FINAL

Am386™DX/DXL

High-Performance, Low-Power, 32-Bit Microprocessor

Advanced
Micro
Devices

DISTINCTIVE CHARACTERISTICS

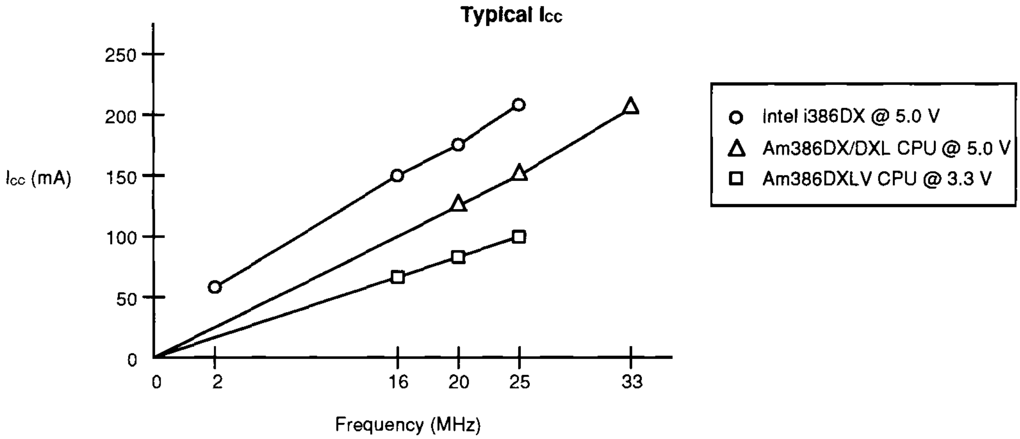
- **Ideal for portable PCs**
  - —True static design for long battery life (Am386DXL microprocessor)
  - —Typical standby $I_{CC} < 20$ µA at DC (0 MHz) (Am386DXL microprocessor)
  - —Typical operating $I_{CC} = 210$ mA at 33 MHz
  - —Lower power consumption than Intel i386DX or Intel i386SX
  - —Small footprint 132-pin PQFP package
  - —Wide range of chip sets and BIOS available to support standby mode capabilities
  - —Performance on demand (0 to 40 MHz)
- **Ideal for desktop PCs**
  - —40-, 33-, 25-, and 20-MHz operating speeds
  - —Lower heat dissipation facilitates fan reduction or elimination for cost savings and noise reduction
  - —Pin-for-pin replacement for Intel i386DX
- **Compatible with 386DX systems and software**
- **Supports 387DX-compatible math coprocessors**
- **AMD advanced 0.8 micron CMOS technology**

GENERAL DESCRIPTION

The Am386DX/DXL microprocessor is a high-speed, true static implementation of the Intel i386DX microprocessor. It is ideal for both desktop and battery-powered portable personal computers. For desktop PCs, the Am386DXL microprocessor offers a 21% increase in the maximum operating speed from 33 to 40 MHz. Also, this device offers lower heat dissipation, allowing system designers to remove or reduce the size and cost of the system cooling fan.

For portables, the Am386DXL microprocessor's true static design offers longer battery life with low operating power consumption and standby mode. At 33 MHz, this device has 40% lower operating I_{CC} than the Intel i386DX. Standby mode allows the Am386DXL microprocessor to be clocked down to 0 MHz (DC) and retain full register contents. In standby mode, typical current draw is less than 20 µA, nearly a 1000× reduction in power consumption versus the Intel i386DX or Intel i386SX.

Additionally, the Am386DXL microprocessor is available in a small footprint 132-pin plastic quad flat pack (PQFP) package. This surface-mount package is 40% smaller than PGA, allowing smaller, lower-cost board designs without the need for a socket.

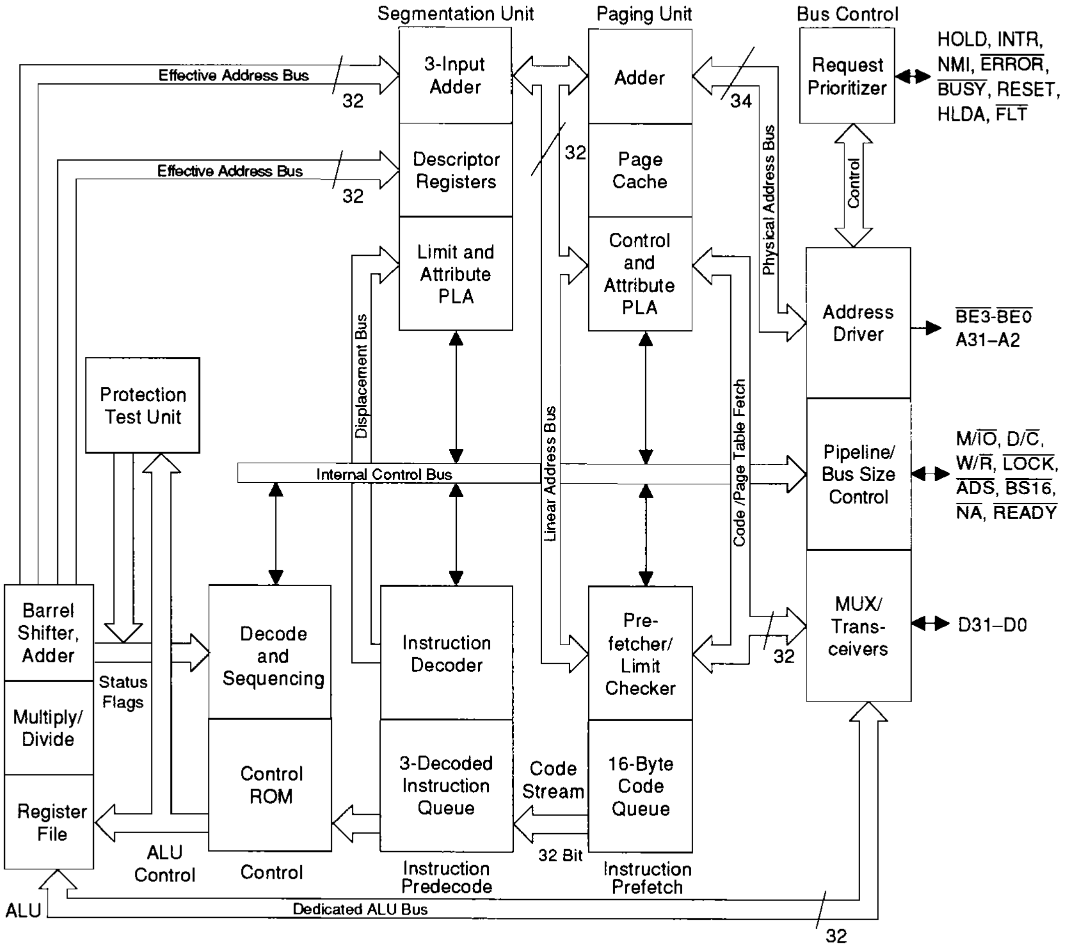


Note: Inputs at V_{CC} or V_{SS}.

Typical Power Consumption

Publication #: **15021** Rev. **D** Amendment: **/0**
Issue Date: **October 1992**

BLOCK DIAGRAM



FUNCTIONAL DESCRIPTION

True Static Operation (Am386DXL CPU)

The Am386DXL microprocessor incorporates a true static design. Unlike dynamic circuit design, the Am386DXL device eliminates the minimum operating frequency restriction. It may be clocked from its maximum speed of 40 MHz all the way down to 0 MHz (DC). System designers can use this feature to design true 32-bit battery-powered portable PCs with long battery life.

