		
memory by CL	`00001111`	`10TTT101`	`mod reg r/m`		4	5	
BSWAP = Byte Swap	`00001111`	`11001 reg`			1		
XADD = Exchange and Add							
reg1, reg2	`00001111`	`1100000w`	`11 reg2 reg1`		3		
memory, reg	`00001111`	`1100000w`	`mod reg r/m`		4	6/2	U/L

Table 10-1 Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
INTEGER OPERATIONS (continued)							
CMPXCHG = Compare and Exchange							
reg1, reg2	0 0 0 0 1 1 1 1	1 0 1 1 0 0 0 w	1 1 reg2 reg1		6		
memory, reg	0 0 0 0 1 1 1 1	1 0 1 1 0 0 0 w	mod reg r/m		7/10	2	6
CONTROL TRANSFER (within segment)							
NOTE: Times are jump taken/not taken							
Jccc = Jump on ccc							
8-bit displacement	0 1 1 1 T T Tn	8-bit disp.			3/1		T/NT,23
full displacement	0 0 0 0 1 1 1 1	1 0 0 0 t t t n	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	0 0 0 0 1 1 1 1	1 0 0 1 T T Tn	1 1 0 0 0 reg		4/3		
memory	0 0 0 0 1 1 1 1	1 0 0 1 T T Tn	mod 0 0 0 r/m		3/4		

Mnemonic cccc	Condition	ttn
O	Overflow	0 0 0 0
NO	No Overflow	0 0 0 1
B/NAE	Below/Not Above or Equal	0 0 1 0
NB/AE	Not Below/Above or Equal	0 0 1 1
E/Z	Equal/Zero	0 1 0 0
NE/NZ	Not Equal/Not Zero	0 1 0 1
BE/NA	Below or Equal/Not Above	0 1 1 0
NBE/A	Not Below or Equal/Above	0 1 1 1
S	Sign	1 0 0 0
NS	Not Sign	1 0 0 1
P/PE	Parity/Parity Even	1 0 1 0
NP/PO	Not Parity/Parity Odd	1 0 1 1
L/NGE	Less Than/Not Greater or Equal	1 1 0 0
NL/GE	Not Less Than/Greater or Equal	1 1 0 1
LE/G	Less Than or Equal/Greater Than	1 1 1 0
NLE/G	Not Less Than or Equal/Greater Than	1 1 1 1

INSTRUCTION	FORMAT		Cache Hit	Penalty if Cache Miss	Notes*
LOOP = LOOP CX Times	1 1 1 0 0 0 1 0	8-bit disp.	7/6		L/NL,23
LOOPZ/LOOPE					
= Loop while Not Zero	1 1 1 0 0 0 0 1	8-bit disp.	9/6		L/NL,23
LOOPNZ/LOOPNE					
= Loop while Not Zero	1 1 1 0 0 0 0 0	8-bit disp.	9/6		L/NL,23
JCXZ = Jump on CX Zero	1 1 1 0 0 0 1 1	8-bit disp.	8/5		T/NT,23
JECXZ = Jump on ECX Zero	1 1 1 0 0 0 1 1	8-bit disp.	8/5		T/NT,23
(Address Size Prefix Differentiates JCXZ for JECXZ)					
JMP = Unconditional Jump (within segment)					
Short	1 1 1 0 1 0 1 1	8-bit disp.	3		7,23

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT		Cache Hit	Penalty if Cache Miss	Notes*
CONTROL TRANSFER (within segment) (continued)					
Direct	`1 1 1 0 1 0 0 1`	full displacement	3		7,23
Register Indirect	`1 1 1 1 1 1 1 1`	`1 1  1 0 0  reg`	5		7,23
Memory Indirect	`1 1 1 1 1 1 1 1`	`mod 1 0 0 r/m`	5	5	7
CALL = Call (within segment)					
Direct	`1 1 1 0 1 0 0 0`	full displacement	3		7,23
Register Indirect	`1 1 1 1 1 1 1 1`	`1 1  0 1 0  reg`	5		7,23
Memory Indirect	`1 1 1 1 1 1 1 1`	`mod 0 1 0 r/m`	5	5	7
RET = Return from CALL					
(within segment)	`1 1 0 0 0 0 1 1`		5	5	
Adding immediate to SP	`1 1 0 0 0 0 1 0`	16-bit disp.	5	5	
ENTER = Enter Procedure	`1 1 0 0 1 0 0 0`	16-bit disp./ 8-bit level			
Level=0			14		
Level=1			17		
Level (L) > 1			17+3L		8
LEAVE = Leave Procedure	`1 1 0 0 1 0 0 1`		5	1	
MULTIPLE-SEGMENT INSTRUCTIONS					
MOV = Move					
reg. to segment reg.	`1 0 0 0 1 1 1 0`	`1 1 sreg3 reg`	3/9	0/3	RV/P,9
memory to segment reg.	`1 0 0 0 1 1 1 0`	`mod sreg3 r/m`	3/9	2/5	RV/P,9
segment reg. to reg.	`1 0 0 0 1 1 0 0`	`1 1 sreg3 reg`	3		
segment reg. to memory	`1 0 0 0 1 1 0 0`	`mod sreg3 r/m`	3		
PUSH = Push					
segment reg.	`0 0 0 sreg2 1 1 0`		3		
(ES, CS, SS, or DS)					
segment reg. (FS or GS)	`0 0 0 0 1 1 1 1`	`1 0 sreg3 0 0 0`	3		
POP = Pop					
segment reg.	`0 0 0 sreg2 1 1 1`		3/9	2/5	RV/P, 9
(ES, SS, or DS)					
segment reg. (FS or GS)	`0 0 0 0 1 1 1 1`	`1 0 sreg3 0 0 1`	3/9	2/5	RV/P, 9
LDS = Load Pointer to DS	`1 1 0 0 0 1 0 1`	`mod reg r/m`	6/12	7/10	RV/P, 9
LES = Load Pointer to ED	`1 1 0 0 0 1 0 0`	`mod reg r/m`	6/12	7/10	RV/P, 9
LFS = Load Pointer to FS	`0 0 0 0 1 1 1 1`	`1 0 1 1 0 1 0 0` `mod reg r/m`	6/12	7/10	RV/P, 9
LGS = Load Pointer to GS	`0 0 0 0 1 1 1 1`	`1 0 1 1 0 1 0 1` `mod reg r/m`	6/12	7/10	RV/P, 9
LSS = Load Pointer to SS	`0 0 0 0 1 1 1 1`	`1 0 1 1 0 0 1 0` `mod reg r/m`	6/12	7/10	RV/P, 9
CALL = Call					
Direct intersegment	`1 0 0 1 1 0 1 0`	unsigned full offset, selector	18	2	R, 7, 22
to same level			20	3	P, 9
thru Gate to same level			35	6	P, 9
to inner level, no parameters			69	17	P, 9
to inner level, x parameter (d) words			77+4X	17+n	P, 11, 9
to TSS			37+TS	3	P, 11, 9
thru Task Gate			38+TS	3	P, 10, 9