Standby Mode (Am386DXL CPU)

This true static design allows for a standby mode. At any of its operating speeds (40 MHz to DC), the Am386DXL microprocessor will retain its state (i.e., the contents of all of its registers). By shutting off the clock completely, the device enters standby mode. Since power consumption is a function of clock frequency, operating power consumption is reduced as the frequency is lowered. In standby mode, typical current draw is reduced to less than 20 µA at DC.

Not only does this feature save battery life, but it also simplifies the design of power-conscious notebook computers in the following ways:

1. Eliminates the need for software in BIOS to save and restore the contents of registers.
2. Allows simpler circuitry to control stopping of the clock since the system does not need to know the processor state.

Lower Operating I_{CC}

True static design also allows lower operating I_{CC} when operating at any speed. See the following graph for typical current at operating speeds.

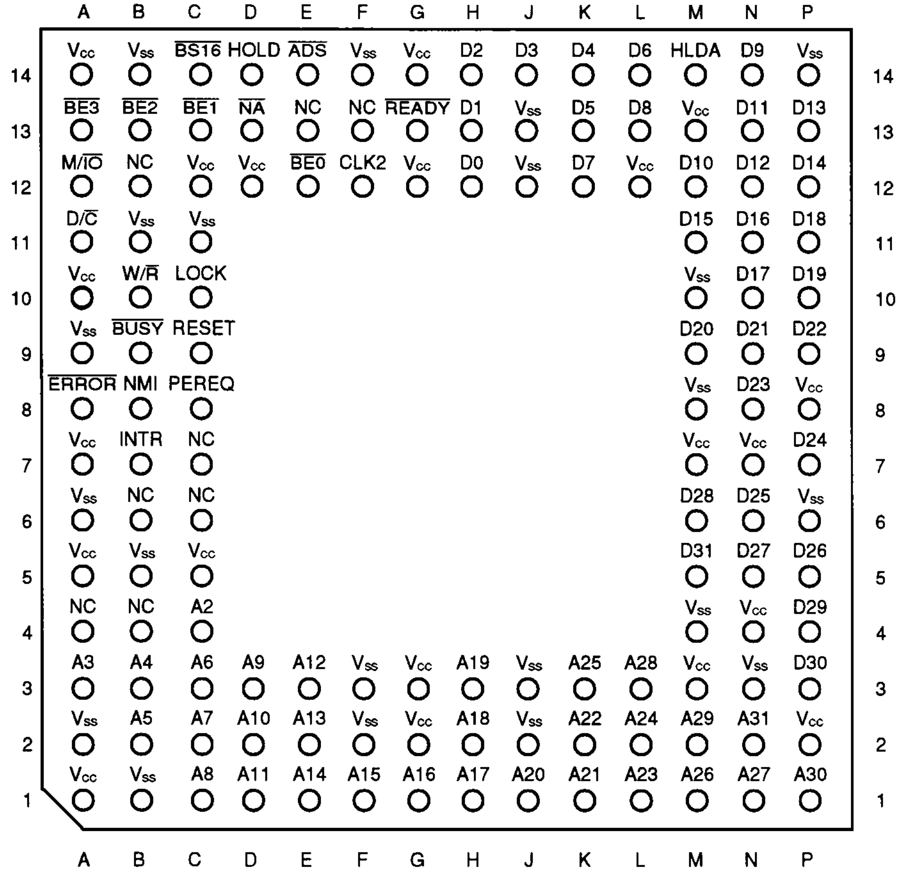
Performance On Demand

The Am386DXL microprocessor retains its state at any speed from 0 MHz (DC) to its maximum operating speed (20, 25, 33, or 40 MHz). With this feature, system designers may vary the operating speed of the system to extend the battery life in portable systems.

For example, the system could operate at low speeds during inactivity or polling operations. However, upon interrupt, the system clock can be increased up to its maximum speed. After a user-defined time-out period, the system can be returned to a low (or 0 MHz) operating speed without losing its state. This design maximizes life while achieving optimal performance.

CONNECTION DIAGRAMS

132-Lead Ceramic Pin Grid Array (PGA) Package — Top Side View

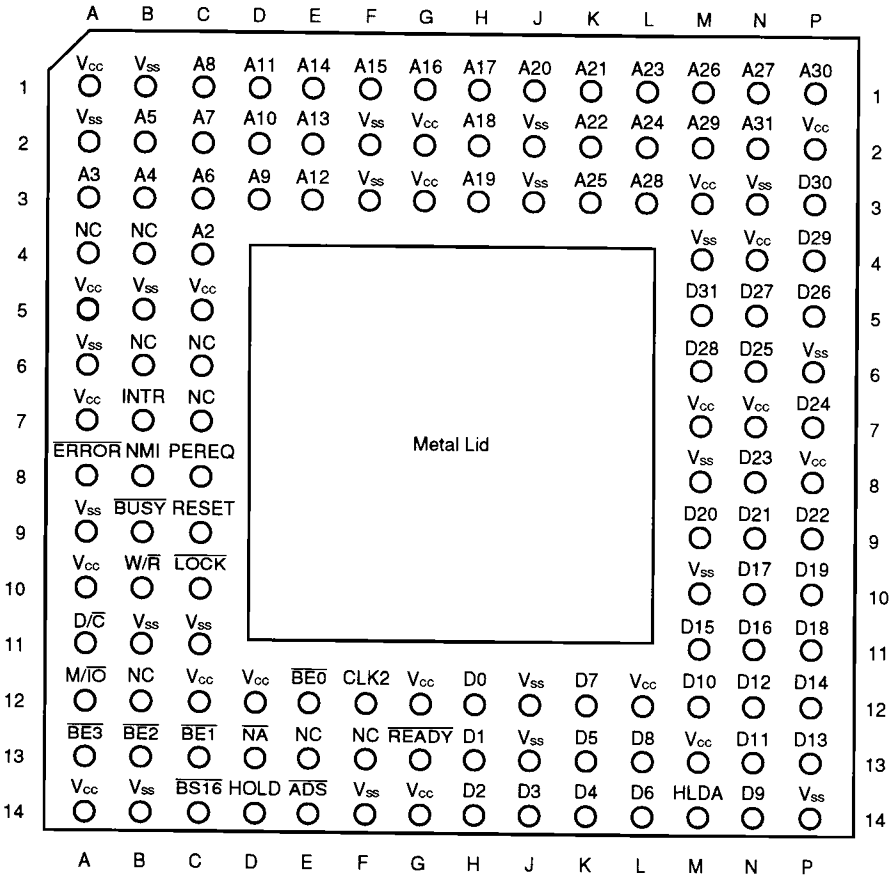


	A	B	C	D	E	F	G	H	J	K	L	M	N	P	
14	V_{CC}	V_{SS}	$\overline{\text{BS16}}$	HOLD	$\overline{\text{ADS}}$	V_{SS}	V_{CC}	D2	D3	D4	D6	HLDA	D9	V_{SS}	14
13	$\overline{\text{BE3}}$	$\overline{\text{BE2}}$	$\overline{\text{BE1}}$	$\overline{\text{NA}}$	NC	NC	$\overline{\text{READY}}$	D1	V_{SS}	D5	D8	V_{CC}	D11	D13	13
12	M/$\overline{\text{IO}}$	NC	V_{CC}	V_{CC}	$\overline{\text{BE0}}$	CLK2	V_{CC}	D0	V_{SS}	D7	V_{CC}	D10	D12	D14	12
11	D/$\overline{\text{C}}$	V_{SS}	V_{SS}									D15	D16	D18	11
10	V_{CC}	W/$\overline{\text{R}}$	LOCK									V_{SS}	D17	D19	10
9	V_{SS}	$\overline{\text{BUSY}}$	RESET									D20	D21	D22	9
8	$\overline{\text{ERROR}}$	NMI	PEREQ									V_{SS}	D23	V_{CC}	8
7	V_{CC}	INTR	NC									V_{CC}	V_{CC}	D24	7
6	V_{SS}	NC	NC									D28	D25	V_{SS}	6
5	V_{CC}	V_{SS}	V_{CC}									D31	D27	D26	5
4	NC	NC	A2									V_{SS}	V_{CC}	D29	4
3	A3	A4	A6	A9	A12	V_{SS}	V_{CC}	A19	V_{SS}	A25	A28	V_{CC}	V_{SS}	D30	3
2	V_{SS}	A5	A7	A10	A13	V_{SS}	V_{CC}	A18	V_{SS}	A22	A24	A29	A31	V_{CC}	2
1	V_{CC}	V_{SS}	A8	A11	A14	A15	A16	A17	A20	A21	A23	A26	A27	A30	1
	A	B	C	D	E	F	G	H	J	K	L	M	N	P	

Note: NC = Not connected; connection of any NC pin may cause a malfunction or incompatability with future shippings of the Am386DX/DXL microprocessor.

CONNECTION DIAGRAMS (continued)

132-Lead Ceramic Pin Grid Array (PGA) Package — Pin Side View



Note: NC = Not connected; connection of any NC pin may cause a malfunction or incompatability with future shippings of the Am386DX/DXL microprocessor.

CONNECTION DIAGRAMS (continued)

PGA Pin Designations (sorted by Functional Grouping)

Address		Data		Control		NC	V_{CC}	V_{SS}
Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin No.	Pin No.	Pin No.
A2	C4	D0	H12	$\overline{\text{ADS}}$	E14	A4	A1	A2
A3	A3	D1	H13	$\overline{\text{BE0}}$	E12	B4	A5	A6
A4	B3	D2	H14	$\overline{\text{BE1}}$	C13	B6	A7	A9
A5	B2	D3	J14	$\overline{\text{BE2}}$	B13	B12	A10	B1
A6	C3	D4	K14	$\overline{\text{BE3}}$	A13	C6	A14	B5
A7	C2	D5	K13	$\overline{\text{BS16}}$	C14	C7	C5	B11
A8	C1	D6	L14	$\overline{\text{BUSY}}$	B9	E13	C12	B14
A9	D3	D7	K12	CLK2	F12	F13	D12	C11
A10	D2	D8	L13	D/$\overline{\text{C}}$	A11		G2	F2
A11	D1	D9	N14	$\overline{\text{ERROR}}$	A8		G3	F3
A12	E3	D10	M12	HLDA	M14		G12	F14
A13	E2	D11	N13	HOLD	D14		G14	J2
A14	E1	D12	N12	INTR	B7		L12	J3
A15	F1	D13	P13	$\overline{\text{LOCK}}$	C10		M3	J12
A16	G1	D14	P12	M/$\overline{\text{IO}}$	A12		M7	J13
A17	H1	D15	M11	$\overline{\text{NA}}$	D13		M13	M4
A18	H2	D16	N11	NMI	B8		N4	M8
A19	H3	D17	N10	PEREQ	C8		N7	M10
A20	J1	D18	P11	$\overline{\text{READY}}$	G13		P2	N3
A21	K1	D19	P10	RESET	C9		P8	P6
A22	K2	D20	M9	W/$\overline{\text{R}}$	B10			P14
A23	L1	D21	N9					
A24	L2	D22	P9					
A25	K3	D23	N8					
A26	M1	D24	P7					
A27	N1	D25	N6					
A28	L3	D26	P5					
A29	M2	D27	N5					
A30	P1	D28	M6					
A31	N2	D29	P4					
		D30	P3					
		D31	M5					

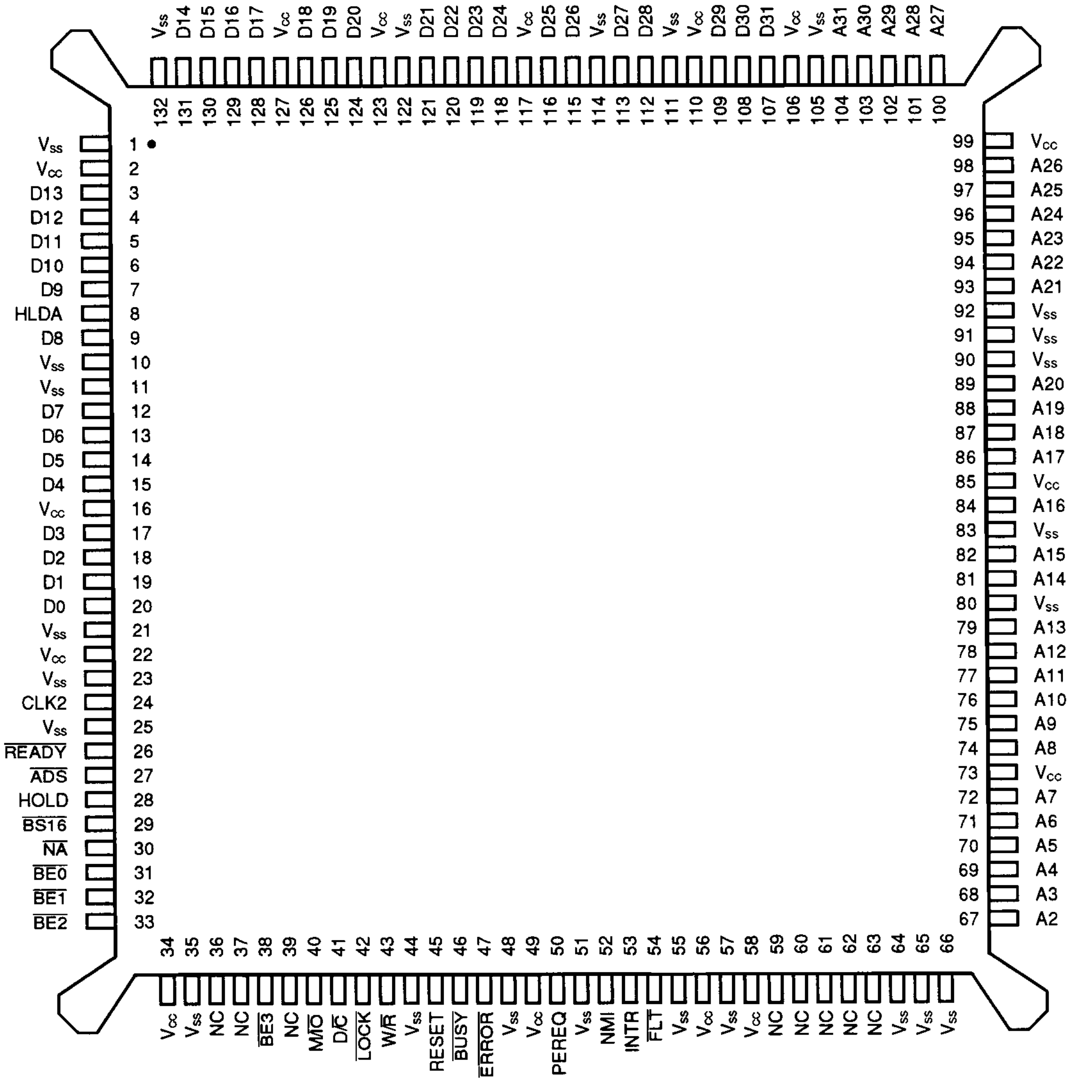
CONNECTION DIAGRAMS (continued)