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT				Cache Hit	Penalty if Cache Miss	Notes*
MULTIPLE-SEGMENT INSTRUCTIONS (continued)							
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m			17	8	R, 7
to same level					20	1 0	P, 9
thru Gate to same level					35	13	P, 9
to inner level, no parameters					69	24	P, 9
to inner level, x parameter (d) words					77+4X	24+n	P, 1 1, 9
to TSS					37+TS	1 0	P, 1 0, 9
thru Task Gate					38+TS	1 0	P, 1 0, 9
RET = Return from CALL							
intersegment	1 1 0 0 1 0 1 1				13	8	R, 7
to same level					17	9	P, 9
to outer level					35	12	P, 9
intersegment adding	1 1 0 0 1 0 1 0	16-bit disp.					
imm. to SP					14	8	R, 7
to same level					18	9	P, 9
to outer level					36	12	P, 9
JMP = Unconditional Jump							
Direct intersegment	1 1 1 0 1 0 1 0	unsigned full offset, selector			17	2	R, 7, 22
to same level					19	3	P, 9
thru Call Gate to same level					32	6	P, 9
thru TSS					42+TS	3	P, 1 0, 9
thru Task Gate					43+TS	1 0	P, 1 0, 9
Indirect intersegment	1 1 1 1 1 1 1	mod 1 0 1 r/m			13	9	R, 7, 9
to same level					18	1 0	P, 9
thru Call Gate to same level					31	13	P, 9
thru TSS					41+TS	1 0	P, 1 0, 9
thru Task Gate					42+TS	1 0	P, 1 0, 9
BIT MANIPULATION							
BT = TEST BIT							
register, immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	1 1 1 0 0 reg	immed. 8-bit data	3		
memory, immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 0 r/m	immed. 8-bit data	3	1	
reg1, reg2	0 0 0 0 1 1 1 1	1 0 1 0 0 0 1 1	1 1 reg2 reg1		3		
memory, reg	0 0 0 0 1 1 1 1	1 0 1 0 0 0 1 1	mod reg r/m		8	2	

Instruction	TTT
BTS=Test Bit and Set	1 0 1
BTR=Test Bit and Reset	1 1 0
BTC=Test Bit and Compliment	1 1 1

register, immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	1 1 TTT reg	immed. 8-bit data	6		
memory, immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod TTT r/m	immed. 8-bit data	8	2/0	U/L
reg1, reg2	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	1 1 reg2 reg1		6		
memory, reg	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	mod reg r/m		13	3/1	U/L
BSF = Scan Bit Forward							
reg1, reg2	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	1 1 reg2 reg1		6/42		MN/MX, 12
memory, reg	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	mod reg r/m		7/43	2	MN/MX, 13

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
BIT MANIPULATION (continued)						
BSR = Scan Bit Reverse						
reg1, reg2	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 1	1 1 reg2 reg1	6/1 03		MN/MX, 14
memory, reg	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 1	mod reg r/m	7/1 04	1	MN/MX, 15
STRING INSTRUCTIONS						
CMPS = Compare Byte Word	1 0 1 0 0 1 1 w			8	6	16
LODS = Load Byte/Word to AL/EX/EAX	1 0 1 0 1 1 0 w			5	2	
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w			7	2	16
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w			6	2	
STOS = Store Byte/Word from AL/EX/EAX	1 0 1 0 1 0 1 w			5		
XLAT = Translate String	1 1 0 1 0 1 1 1			4	2	
REPEATED STRING INSTRUCTIONS						
Repeated by Count in CX or ECX (C = Count in CX or ECX)						
REPE CMPS = Compare String (Find Non-Match)	1 1 1 1 0 0 1 1	1 0 1 0 0 1 1 w				
C = 0				5		
C > 0				7+7c		16, 17
REPNE CMPS = Compare String (Find Match)	1 1 1 1 0 0 1 0	1 0 1 0 0 1 1 w				
C = 0				5		
C > 0				7+7c		16, 17
REP LODS = Load String	1 1 1 1 0 0 1 1	1 0 1 0 1 1 0 w				
C = 0				5		
C > 0				7+4c		16, 18
REP MOVS = Move String	1 1 1 1 0 0 1 1	1 0 1 0 0 1 0 w				
C = 0				5		
C = 1				13	1	16
C > 1				12+3c		16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	1 1 1 1 0 0 1 1	1 0 1 0 1 1 1 w				
C = 0				5		
C > 0				7+5c		20
REPNE SCAS = Scan String (Find Non-AL/AX/EAX)	1 1 1 1 0 0 1 0	1 0 1 0 1 1 1 w				
C=0				5		
C>0				7+5c		20
REP STOS = Store String	1 1 1 1 0 0 1 1	1 0 1 0 1 0 1 w				
C = 0				5		
C > 0				7+4c		
FLAG CONTROL						
CLC = Clear Carry Flag	1 1 1 1 1 0 0 0			2		
STC = Set Carry Flag	1 1 1 1 1 0 0 1			2		
CMC = Complement Carry Flag	1 1 1 1 0 1 0 1			2		
CLD = Clear Direction Flag	1 1 1 1 1 1 0 0			2		

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
REPEATED STRING INSTRUCTIONS (continued)						
STD = Set Direction Flag	1 1 1 1 1 1 0 1			2		
CLI = Clear Interrupt Enable Flag	1 1 1 1 1 0 1 0			5		
STI = Set Interrupt Enable Flag	1 1 1 1 1 0 1 1			5		
LAHF = Load AH into Flag	1 0 0 1 1 1 1 1			3		
SAHF = Store AH into Flags	1 0 0 1 1 1 1 0			2		
PUSHF = Push Flags	1 0 0 1 1 1 0 0			4/3		RV/P
POPF = Pop Flags	1 0 0 1 1 1 0 1			9/6		RV/P
DECIMAL ARITHMETIC						
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			3		
AAS = ASCII Adjust for Subtract	0 0 1 1 0 1 1 1			3		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0		15		
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0		14		
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1			2		
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			2		
PROCESSOR CONTROL INSTRUCTIONS						
HLT = Halt	1 1 1 1 0 1 0 0			4		
MOV = Move To and From Control/Debug/Test Registers						
CR0 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	11 000 reg	17	2	
CR2/CR3 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	11 eee reg	4		
Reg from CR0–3	0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	11 eee reg	4		
DR0–3 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 1	11 eee reg	10		
DR6–7 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 1	11 eee reg	10		
Register from DR6–7	0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 1	11 eee reg	9		
Register from DR0–3	0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 1	11 eee reg	9		
TR3 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	11 011 reg	4		
TR4–7 from register	0 0 0 0 1 1 1 1	0 0 1 0 0 1 0 0	11 eee reg	4		
Register from TR3	0 0 0 0 1 1 1 1	0 0 1 0 0 1 0 0	11 011 reg	3		
Register from TR4–7	0 0 0 0 1 1 1 1	0 0 1 0 0 1 0 0	11 eee reg	4		
CLTS = Clear Task Switched Flag	0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 0		7	2	
INVD = Invalidate Data Cache	0 0 0 0 1 1 1 1	0 0 0 0 1 0 0 0		4		
WBINVD	0 0 0 0 1 1 1 1	0 0 0 0 1 0 0 1		5		
= Write-Back and Invalidate Data Cache						
INVLPG = Invalidate TLB Entry						
INVLPG memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 111 r/m	12/11		H/NH
PREFIX BYTES						
Address Size Prefix	0 1 1 0 0 1 1 1			1		
LOCK = Bus Lock Prefix	1 1 1 1 0 0 0 0			1		
Operand Size Prefix	0 1 1 0 0 1 1 0			1		
Segment Override Prefix						
CS:	0 0 1 0 1 1 1 0			1		
DS:	0 0 1 1 1 1 1 0			1		