PGA Pin Designations (sorted by Pin Number)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
A1	V_{cc}	B9	$\overline{\text{BUSY}}$	D3	A9	H1	A17	L13	D8	N7	V_{cc}
A2	V_{ss}	B10	$\text{W}/\overline{\text{R}}$	D12	V_{cc}	H2	A18	L14	D6	N8	D23
A3	A3	B11	V_{ss}	D13	$\overline{\text{NA}}$	H3	A19	M1	A26	N9	D21
A4	NC	B12	NC	D14	HOLD	H12	D0	M2	A29	N10	D17
A5	V_{cc}	B13	$\overline{\text{BE2}}$	E1	A14	H13	D1	M3	V_{cc}	N11	D16
A6	V_{ss}	B14	V_{ss}	E2	A13	H14	D2	M4	V_{ss}	N12	D12
A7	V_{cc}	C1	A8	E3	A12	J1	A20	M5	D31	N13	D11
A8	$\overline{\text{ERROR}}$	C2	A7	E12	$\overline{\text{BE0}}$	J2	V_{ss}	M6	D28	N14	D9
A9	V_{ss}	C3	A6	E13	NC	J3	V_{ss}	M7	V_{cc}	P1	A30
A10	V_{cc}	C4	A2	E14	$\overline{\text{ADS}}$	J12	V_{ss}	M8	V_{ss}	P2	V_{cc}
A11	$\text{D}/\overline{\text{C}}$	C5	V_{cc}	F1	A15	J13	V_{ss}	M9	D20	P3	D30
A12	$\text{M}/\overline{\text{IO}}$	C6	NC	F2	V_{ss}	J14	D3	M10	V_{ss}	P4	D29
A13	$\overline{\text{BE3}}$	C7	NC	F3	V_{ss}	K1	A21	M11	D15	P5	D26
A14	V_{cc}	C8	PEREQ	F12	CLK2	K2	A22	M12	D10	P6	V_{ss}
B1	V_{ss}	C9	RESET	F13	NC	K3	A25	M13	V_{cc}	P7	D24
B2	A5	C10	$\overline{\text{LOCK}}$	F14	V_{ss}	K12	D7	M14	HLDA	P8	V_{cc}
B3	A4	C11	V_{ss}	G1	A16	K13	D5	N1	A27	P9	D22
B4	NC	C12	V_{cc}	G2	V_{cc}	K14	D4	N2	A31	P10	D19
B5	V_{ss}	C13	$\overline{\text{BE1}}$	G3	V_{cc}	L1	A23	N3	V_{ss}	P11	D18
B6	NC	C14	$\overline{\text{BS16}}$	G12	V_{cc}	L2	A24	N4	V_{cc}	P12	D14
B7	INTR	D1	A11	G13	$\overline{\text{READY}}$	L3	A28	N5	D27	P13	D13
B8	NMI	D2	A10	G14	V_{cc}	L12	V_{cc}	N6	D25	P14	V_{ss}

CONNECTION DIAGRAMS (continued)

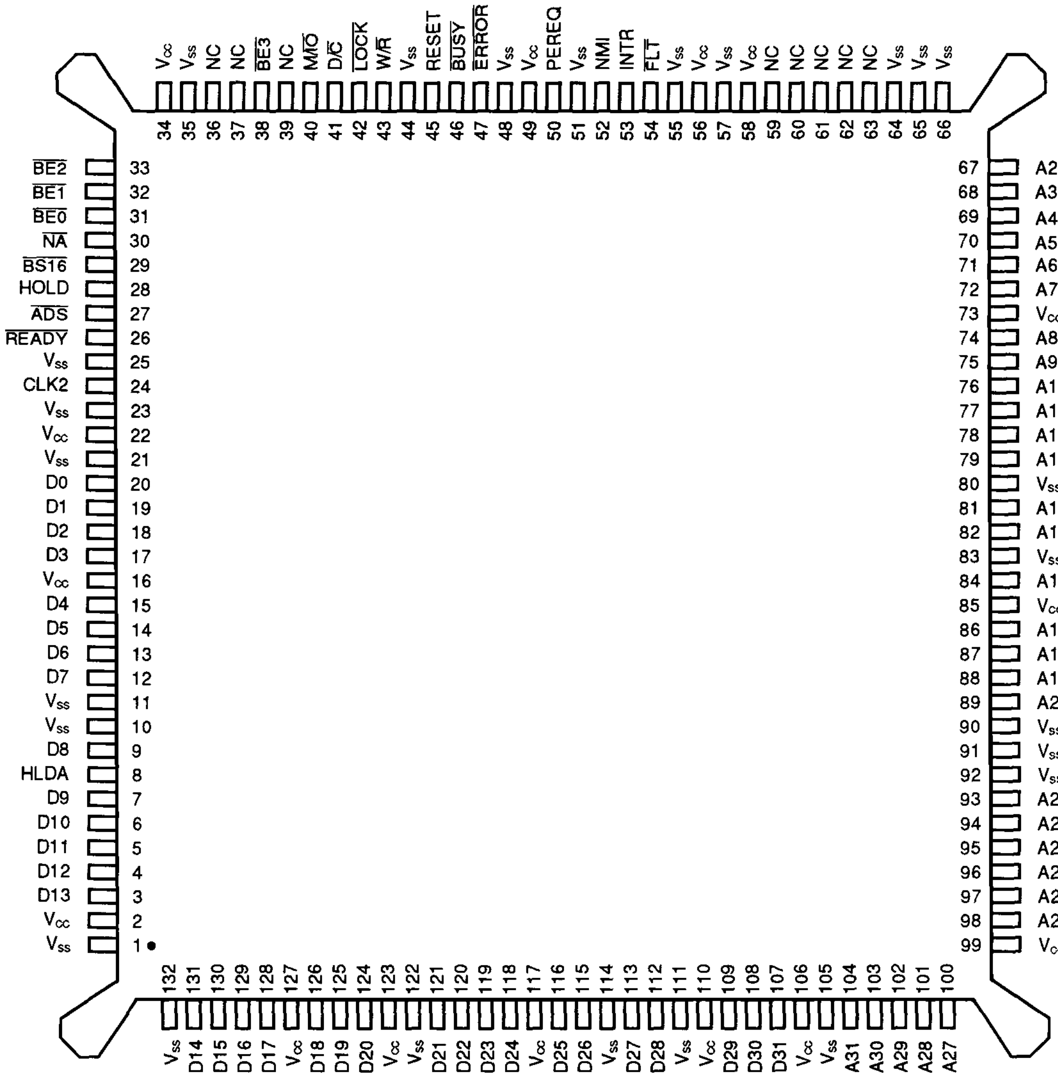
132-Lead Plastic Quad Flat Pack (PQFP) Package—Top Side View



Notes: Pin 1 is marked for orientation.
NC = Not connected; connection of an NC pin may cause a malfunction or incompatibility with future shippings of the Am386DX/DXL microprocessor.

CONNECTION DIAGRAMS (continued)

132-Lead Plastic Quad Flat Pack (PQFP) Package — Pin Side View



Notes: Pin 1 is marked for orientation.
NC = Not connected; connection of an NC pin may cause a malfunction or incompatibility with future shippings of the Am386DX/DXL microprocessor.