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
PROCESSOR CONTROL INSTRUCTIONS (continued)						
ES:	`00100110`			1		
FS:	`00100110`			1		
GS:	`01100101`			1		
SS:	`00110110`			1		
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	3	
From memory	`00001111`	`00000010`	`mod reg r/m`	11	5	
LGDT = Load Global Descriptor						
Table register	`00001111`	`00000001`	`mod 010 r/m`	12	5	
LIDT = Load Interrupt Descriptor						
Table register	`00001111`	`00000001`	`mod 011 r/m`	12	5	
LLDT = Load Local Descriptor						
Table register from reg.	`00001111`	`00000000`	`11 010 reg`	11	3	
Table register from mem.	`00001111`	`00000000`	`mod 010 r/m`	11	6	
LMSW=Load Machine Status Word						
From register	`00001111`	`00000001`	`11 110 reg`	13		
From memory	`00001111`	`00000001`	`mod 110 r/m`	13	1	
LSL = Load Segment Limit						
From register	`00001111`	`00000011`	`11 reg1 reg2`	10	3	
From memory	`00001111`	`00000011`	`mod reg r/m`	10	6	
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 Local Machine Status						
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		

Table 10-1   Am486DX/DX2 Microprocessor Integer Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit	Penalty if Cache Miss	Notes*
PROTECTION CONTROL (continued)						
VERR = Verify Read Access						
Register	00001111	00000000	11 100 r/m	11	3	
Memory	00001111	00000000	mod 100 r/m	11	7	
VERW = Verify Write Access						
To register	00001111	00000000	11 101 reg	11	3	
To memory	00001111	00000000	mod 101 r/m	11	7	
INTERRUPT INSTRUCTIONS						
INTn = Interrupt Type n	11001101	type		INT+ 4/0		RV/P, 21
INT3 = Interrupt Type 3	11001100			INT+0		21
INTO = Interrupt 4 if	11001110					
Overflow Flag Set						
Taken				INT+2		21
Not Taken				3		21
BOUND = Interrupt 5 if Detect						
Value Out Range	01100010	mod reg r/m				
If in range				7	7	21
If out of range				INT+ 24	7	21
IRET = Interrupt Return	11001111					
Real Mode/Virtual Mode				15	8	
Protected Mode						
To same level				20	11	9
To outer level				36	19	9
To nested task (EFLAGS.NT=1)				TS+32	4	9, 10
External Interrupt				INT+ 11		21
NMI = Non-Maskable Interrupt				INT+3		21
Page Fault				INT+24		21
VM86 Exceptions						
CLI				INT+8		21
STI				INT+8		21
INTn				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

Note:
* Notes for Table 10-1 can be found following Table 10-3.

Table 10-2   Task Switch Clock Counts Table

Method	Value for TS	
	Cache Hit	Miss Penalty
VM/Am486 CPU/286/TSS To Am486 CPU TSS	162	55
VM/Am486 CPU/286/ TSS To 286 VMS	143	31
VM/Am486 CPU/286 TSS To VM TSS	140	37

Table 10-3   Interrupt Clock Counts Table

Method	Value for INT		
	Cache Hit	Miss Penalty	Notes
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task Gate	37 + TS	3	10

NOTES   **(Table 10-1 through Table 10-3)**

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 $\pm m$
4. Clocks $=$ {quotient(count/operand length)} *7+9
    $=$ 8 if count $\leq$ operand length (8/16/32)
5. Clocks $=$ {quotient(count/operand length)} *7+9
    $=$ 9 if count $\leq$ 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 six clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to Table 10-2 for value of TS.
11. Add four extra clocks to the cache miss penalty for each 16 bytes.

NOTES   (continued)

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 six clocks for every 16 bytes compared. Entire penalty on first compare.*
18. *Cache miss penalty: add two clocks for every 16 bytes of data. Entire penalty on first load.*
19. *Cache miss penalty: add four clocks for every 16 bytes moved.*
     (One clock for the first operation and three for the second)
20. *Cache miss penalty: add four clocks for every 16 bytes scanned.*
     (Two clocks each for first and second operations)
21. *Refer to Table 10-3 for value on INT.*
22. *Clock count includes one clock for using bother displacement and immediate.*
23. *Refer to assumption 6 (see Section 10.1.2) in the case of a cache miss.*

Table 10-4   Am486DX/DX2 Microprocessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT			Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 8086 Mode	Notes
I/O INSTRUCTIONS								
IN = Input from:								
Fixed Port	`1 1 1 0 0 1 0 w`	`port number`		14	9	29	27	
Variable Port	`1 1 1 0 1 1 0 w`			14	8	28	27	
OUT = Output to:								
Fixed Port	`1 1 1 0 0 1 1 w`	`port number`		16	11	31	29	
Variable Port	`1 1 1 0 1 1 1 w`			16	10	30	29	
INS = Input Byte/Word from DX Port	`0 1 1 0 1 1 0 w`			17	10	32	30	
OUTS = Output Byte/Word to DX Port	`0 1 1 0 1 1 1 w`			17	10	32	30	1
REP INS = Input String	`1 1 1 1 0 0 1 1`	`0 1 1 0 1 1 0 w`		16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	`1 1 1 1 0 0 1 1`	`0 1 1 0 1 1 1 w`		17+5c	11+5c	31+5c	30+5c	3

Notes:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

Table 10-5 Am486DX/DX2 Microprocessor Floating-Point Clock Count Summary

INSTRUCTION	FORMAT			Cache Hit Avg (Lower Range– Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range– Upper Range)	Notes
DATA TRANSFER							
FLD = Real Load to ST (0):							
32-bit memory	1 1 0 1 1 0 01	mod 000 r/m	s-i-b/disp.	3	2		
64-bit memory	1 1 0 1 1 1 01	mod 000 r/m	s-i-b/disp.	3	3		
80-bit memory	1 1 0 1 1 0 11	mod 101 r/m	s-i-b/disp.	6	4		
ST(i)	1 1 0 1 1 0 11	11000 ST(i)		4			
FILD = Integer Load to ST(0)							
16-bit memory	1 1 0 1 1 1 11	mod 000 r/m	s-i-b/disp.	14.5(13–16)	2	4	
32-bit memory	1 1 0 1 1 0 11	mod 000 r/m	s-i-b/disp.	5(9–12)	2	4(2–4)	
64-bit memory	1 1 0 1 1 1 11	mod 101 r/m	s-i-b/disp.	316.8(10–18)	3	7.8(2–8)	
FBLD = BCD Load to ST(0)	1 1 0 1 1 1 11	mod 100 r/m	s-i-b/disp.	75(70–103)	4	7.7(2–8)	
FST = Store Real from ST(0)							
32-bit memory	1 1 0 1 1 110	mod 010 r/m	s-i-b/disp.	7			1
64-bit memory	1 1 0 1 1 101	mod 010 r/m	s-i-b/disp.	8			2
ST(i)	1 1 0 1 1 101	11010 ST(i)		3			
FSTP = Store Real from ST(0) and Pop							
32-bit memory	1 1 0 1 1 001	mod 011 r/m	s-i-b/disp.	7			1
64-bit memory	1 1 0 1 1 101	mod 011 r/m	s-i-b/disp.	8			2
80-bit memory	1 1 0 1 1 0 11	mod 111 r/m	s-i-b/disp.	6			
ST(i)	1 1 0 1 1 101	11001 ST(i)		3			
FIST = Store Integer from ST(0)							
16-bit memory	1 1 0 1 1 1 11	mod 101 r/m	s-i-b/disp.	33.4(29–34)			
32-bit memory	1 1 0 1 1 0 11	mod 010 r/m	s-i-b/disp.	32.4(28–34)			
FISTP = Store Integer from ST(0) and Pop							
16-bit memory	1 1 0 1 1 1 11	mod 011 r/m	s-i-b/disp.	33.4(29–34)			
32-bit memory	1 1 0 1 1 0 11	mod 011 r/m	s-i-b/disp.	33.4(29–34)			
64-bit memory	1 1 0 1 1 1 11	mod 111 r/m	s-i-b/disp.	33.4(29–34)			
FBSTP =	1 1 0 1 1 1 11	mod 110 r/m	s-i-b/disp.	175(172–176)			
Store BCD from ST(0) and Pop							
FXCH = Exchange ST(0) and ST(i)	1 1 0 1 1 001	11001 ST(i)		4			
COMPARISON INSTRUCTIONS							
FCOM = Compare ST(0) with Real							
32-bit memory	1 1 0 1 1 000	mod 010 r/m	s-i-b/disp.	4	2	1	
64-bit memory	1 1 0 1 1 100	mod 010 r/m	s-i-b/disp.	4	3	1	
ST(i)	1 1 0 1 1 000	1 1 0 1 0 ST(i)		4		1	
FCOM P = Compare ST(0) with Real and Pop							
32-bit memory	1 1 0 1 1 000	mod 011 r/m	s-i-b/disp.	4	2	1	
64-bit memory	1 1 0 1 1 100	mod 011 r/m	s-i-b/disp.	4	3	1	
ST(i)	1 1 0 1 1 000	1 1 0 1 1 ST(i)		4		1	

Table 10-5   Am486DX/DX2 Microprocessor Floating-Point Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit Avg (Lower Range– Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range– Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)							
FCOMPP = Compare ST(0) with ST(i) and Pop Twice	1 1 0 1 1 1 10	1 1 0 1 1 0 0 1		5		1	
FICOM = Compare ST(0) with Integer							
16-bit memory	1 1 0 1 1 1 10	mod 010 r/m	s-i-b/disp.	18(16–20)	2	1	
32-bit memory	1 1 0 1 1 0 10	mod 010 r/m	s-i-b/disp.	16.5(15–17)	2	1	
FICOM P = Compare ST(0) with Integer							
16-bit memory	1 1 0 1 1 1 10	mod 011 r/m	s-i-b/disp.	18(16–20)	2	1	
32-bit memory	1 1 0 1 1 0 10	mod 011 r/m	s-i-b/disp.	16.5(15–17)	2	1	
FTST = Compare ST(0) with 0.0	1 1 0 1 1 001	1 1 1 0 0 1 0 0		4		1	
FUCOM = Unordered compare ST(0) with ST(i)	1 1 0 1 1 001	1 1 1 1 0 ST(i)		4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	1 1 0 1 1 101	1 1 1 0 1 ST(i)		4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	1 1 0 1 1 010	1 1 1 0 1 0 0 1		5		1	
FXAM = Examine ST(0)	1 1 0 1 1 001	1 1 1 0 0 1 0 1		8			
CONSTANTS							
FLDZ = Load + 0.0 into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 1 0		4			
FLD1 = Load + 1.0 into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 0 0 0		4			
FLDP1 = Load π into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 0 1 1		8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 0 0 1		8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 0 1 1		8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 1 0 0		8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	1 1 0 1 1 001	1 1 0 1 1 1 1 0 1		8		2	
ARITHMETIC							
FADD = Add Real with ST(0)							
ST(0)← ST(0) + 32-bit memory	1 1 0 1 1 000	mod 000 r/m	s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0)← ST(0) + 64-bit memory	1 1 0 1 1 100	mod 000 r/m	s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d)← ST(i) – ST(0)	1 1 0 1 1 d00	1 1 0 0 0 ST(i)		10(8–20)		7(5–17)	
FADD = Add real with ST(0) and Pop (ST(i) ← ST(0) – ST(i))	1 1 0 1 1 1 10	1 1 0 0 0 ST(i)		10(8–20)		7(5–17)	
FSUB = Subtract real from ST(0)							
ST(0)← ST(0) – 32-bit memory	1 1 0 1 1 000	mod 100 r/m	s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0)← ST(0) – 64-bit memory	1 1 0 1 1 100	mod 100 r/m	s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d)← ST(0) – ST(i)	1 1 0 1 1 d00	1 1 1 0 1 ST(i)		10(8–20)		7(5–17)	
FSUBP = Subtract real from ST(i) and Pop (ST(i) ← ST(0) – ST(i))	1 1 0 1 1 1 10	1 1 1 0 1 ST(i)	s-i-b/disp.	10(8–20)		7(5–17)	

Table 10-5 Am486DX/DX2 Microprocessor Floating-Point Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit Avg (Lower Range–Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range–Upper Range)	Notes
ARITHMETIC (Continued)							
FSUB R = Subtract real from ST(0)							
ST(0)← 32-bit memory − ST(0)	1 1 0 1 1 000	mod 000 r/m	s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0)← 64-bit memory − ST(0)	1 1 0 1 1 100	mod 000 r/m	s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d)← ST(i) − ST(0)	1 1 0 1 1 d00	1 1 1 0 10 ST(i)		10(8–20)		7(5–17)	
FSUBRP = Subtract real reversed and Pop from (ST(i) ← ST(i) − ST(0))	1 1 0 1 1 1 1 0	1 1 1 0 0 ST(i)	s-i-b/disp.	10(8–20)		7(5–17)	
FMUL = Multiply real with ST(0)							
ST(0)← ST(0) x 32-bit memory	1 1 0 1 1 0 0 0	mod 000 r/m	s-i-b/disp.	11	2	8	
ST(0)← ST(0) x 64-bit memory	1 1 0 1 1 100	mod 000 r/m	s-i-b/disp.	14	3	11	
ST(d)← ST(0) x ST(i)	1 1 0 1 1 d00	1 1 0 0 1 ST(i)		16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(1) ← ST(0) x ST(i))	1 1 0 1 1 1 1 0	1 1 0 0 1 ST(i)	s-i-b/disp.	16		13	
FDIV = Divide ST(0) by Real							
ST(0)← ST(0)/32-bit memory	1 1 0 1 1 000	mod 110 r/m	s-i-b/disp.	73	2	70	3
ST(0)← ST(0)/64-bit memory	1 1 0 1 1 100	mod 110 r/m	s-i-b/disp.	73	3	70	3
ST(d)← ST(0)/ST(i)	1 1 0 1 1 d00	1 1 1 1 d ST(i)		73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0) / ST(i))	1 1 0 1 1 1 1 0	1 1 1 1 1 ST(i)	s-i-b/disp.	73		70	3
FDIVR = Divide real reversed (Real/ST(0))							
ST(0)← 32-bit memory/ST(0)	1 1 0 1 1 000	mod 111 r/m	s-i-b/disp.	73	2	70	3
ST(0)← 64-bit memory/ST(0)	1 1 0 1 1 100	mod 111 r/m	s-i-b/disp.	73	3	70	3
ST(d)← ST(i)/ST(0)	1 1 0 1 1 d00	1 1 1 1 d ST(i)		73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(0) / ST(i))	1 1 0 1 1 1 1 0	1 1 1 1 0 ST(i)	s-i-b/disp.	73		70	3
FIADD = Add Integer to ST(0)							
ST(0)← ST(0) + 16-bit memory	1 1 0 1 1 110	mod 000 r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0)← ST(0) + 32-bit memory	1 1 0 1 1 010	mod 000 r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUB = Subtract Integer from ST(0)							
ST(0)← ST(0) − 16-bit memory	1 1 0 1 1 110	mod 100 r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0)← ST(0) − 32-bit memory	1 1 0 1 1 010	mod 100 r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUBR = Integer Subtract LReversed							
ST(0)← 16-bit memory − ST(0)	1 1 0 1 1 110	mod 101 r/m	s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0)← 32-bit memory − ST(0)	1 1 0 1 1 010	mod 101 r/m	s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FIMUL = Multiply Integer with ST(0)							
ST(0)← ST(0) + 16-bit memory	1 1 0 1 1 110	mod 001 r/m	s-i-b/disp.	25(23–27)	2	8	
ST(0)← ST(0) + 32-bit memory	1 1 0 1 1 010	mod 001 r/m	s-i-b/disp.	23.5(22–24)	2	8	
FIDIV = Integer Divide							
ST(0)← ST(0)/16-bit memory	1 1 0 1 1 110	mod 110 r/m	s-i-b/disp.	87(85–89)	2	70	3
ST(0)← ST(0)/32-bit memory	1 1 0 1 1 010	mod 110 r/m	s-i-b/disp.	85.5(84–86)	2	70	3