CONNECTION DIAGRAMS (continued)

PQFP Pin Designations (sorted by Functional Grouping)

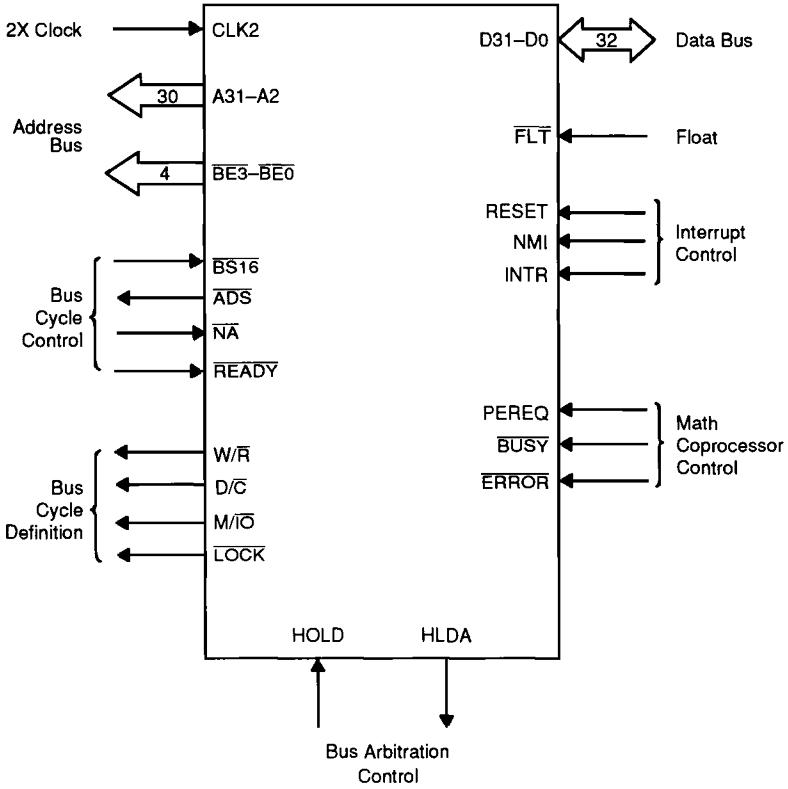
Address		Data		Control		NC	V_{cc}	V_{ss}
Pin Name	**Pin No.**	**Pin Name**	**Pin No.**	**Pin Name**	**Pin No.**	**Pin No.**	**Pin No.**	**Pin No.**
A2	67	D0	20	$\overline{ADS}$	27	36	2	1
A3	68	D1	19	$\overline{BE0}$	31	37	16	10
A4	69	D2	18	$\overline{BE1}$	32	39	22	11
A5	70	D3	17	$\overline{BE2}$	33	59	34	21
A6	71	D4	15	$\overline{BE3}$	38	60	49	23
A7	72	D5	14	$\overline{BS16}$	29	61	56	25
A8	74	D6	13	$\overline{BUSY}$	46	62	58	35
A9	75	D7	12	CLK2	24	63	73	44
A10	76	D8	9	D/$\overline{C}$	41		85	48
A11	77	D9	7	$\overline{ERROR}$	47		99	51
A12	78	D10	6	$\overline{FLT}$	54		106	55
A13	79	D11	5	HLDA	8		110	57
A14	81	D12	4	HOLD	28		117	64
A15	82	D13	3	INTR	53		123	65
A16	84	D14	131	$\overline{LOCK}$	42		127	66
A17	86	D15	130	M/$\overline{IO}$	40			80
A18	87	D16	129	$\overline{NA}$	30			83
A19	88	D17	128	NMI	52			90
A20	89	D18	126	PEREQ	50			91
A21	93	D19	125	$\overline{READY}$	26			92
A22	94	D20	124	RESET	45			105
A23	95	D21	121	W/$\overline{R}$	43			111
A24	96	D22	120					114
A25	97	D23	119					122
A26	98	D24	118					132
A27	100	D25	116					
A28	101	D26	115					
A29	102	D27	113					
A30	103	D28	112					
A31	104	D29	109					
		D30	108					
		D31	107					

CONNECTION DIAGRAMS (continued)

PQFP Pin Designations (sorted by Pin Number)

Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name	Pin No.	Pin Name
1	V_{SS}	23	V_{SS}	45	RESET	67	A2	89	A20	111	V_{SS}
2	V_{CC}	24	CLK2	46	$\overline{BUSY}$	68	A3	90	V_{SS}	112	D28
3	D13	25	V_{SS}	47	$\overline{ERROR}$	69	A4	91	V_{SS}	113	D27
4	D12	26	$\overline{READY}$	48	V_{SS}	70	A5	92	V_{SS}	114	V_{SS}
5	D11	27	$\overline{ADS}$	49	V_{CC}	71	A6	93	A21	115	D26
6	D10	28	HOLD	50	PEREQ	72	A7	94	A22	116	D25
7	D9	29	$\overline{BS16}$	51	V_{SS}	73	V_{CC}	95	A23	117	V_{CC}
8	HLDA	30	$\overline{NA}$	52	NMI	74	A8	96	A24	118	D24
9	D8	31	$\overline{BE0}$	53	INTR	75	A9	97	A25	119	D23
10	V_{SS}	32	$\overline{BE1}$	54	$\overline{FLT}$	76	A10	98	A26	120	D22
11	V_{SS}	33	$\overline{BE2}$	55	V_{SS}	77	A11	99	V_{CC}	121	D21
12	D7	34	V_{CC}	56	V_{CC}	78	A12	100	A27	122	V_{SS}
13	D6	35	V_{SS}	57	V_{SS}	79	A13	101	A28	123	V_{CC}
14	D5	36	NC	58	V_{CC}	80	V_{SS}	102	A29	124	D20
15	D4	37	NC	59	NC	81	A14	102	A30	125	D19
16	V_{CC}	38	$\overline{BE3}$	60	NC	82	A15	104	A31	126	D18
17	D3	39	NC	61	NC	83	V_{SS}	105	V_{SS}	127	V_{CC}
18	D2	40	$M/\overline{IO}$	62	NC	84	A16	106	V_{CC}	128	D17
19	D1	41	$D/\overline{C}$	63	NC	85	V_{CC}	107	D31	129	D16
20	D0	42	$\overline{LOCK}$	64	V_{SS}	86	A17	108	D30	130	D15
21	V_{SS}	43	$W/\overline{R}$	65	V_{SS}	87	A18	109	D29	131	D14
22	V_{CC}	44	V_{SS}	66	V_{SS}	88	A19	110	V_{CC}	132	V_{SS}

LOGIC SYMBOL



15021B–003

ORDERING INFORMATION

Standard Products

AMD standard products are available in several packages and operating ranges. The order number (Valid Combination) is formed by a combination of the elements below.

A **80386DXL** **-33**

OPTIONAL PROCESSING (PQFP Only)
None = Trimmed and Formed PQFP in High Temp Trays
/F = Ringed PQFP in Horizontal Tubes

TEMPERATURE RANGE
Blank = Commercial (T_{CASE} = 0°C to +85°C for PGA)
(T_{CASE} = 0°C to +100°C for PQFP)

SPEED OPTION
–40 = 40 MHz
–33 = 33 MHz
–25 = 25 MHz
–20 = 20 MHz

DEVICE NUMBER/DESCRIPTION
80386DX/DXL
Am386DX/DXL High-Performance, Low-Power, 32-Bit Microprocessor

PACKAGE TYPE
A = 132-Lead Ceramic Pin Grid Array (CGX 132)
NG = 132-Lead Plastic Quad Flat Pack (PQB 132)

Valid Combinations	
A80386DX/DXL	–40 –33 –25 –20
NG80386DX	–40, –40/F
NG80386DX/DXL	–33, –33/F –25, –25/F

Valid Combinations

Valid Combinations lists configurations planned to be supported in volume for this device. All speeds may not be available in all package combinations. Consult the local AMD sales office to confirm availability of specific valid combinations and to check on newly released combinations.