Table 10-5 Am486DX/DX2 Microprocessor Floating-Point Clock Count Summary (continued)

INSTRUCTION	FORMAT			Cache Hit Avg (Lower Range– Upper Range)	Penalty If Cache Miss	Concurrent Execution Avg (Lower Range– Upper Range)	Notes
ARITHMETIC (Continued)							
FIDIVR = Integer Divide Reversed							
ST(0)← 16-bit memory/ST(0)	1 1 0 1 1 110	mod 111 r/m	s-i-b/disp.	87(85–89)	2	70	3
ST(0)← 32-bit memory/ST(0)	1 1 0 1 1 010	mod 111 r/m	s-i-b/disp.	85.5(84–86)	2	70	3
FSQRT = Square Root	1 1 0 1 1 001	1 1 1 1 1 0 1 0	s-i-b/disp.	85.5(83–87)		70	
Fscale = Scale ST(0) by ST(i)	1 1 0 1 1 1 1 1	1 1 1 1 1 1 0 1	s-i-b/disp.	31(30–32)		2	
Fxtract = Extract components of ST(0)	1 1 0 1 1 001	1 1 1 1 0 1 0 0	s-i-b/disp.	19(16–20)		4(2–4)	
FPREM = Partial Reminder	1 1 0 1 1 001	1 1 1 1 1 0 0 0	s-i-b/disp.	84(70–138)		2(2–8)	
FPREM1 = Partial Reminders (IEEE)	1 1 0 1 1 001	1 1 1 1 0 1 0 1	s-i-b/disp.	94.5(72–167)		5.5(2–18)	
FRNDINT = Absolute value of ST(0)	1 1 0 1 1 001	1 1 1 1 1 1 0 0	s-i-b/disp.	29.1(21–30)		7.4(2–8)	
FABS = Absoulte value of ST(0)	1 1 0 1 1 001	1 1 1 1 0 0 0 1	s-i-b/disp.	3			
FCHS = Change sign of ST(0)	1 1 0 1 1 1001	1 1 1 1 0 0 0 0	s-i-b/disp.	6			
TRANSCENDENTAL							
FCOS = Cosine of ST(0)	1 1 0 1 1 001	1 1 1 1 1 1 1 1	s-i-b/disp.	241(193–279)		2	6,7
FPTAN = Partial tangent of ST(0)	1 1 0 1 1 001	1 1 1 1 0 0 1 0	s-i-b/disp.	244(200–273)		70	6,7
FPATAN = Partial arctangent	1 1 0 1 1 001	1 1 1 1 0 0 1 1	s-i-b/disp.	289(218–303)		5(2–17)	6
FSIN = Sine of ST(0)	1 1 0 1 1 001	1 1 1 1 1 1 1 0	s-i-b/disp.	241(193–279)		2	6,7
FSINCOS = Sine and cosine of ST(0)	1 1 0 1 1 001	1 1 1 1 1 0 1 1	s-i-b/disp.	291(243–329)		2	6,7
F2XM1 = $2^{ST(0)} - 1$	1 1 0 1 1 001	1 1 1 1 0 0 0 0	s-i-b/disp.	242(140–279)		2	6
FLY2X = ST(1) x $\log_2$(ST(0))	1 1 0 1 1 001	1 1 1 1 0 0 0 1	s-i-b/disp.	311(196–329)		13	6
FLY2XP1 = ST(1) x $\log_2$ (ST(0) + 1.0)	1 1 0 1 1 001	1 1 1 1 1 0 0 1	s-i-b/disp.	313(171–326)		13	6
PROCESSOR CONTROL							
FINIT = Initialize FPU	1 1 0 1 1 011	1 1 1 0 0 0 1 1	s-i-b/disp.	17			4
FSTSW AX = Store status word into AX	1 1 0 1 1 111	1 1 1 0 0 0 0 0	s-i-b/disp.	3			5
FSTSW = Store status word into memory	1 1 0 1 1 101	mod 111 r/m	s-i-b/disp.	3			5
FLDCW = Load control word	1 1 0 1 1 001	mod 101 r/m	s-i-b/disp.	4	2		
FSTCW = Load control word	1 1 0 1 1 111	mod 111 r/m	s-i-b/disp.	3			5
FCLEX = Clear exceptions	1 1 0 1 1 011	1 1 1 0 0 0 1 0	s-i-b/disp.	7			4
FSTENV = Store environment	1 1 0 1 1 001	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
FLDEVN = Load environment	1 1 0 1 1 001	mod 100 r/m	s-i-b/disp.				
Real and Virtual Modes 16-bit Address				44	2		
Real and Virtual Modes 32-bit Address				44	2		
Protected Mode 16-bit Address				34	2		
Protected Mode 32-bit Address				34	2		

Table 10-5   Am486DX/DX2 Microprocessor Floating-Point Clock Count Summary (continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range– Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range– Upper Range)	Notes
PROCESSOR CONTROL (Continued)					
FSAVE = Save state	`1 1 0 1 1  101` `mod 100 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	`1 1 0 1 1 101` `mod 100 r/m` `s-i-b/`				
Real and Virtual Modes 16-bit Address		131	23		
Real and Virtual Modes 32-bit Address		131	27		
Protected Mode 16-bit Address		120	23		
Protected Mode 32-bit Address		120	27		
FINCSTP = Increment Stack Pointer	`1 1 0 1 1 001` `1 1 1 1  0 1 1 1`	3			
FDECSTP = Decrement Stack Pointer	`1 1 0 1 1 001` `1 1 1 1  0 1 1 0`	3			
FFREE = Free ST(i)	`1 1 0 1 1 101` `1 1 0 0 0  ST(i)`	3			
FNOP = No operations	`1 1 0 1 1 001` `1 1 0 1  0 0 0 0`	3			
WAIT = Wait until FPU ready (Minum/Maximum)	`1 0 0 1 1 0 1 1`	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

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 can 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 Figure 10-1 does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10-6 is a complete list of all fields appearing in the Am486DX/DX2 microprocessor instruction set. Detailed tables for each field follow Table 10-6.