PIN DESCRIPTION

A31–A2

Address Bus (Outputs)

Outputs physical memory or port I/O addresses.

$\overline{\text{ADS}}$

Address Status (Active Low; Output)

Indicates that a valid bus cycle definition and address (W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$, $\overline{\text{BE0}}$, $\overline{\text{BE1}}$, $\overline{\text{BE2}}$, $\overline{\text{BE3}}$, and A31–A2) are being driven at the Am386DX/DXL microprocessor pins.

$\overline{\text{BE3}}$–$\overline{\text{BE0}}$

Byte Enables (Active Low; Outputs)

Indicate which data bytes of the data bus take part in a bus cycle.

$\overline{\text{BS16}}$

Bus Size 16 (Active Low; Input)

Allows direct connection of 32-bit and 16-bit data buses.

$\overline{\text{BUSY}}$

Busy (Active Low; Input)

Signals a busy condition from a processor extension.

CLK2

Clock (Input)

Provides the fundamental timing for the Am386DX/DXL microprocessor.

D31–D0

Data Bus (Inputs/Outputs)

Inputs data during memory, I/O, and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles.

D/$\overline{\text{C}}$

Data/Control (Output)

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

$\overline{\text{ERROR}}$

Error (Active Low; Input)

Signals an error condition from a processor extension.

$\overline{\text{FLT}}$

Float (Active Low; Input)

An input signal which forces all bidirectional and output signals, including HLDA, to the three-state condition.

$\overline{\text{FLT}}$ has an internal pull-up resistor, and if it is not used it should be unconnected.

HLDA

Bus Hold Acknowledge (Active High; Output)

Indicates that the Am386DX/DXL microprocessor has surrendered control of its local bus to another bus master.

HOLD

Bus Hold Request (Active High; Input)

Allows another bus master to request control of the local bus.

INTR

Interrupt Request (Active High; Input)

A maskable input that signals the Am386DX/DXL microprocessor to suspend execution of the current program and execute an interrupt acknowledge function.

$\overline{\text{LOCK}}$

Bus Lock (Active Low; Output)

A bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active.

M/$\overline{\text{IO}}$

Memory I/O (Output)

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

$\overline{\text{NA}}$

Next Address (Active Low; Input)

Used to request address pipelining.

NC

No Connect

Should always remain unconnected. Connection of a NC pin may cause the processor to malfunction or be incompatible with future steppings of the Am386DX/DXL microprocessor.

NMI

Non-Maskable Interrupt Request (Active High; Input)

A non-maskable input that signals the Am386DX/DXL microprocessor to suspend execution of the current program and execute an interrupt acknowledge function.

PEREQ

Processor Extension Request (Active High; Input)

Indicates that the processor extension has data to be transferred by the Am386DX/DXL microprocessor.

$\overline{\text{READY}}$

Bus Ready (Active Low; Input)

Terminates the bus cycle.

RESET

Reset (Active High; Input)

Suspends any operation in progress and places the Am386DX/DXL microprocessor in a known reset state.

V_{CC}

System Power (Input)

Provides the +5-V nominal DC supply input.

V_{SS}

System Ground (Input)

Provides 0-V connection from which all inputs and outputs are measured.

W/$\overline{\text{R}}$

Write/Read (Output)

A bus cycle definition pin that distinguishes write cycles from read cycles.

BASE ARCHITECTURE

Introduction

The Am386DX/DXL microprocessor consists of a central processing unit, a memory management unit, and a bus interface.

The central processing unit consists of the execution unit and instruction unit. The execution unit contains the eight 32-bit general purpose registers that are used for both address calculation, data operations, and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The multiply and divide logic uses a 1-bit per cycle algorithm. The multiply algorithm stops the iteration when the most significant bits of the multiplier are all zero. This allows typical 32-bit multiplies to be executed in under 1 ms. The instruction unit decodes the instruction op-codes and stores them in the decoded instruction queue for immediate use by the execution unit.

The Memory Management Unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability and efficient sharing. The paging mechanism operates beneath and is transparent to the segmentation process to allow management of the physical address space. Each segment is divided into one or more 4-Kb pages. To implement a virtual memory system, the Am386DX/DXL microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to 4 Gb in size. A given region of the linear address space, 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 Am386DX/DXL microprocessor can have a maximum of 16,381 segments of up to 4 Gb each, thus providing 64 tb (trillion bytes) or virtual memory to each task.

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

The Am386DX/DXL microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Virtual Address Mode (Protected Mode). In Real Mode, the Am386DX/DXL device operates as a very fast 8086 but with 32-bit extensions, if desired. Real Mode is required primarily to setup the processor for Protected Mode operation. Protected Mode provides address 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 such task behaves with 8086 semantics, thus allowing 8086 software (an application program or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host Am386DX/DXL microprocessor operating system by the use of paging and the I/O Permission Bitmap.

Finally, to facilitate high-performance system hardware designs, the Am386DX/DXL microprocessor bus interface offers address pipelining, dynamic data bus sizing, and direct Byte Enable signals for each byte of the data bus. These hardware features are described fully beginning in the Functional Data section.

Register Overview

The Am386DX/DXL microprocessor has 32 register resources in the following categories.

- General Purpose Registers
- Segment Registers
- Instruction Pointer and Flags
- Control Registers
- System Address Registers
- Debug Registers
- Test Registers

The registers are a superset of the 8086, 80186, and 80286 registers, so all 16-bit 80186 and 80286 registers are contained within the 32-bit Am386DX/DXL microprocessor.

Figure 1 shows all the Am386DX/DXL microprocessor base architecture registers that include the general address and data registers, the instruction pointer, and the flags register. The contents of these registers are task-specific, so these registers are automatically loaded with a new context upon a task switch operation.

The base architecture also includes six directly accessible segments, each up to 4 Gb in size. The segments are indicated by the selector values placed in Am386DX/DXL CPU segment registers of Figure 1. Various selector values can be loaded as a program executes, if desired.

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

The other types of registers Control, System Address, Debug, and Test are primarily used by system software.

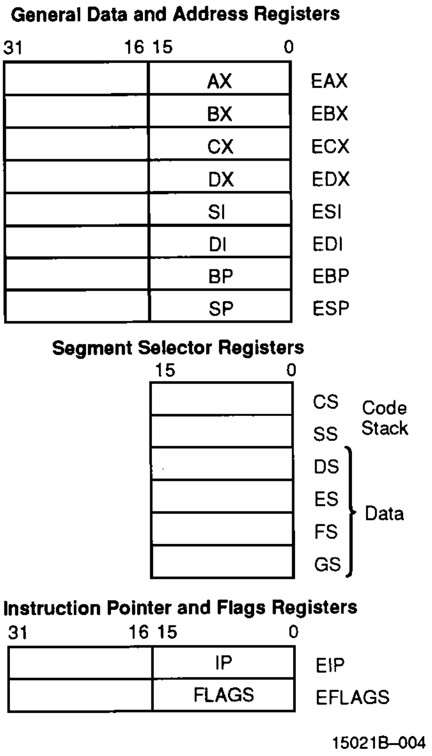


Figure 1. Base Architecture Registers

Register Descriptions

General-Purpose Registers

The eight general-purpose registers of 32 bits hold data or address quantities. The general registers, Figure 2, support data operands of 1, 8, 16, 32, and 64 bits and bit fields of 1 to 32 bits. They support address operands of 16 and 32 bits. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP.

The least significant 16 bits of the registers can be accessed separately. This is done by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP, and SP. When accessed as a 16-bit operand, the upper 16 bits of the register are neither used nor changed.