10.2.1 32-Bit Extensions of the Instruction Set

With the Am486DX/DX2 microprocessor, the 8086/80186/ 80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having two 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 Am486DX/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, individually allow overriding the Default selection of operand size and effective address size. These prefixes can precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes can be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix toggles 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 the presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. Another example, if the default effective address size is 16 bits, the 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 Am486DX/DX2 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 not important.

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.

10.2.2 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.

Figure 10-1 General Instruction Format
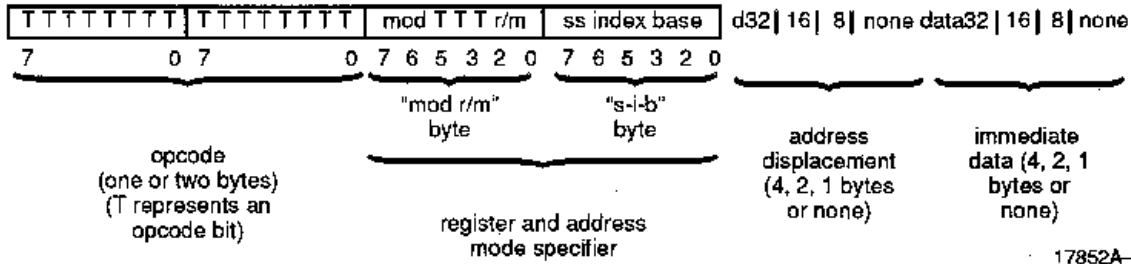
17852A–094

10.2.2.1 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 Table 10-7.

10.2.2.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which can 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. The encoding of the reg field when the w field is not present in the instruction is shown in Table 10-8.

The encoding of the reg field when the w field is present in the instruction during 16-bit data operations is shown in Table 10-9.

The register specified by the reg field when the w field is present in the instruction during 32-bit data operations is shown in Table 10-10.

Table 10-6 Fields within Am486 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
ttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Table 10-7 Encoding of the Operand Length (w) Field

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

Note:

Table 10-1 through Table 10-5 show encoding of individual instructions

Table 10-8   Encoding of the reg Field (w Field not Present 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

Table 10-9   Encoding of the reg Field (w Field is Present, Instruction 16 Bits)

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

Table 10-10   Register Specified by the reg Field (w Field is Present, Instruction 32 Bits)

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

10.2.2.3   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 Am486DX/DX2 microprocessor's FS and GS segment registers to be specified.

Table 10-11   2-Bit sreg2 Field

2–Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

Table 10-12   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

10.2.2.4   Encoding of Address Mode

Except for special instructions such as PUSH or POP, where the addressing mode is predetermined, 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 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 s-i-b 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, can 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.

Table 10-13 through Table 10-15 define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Table 10-13 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]

mod r/m	Effective Address
01 000	DS:[BX + SL + 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]

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

Table 10-14 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]

mod r/m	Effective Address
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]

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

Table 10-15 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)]

mod base	Effective Address
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]

mod base	Effective Address
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 the index field is 100, indicating "no index register," then the ss field MUST equal 00. If the index is 100 and ss does not equal 00, the effective address is undefined.

10.2.2.5 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.

Table 10-16 Encoding of d Field

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

10.2.2.6 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.

Table 10-17 Encoding of s Field

s	Effect on Immediate Data 8	Effect on Immediate Data 16/32
0	None	None
1	Sign-Extend Data 8 to fill 16-Bit or 32-Bit Destination	None

10.2.2.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with "n" indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Table 10-18 Encoding of tttn Field

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/G	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

Table 10-19 Encoding of eee Field

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	

10.2.2.8   Encoding of Control or Debug or Test Register (eee) Field

This field is used to load and store the Control, Debug, and Test registers.

10.2.3   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.

Table 10-20 Encoding of Floating-Point Instruction Fields

	Instruction									Optional Fields	
	First Byte			Second 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		

11 COMPARISON OF Am486DX/DX2 CPU AND THE 386 CPU WITH MATH COPROCESSOR

The differences between the Am486DX/DX2 microprocessor and the 386 microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.

2. The Am486DX/DX2 microprocessor bus is significantly faster than the 386 microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles, and 8-bit bus support. The Hardware Interface and Bus Operation Sections (see Chapters 6 and 7 of this manual) should be carefully read to understand the Am486DX/DX2 microprocessor bus functionality.

3. To support the on-chip cache, new bits have been added to control register 0 (CD and NW) (see Section 2.2.2.1), new pins have been added to the bus (see Chapter 6), and new bus cycle types have been added (see Chapter 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.

4. The complete 387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during floating-point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.

5. The Am486DX/DX2 microprocessor supports new floating-point error reporting modes to guarantee DOS compatibility. These new modes require a new bit in control register 0 (NE) (see Section 2.2.2.1) and new pins ($\overline{FERR}$ and $\overline{IGNNE}$) (see Sections 6.2.13 and 7.2.14).

6. In some cases $\overline{FERR}$ 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 exception (see Sections 6.2.13 and 7.2.14). For both of these cases, the 387 math coprocessor asserts $\overline{ERROR}$ 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) (see Section 4.5.2.5).

9. A new page protection feature has been added. This feature requires a new bit in control register 0 (WP) (see Sections 2.2.2.1 and 4.5.3).

10. A new Alignment Check feature has been added. This feature requires a new bit in the flags register (AC) (see Section 2.2.1.3) and a new bit in control register 0 (AM) (see Section 2.2.2.1).

11. The replacement algorithm for the TLB 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, can be different after reset.

12 CONVERTING AN EXISTING Am486DX CPU DESIGN

Converting an Am486DX CPU system design to an Am486DX2 CPU design provides more performance for a small difference in cost. Migrating from a 33-MHz Am486DX CPU to a 50-MHz Am486DX2 CPU could increase performance by 35%. Conversion can be as easy as replacing one or two devices.

A few system details should be checked first to be sure the design is ready for the Am486DX2 CPU. Check with your BIOS vendor to be sure any BIOS issues have been resolved. The BIOS for the Am486DX CPU may have timing loops. Since the Am486DX2 CPU runs instructions twice as fast as the Am486DX 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 Am486DX CPU and the Am486DX2 CPU. The BIOS may need to be modified to identify the Am486DX2 CPU code properly.

Other system parameters to watch out for are the thermal and power supply specifications. Refer to the Am486 device data sheets, order numbers 17914 and 17852. Since the processor core runs twice as fast for the same input clock, the Am486DX2 CPU uses more power and generates more heat than the Am486DX CPU. Be sure there is adequate cooling and adequate power built into the design. A heat sink is a recommended method to help provide cooling for the Am486DX2 CPU.

The system checks mentioned above are common to all conversions from an Am486DX CPU to an Am486DX2 CPU regardless of the speed of the processor or system.

Migrating from a 33-MHz Am486DX CPU to a 50-MHz Am486DX2 CPU is a two step process. The first step is to change the frequency source for the CPU from 33 MHz to 25 MHz. The Am486DX2 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 that 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.

Because the Am486DX2 CPU core runs twice as fast as its external bus, it is more sensitive to wait states. The Am486DX2 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 Am486DX2 CPU even more than it benefits the Am486DX CPU and helps to hide the effects of a slower memory subsystem. The Am486DX CPU gains an average of 3–9% performance by the addition of a second-level cache, but the Am486DX2 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.

The graph shown in Figure 12-2 shows a set of benchmarks known to have a poor cache hit rate. This is shown for memory tuning purposes and is not to be taken as absolute performance.