Finally, 8-bit operations can individually access the lower byte (bits 7–0) and the higher byte (bits 15–8) of general purpose registers AX, BX, CX, and DX. The lower 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.

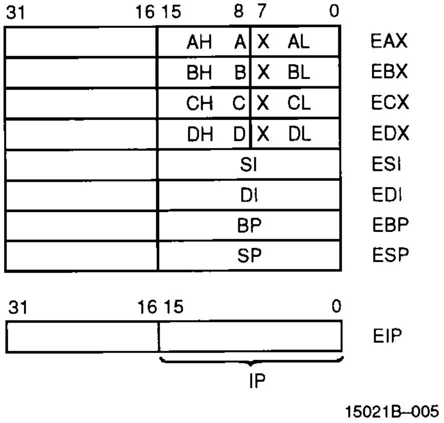


Figure 2. General Registers and Instruction Pointer

Instruction Pointer

The instruction pointer, Figure 2, 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 EIP contain the 16-bit instruction pointer named IP, which is used by 16-bit addressing.

Flags Register

The Flags Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 3, control certain operations and indicate status of the Am386DX/DXL microprocessor. The lower 16 bits (bits 15–0) of EFLAGS contain the 16-bit flag register named FLAGS, which is most useful when executing 8086 and 80286 code.

Note in the following descriptions, *set* means set to 1 and *reset* means reset to 0.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the Am386DX/DXL microprocessor is in Protected Mode, the Am386DX/DXL microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating Exception 13 faults on privileged op-codes. The VM bit can be set only in Protected Mode by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signaled) except the IRET instruction and the POPF instruction. (JMP, CALL, and INT instructions causing a task switch.) These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an 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.

NT (Nested Task, bit 14)

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

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

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an Exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the 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)

OD 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 bits 7, 15, and 31, respectively.

DF (Direction Flag, bit 10)

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

IF (INTR Enable Flag, bit 9)

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

TF (Trap Enable Flag, bit 8)

TF controls the generation of Exception 1 trap when single-stepping through code. When TF is set, the Am386DX/DXL 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 register 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 bits 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)

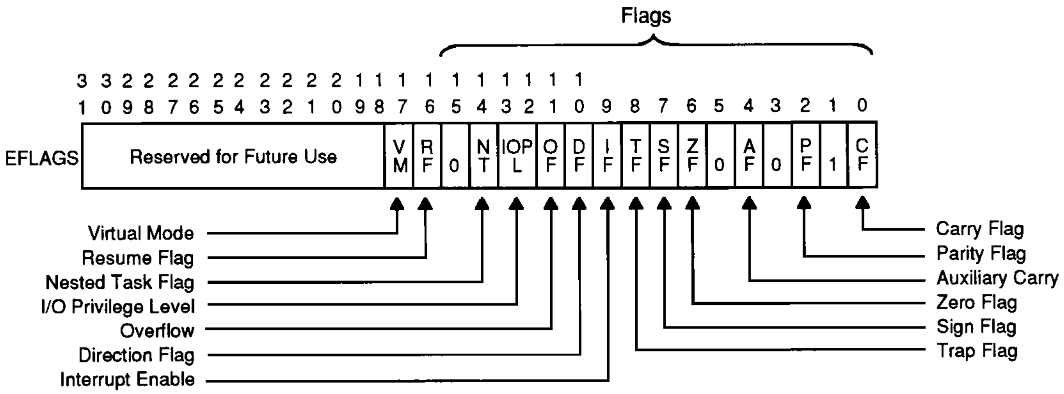
The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise, AF is reset. AF is affected by carry out of, or borrow into, bit 3 only; regardless of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

PF (Parity flags, bit 2)

PF is set if the low-order 8 bits of the operation contain an even number of 1s (even parity). PF is reset if the low-order 8 bits have odd parity. PF is a function of only the low-order 8 bits, regardless of operand size.

CF (Carry Flag, bit 0)

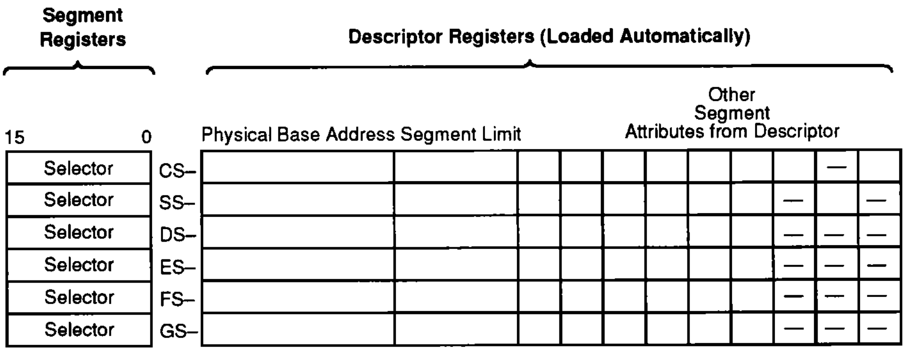
CF is set if the operation resulted in a carry out of (addition) or a borrow into (subtraction) the high-order bit. Otherwise, CF is reset. For 8-, 16-, or 32-bit operations, CF is set according to carry/ borrow at bits 7, 15, or 31, respectively.



Note: 0 indicates "Reserved for Future Use." Do not define; see Section Compatibility.

15021B–006

Figure 3. Flags Registers



15021B–007

Figure 4. Segment Registers and Associated Descriptor Registers

Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. Segment registers are shown in Figure 4. In Protected Mode, each segment may range in size from one byte up to the entire linear and physical space of the machine, 4 Gb (2^{32} bytes). If a maximum sized segment is used (limit = FFFFFFFFH), it should be Dword aligned (i.e., the least two significant bits of the segment base should be zero). This alignment will avoid a segment limit violation (Exception 13) caused by the wrap around. In Real Address Mode, the maximum segment size is fixed at 64 Kb (2^{16} bytes).

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

Segment Descriptor Registers

The segment descriptor registers are not programmer visible, yet it is very useful to understand their content. Inside the Am386DX/DXL microprocessor, a descriptor register (programmer invisible) is associated with each programmer-visible segment register, as shown by Figure 4. Each descriptor register holds a 32-bit segment base address, a 32-bit segment limit, and the other necessary segment attributes.

When a selector value is loaded into a segment register, the associated descriptor 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 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.

Control Registers

The Am386DX/DXL microprocessor has three control registers of 32 bits: CR0, CR2, and CR3 to hold machine state of a global nature (not specific to an individual task). These registers, along with System Address Registers described in the next section, hold machine state that affects all tasks in the system. To access the Control Registers, load and store instructions are defined.

CR0: Machine Control Register (Includes 80286 Machine Status Word)

CR0, shown in Figure 5, contains six defined bits for control and status purposes. The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW) for compatibility with 80286 Protected Mode. LMSW and SMSW instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. For compatibility with 80286 operating systems, the Am386DX/DXL microprocessor LMSW instructions work in an identical fashion to the LMSW instruction on the 80286 (i.e., it only operates on the low-order 16 bits of CR0 and it ignores the new bits in CR0). New Am386DX/DXL 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 set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

R (Reserved, bit 4)

This bit is Reserved for Future Use. When loading CR0 care should be taken to not alter the value of this bit.

TS (Task Switched, bit 3)

TS is automatically set whenever a task switch operation is performed. If TS is set, a coprocessor ESCape op-code will cause a Coprocessor Not Available trap (Exception 7). The trap handler typically saves a 387DX math coprocessor context belonging to a previous task, loads a 387DX math coprocessor state belonging to the current task, and clears the TS bit before returning to the faulting coprocessor op-code.