With the absence of a second-level cache, the memory subsystem becomes critical to gaining performance when converting from a 33-MHz Am486DX CPU to a 50-MHz Am486DX2 CPU. For slow memory systems without tuning, the 50-MHz Am486DX2 CPU can possibly run slower than the 33-MHz Am486DX CPU (see Figure 12-2). By tuning the memory design, the 50-MHz Am486DX2 CPU can reach equivalent performance to the 33-MHz Am486DX 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.
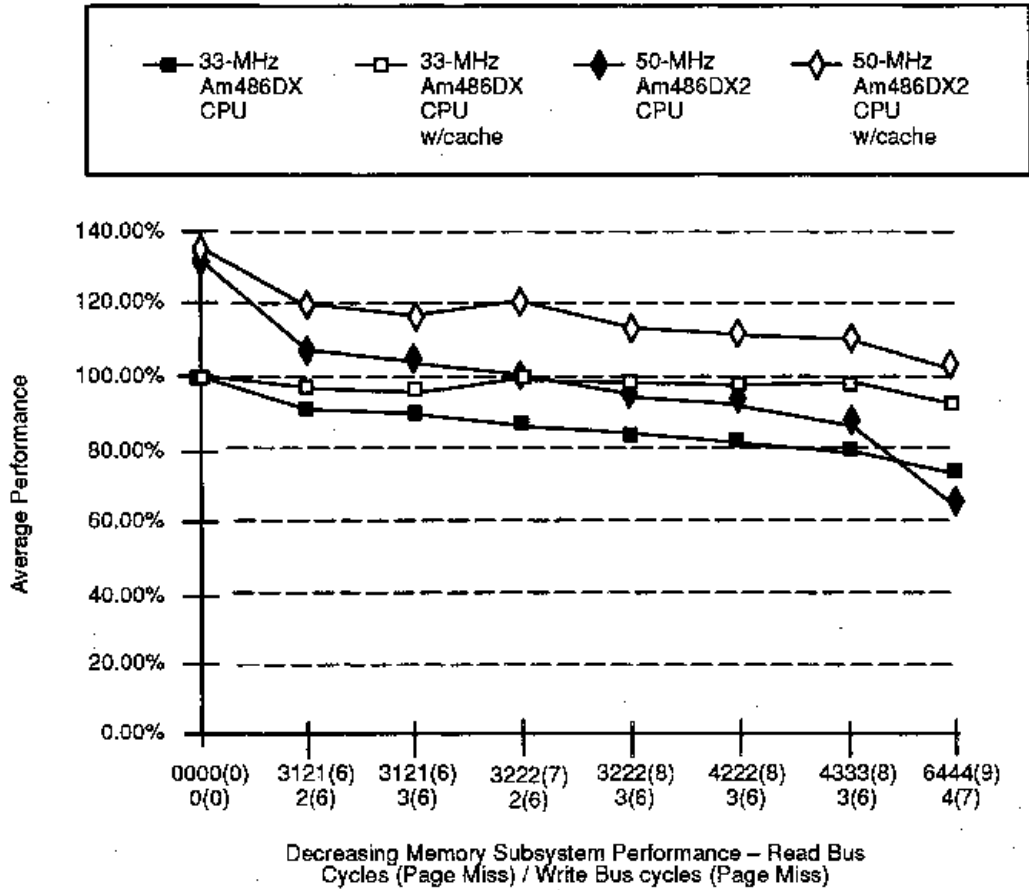
A typical ISA chip set (such as the PCnet™ ISA Am79C960 device with an L2 cache), allows 6-4-4-4 bus cycles at 33-MHz with 80-ns DRAMs for the Am486DX CPU. Without modifying the memory subsystem, the 50-MHz Am486DX2 CPU achieved an average of 7–12% improvement over the 33-MHz Am486DX CPU. By reducing the bus cycles at 25-MHz to 5-2-2-2 (still with 80-ns DRAMS), Am486DX2 CPU improved to achieve an average of 15–20% more performance than the 33-MHz Am486DX 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 Am486DX2 CPU even more.

Figure 12-1 Flowchart for Am486DX CPU to Am486DX2 CPU Conversion

25-MHz Am486DX CPU
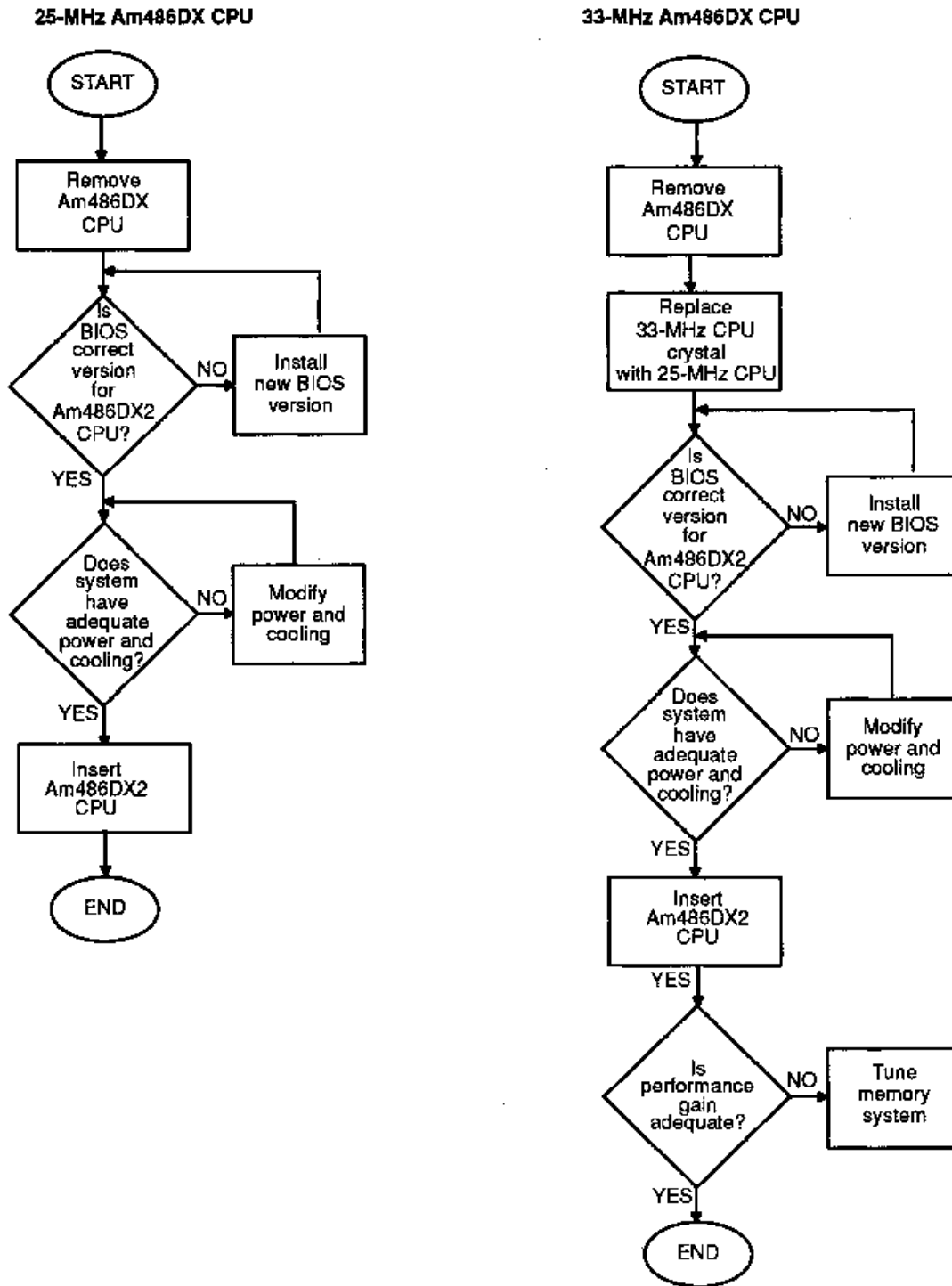
33-MHz Am486DX CPU

17914A–001

Figure 12-2   Performance of 50-MHz Am486DX2 CPU vs. 33-MHz Am486DX CPU

Legend:
- ■ 33-MHz Am486DX CPU
- □ 33-MHz Am486DX CPU w/cache
- ◆ 50-MHz Am486DX2 CPU
- ◇ 50-MHz Am486DX2 CPU w/cache

Average Performance (y-axis): 0.00% to 140.00%

x-axis labels:
0000(0) 0(0) | 3121(6) 2(6) | 3121(6) 3(6) | 3222(7) 2(6) | 3222(8) 3(6) | 4222(8) 3(6) | 4333(8) 3(6) | 6444(9) 4(7)

Decreasing Memory Subsystem Performance – Read Bus
Cycles (Page Miss) / Write Bus cycles (Page Miss)

North American

ALABAMA	(205) 830-9192
ARIZONA	(602) 242-4400
CALIFORNIA,	
Calabasas	(818) 878-9988
Irvine	(714) 450-7500
Sacramento (Roseville)	(916) 786-6700
San Diego	(619) 560-7030
San Jose	(408) 922-0300
CANADA, Ontario,	
Kanata	(613) 592-0060
Woodbridge	(905) 856-3377
COLORADO	(303) 741-2900
CONNECTICUT	(203) 264-7800
FLORIDA,	
Clearwater	(813) 530-9971
Ft. Lauderdale	(954) 938-9550
Orlando (Longwood)	(407) 862-9292
GEORGIA	(770) 449-7920
IDAHO	(208) 377-0393
ILLINOIS, Chicago (Itasca)	(708) 773-4422
KENTUCKY	(606) 224-1353
MARYLAND	(410) 381-3790
MASSACHUSETTS	(617) 273-3970
MINNESOTA	(612) 938-0001
NEW JERSEY,	
Cherry Hill	(609) 662-2900
Parsippany	(201) 299-0002
NEW YORK,	
Brewster	(914) 279-8323
Rochester	(716) 425-8050
NORTH CAROLINA,	
Charlotte	(704) 875-3091
Raleigh	(919) 878-8111
OHIO,	
Columbus (Westerville)	(614) 891-6455
Dayton	(513) 439-0268
OREGON	(503) 245-0080
PENNSYLVANIA	(610) 398-8006
TEXAS,	
Austin	(512) 346-7830
Dallas	(214) 934-9099
Houston	(713) 376-8084

International

AUSTRALIA, N Sydney	TEL	(61) 2 9959-1937
	FAX	(61) 2 9959-1037
BELGIUM, Antwerpen	TEL	(03) 248-4300
	FAX	(03) 248-4642
CHINA,		
Beijing	TEL	(8610) 501-1566
	FAX	(8610) 465-1291
Shanghai	TEL	(8621) 6267-8857
	TEL	(8621) 6267-9683
	FAX	(8621) 6267-8110
FINLAND, Helsinki	TEL	(358) 9 881 3117
	FAX	(358) 9 804 1110
FRANCE, Paris	TEL	(1) 49-75-1010
	FAX	(1) 49-75-1013
GERMANY,		
Bad Homburg	TEL	(06172) 92670
	FAX	(06172) 23195
München	TEL	(089) 450530
	FAX	(089) 406490
HONG KONG, Kowloon	TEL	(852) 2956-0388
	FAX	(852) 2956-0588
ITALY, Milano	TEL	(02) 381961
	FAX	(02) 3810-3458
JAPAN,		
Osaka	TEL	(06) 243-3250
	FAX	(06) 243-3253
Tokyo	TEL	(03) 3346-7600
	FAX	(03) 3346-5197
KOREA, Seoul	TEL	(82) 2784-0030
	FAX	(82) 2784-8014
SINGAPORE, Singapore	TEL	(65) 337-7033
	FAX	(65) 338-1611
SCOTLAND, Stirling	TEL	(44) 7186-450024
	FAX	(44) 1786-446188
SWITZERLAND, Geneva	TEL	(41) 22-788-0251
	FAX	(41) 22-788-0617
SWEDEN,		
Stockholm area	TEL	(08) 629-2850
(Bromma)	FAX	(08) 98-0908
TAIWAN, Taipei	TEL	(886) 2715-3536
	FAX	(886) 2712-2182
UNITED KINGDOM,		
London area	TEL	(01483) 74-0440
(Woking)	FAX	(01483) 75-6196
Manchester area	TEL	(01925) 83-0380
(Warrington)	FAX	(01925) 83-0204

North American Representatives

ARIZONA,	
Scottsdale – THORSON DESERT STATES	(602) 998-2444
CALIFORNIA,	
Chula Vista – SONIKA ELECTRONICA	(619) 498-8340
CANADA,	
Burnaby, B.C. – DAVETEK MARKETING	(604) 430-3680
Dorval, Quebec – POLAR COMPONENTS	(514) 683-3141
Kanata, Ontario – POLAR COMPONENTS	(613) 592-8807
Woodbridge, Ontario – POLAR COMPONENTS	(416) 410-3377
ILLINOIS,	
Skokie – INDUSTRIAL REPS, INC.	(847) 967-8430
INDIANA,	
Kokomo – SCHILLINGER ASSOC.	(317) 457-7241
IOWA,	
Cedar Rapids – LORENZ SALES	(319) 377-4666
KANSAS,	
Merriam – LORENZ SALES	(913) 469-1312
Wichita – LORENZ SALES	(316) 721-0500
MEXICO,	
Guadalajara – SONIKA ELECTRONICA	(523) 647-4250
Mexico City – SONIKA ELECTRONICA	(525) 754-6480
Monterrey – SONIKA ELECTRONICA	(528) 358-9280
MICHIGAN,	
Brighton – COM-TEK SALES, INC	(810) 227-0007
Holland – COM-TEK SALES, INC	(616) 335-8418
MINNESOTA,	
Edina – MEL FOSTER TECH. SALES, INC	(612) 941-9790
MISSOURI,	
St Louis – LORENZ SALES	(314) 997-4558
NEBRASKA,	
Lincoln – LORENZ SALES	(402) 475-4660
NEW YORK,	
Plainview – COMPONENT CONSULTANTS	(516) 273-5050
East Syracuse – NYCOM	(315) 437-8343
Fairport – NYCOM	(716) 425-5120
OHIO,	
Centerville – DOLFUSS ROOT & CO	(513) 433-6776
Powell – DOLFUSS ROOT & CO	(614) 781-0725
Middleburg Hts – DOLFUSS ROOT & CO	(216) 816-1660
PUERTO RICO,	
Caguas – COMP REP ASSOC, INC	(787) 746-6550
UTAH,	
Murray – FRONT RANGE MARKETING	(801) 288-2500
WASHINGTON,	
Kirkland – ELECTRA TECHNICAL SALES	(206) 821-7442
WISCONSIN,	
Pewaukee – Industrial Representatives, Inc.	(414) 574-9393

AMD◢