EM (Emulate Coprocessor, bit 2)

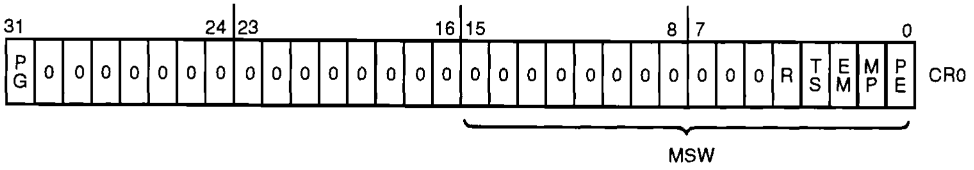
The Emulate coprocessor bit is set to cause all coprocessor op-codes to generate a Coprocessor Not Available fault (Exception 7). It is reset to allow coprocessor op-codes to be executed on an actual 387DX math coprocessor (this is the default case after reset). Note that the WAIT op-code is not affected by the EM bit setting.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if the WAIT op-code will generate a Coprocessor Not Available fault (Exception 7) when TS = 1. When both MP = 1 and TS = 1, the WAIT op-code generates a trap. Otherwise, the WAIT op-code does not generate a trap. Note that TS is automatically set whenever a task switch operation is performed.

PE (Protection Enable, bit 0)

The PE bit is set to enable the Protected Mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by a load into CR0. Resetting the PE bit is typically part of a longer instruction sequence needed for proper transition from Protected Mode to Real Mode. Note that for strict 80286 compatibility, PE cannot be reset by the LMSW instruction.



Note: 0 indicates "Reserved for Future Use." Do not define; see Section Compatibility.

Figure 5. Control Register 0

15021B–008

CR1: Reserved

CR1 is reserved for future processors.

CR2: Page Fault Linear Address

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

CR3: Page Directory Base Address

CR3, shown in Figure 6, contains the physical base address of the page directory table. The Am386DX/DXL microprocessor page directory table is always page-aligned (4-Kb aligned). Therefore, the lowest 12 bits of CR3 are ignored when written and they store as undefined.

A task switch through a TSS that changes the value in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the paging unit cache. Note that if the value in CR3 does not change during the task switch, the cached page table entries are not flushed.

System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286 CPU and Am386DX/DXL microprocessor protection model.

These tables or segments are:

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

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers illustrated in Figure 7. These registers are named GDTR, IDTR, LDTR, and TR, respectively. The Protected Mode Architecture section describes the use of these registers.

GDTR and IDTR

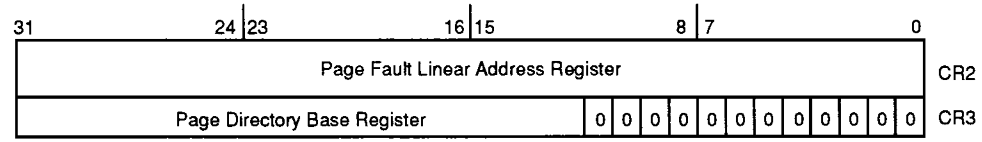
These registers hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

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

LDTR and TR

These registers hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

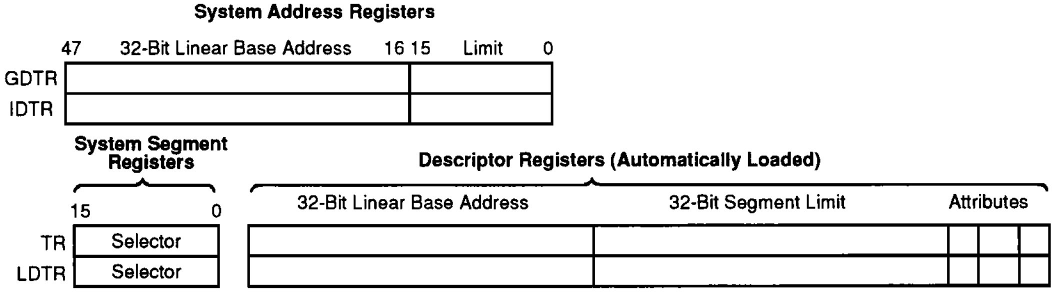
The LDT and TSS segments, since they are task-specific segments, are defined by selector values stored in the system segment registers. Note that a segment descriptor register (programmer-invisible) is associated with each system segment register.



Note: 0 indicates "Reserved for Future Use." Do not define; see Section Compatibility.

15021B–009

Figure 6. Control Registers 2 and 3



15021B–010

Figure 7. System Address and System Segment Registers

Debug and Test Registers

Debug Registers: The six programmer accessible debug registers 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, and the Debug Status Register DR6 displays the current state of the breakpoints. The use of the debug registers is described in the Debugging Support section.

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Look-Aside Buffer portion of the Am386DX/DXL microprocessor. TR6 is the command test register, and TR7 is the data register that contains the data of the Translation Look-Aside buffer test. Their use is discussed in the Testability section. Figure 8 shows the Debug and Test registers.

Debug Registers

31 — 0	
Linear Breakpoint Address 0	DR0
Linear Breakpoint Address 1	DR1
Linear Breakpoint Address 2	DR2
Linear Breakpoint Address 3	DR3
Reserved for Future use. Do not define.	DR4
Reserved for Future use. Do not define.	DR5
Breakpoint Status	DR6
Breakpoint Control	DR7

Test Registers (For Page Cache)

31 — 0	
Test Control	TR6
Test Status	TR7

15021B–011

Figure 8. Debug and Test Registers

Register Accessibility

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 1 summarizes these differences. See the Protected Mode Architecture section for further details.

Compatibility

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain Am386DX/DXL microprocessor register bits are Reserved for Future Use. When reserved bits are called out, treat them as fully undefined. This is essential for software compatibility with future processors! Follow the guidelines below:

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

Depending upon the values of undefined register bits will make your software dependent upon the unspecified Am386DX/DXL microprocessor handling of these bits. Depending on undefined values risks making software incompatible with future processors that define usages for the Am386DX/DXL CPU undefined bits. Avoid any software dependence upon the state of undefined Am386DX/DXL CPU register bits.

Table 1. Register Usage

	Use In Real Mode		Use In Protected Mode		Use In Virtual 8086 Mode	
Register	**Load**	**Store**	**Load**	**Store**	**Load**	**Store**
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Registers	Yes	Yes	Yes	Yes	Yes	Yes
Flag Registers	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
Debug Control	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.

Instruction Set

Instruction Set Overview

The instruction set is divided into nine categories of operations.

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

These Am386DX/DXL microprocessor instructions are listed in Table 2.

All Am386DX/DXL microprocessor instructions operate on either 0, 1, 2, or 3 operands where an operand resides in a register in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2-bytes long. Since the Am386DX/DXL device has a 16-byte instruction queue, an average of 5 instructions will be prefetched. The use of two operands permits the following types of common instructions.

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

The operands can be either 8-, 16-, or 32-bits long. As a general rule, when executing code written for the Am386DX/DXL microprocessor (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 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).

Addressing Modes

Addressing Modes Overview

The Am386DX/DXL 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.

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 op-code.

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 of 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 that requires one additional clock.

As shown in Figure 9, the effective address (EA) of an operand is calculated according to the following formula.

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

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16-, or 32-bit displacement.

Example: INC Word PTR [500]

Register Indirect Mode: A Base register contains the address of the operand.

Example: MOV [ECX], EDX

Based Mode: A Base register's contents is added to a Displacement to form the operands offset.

Example: MOV ECX, [EAX + 24]

Table 2. Am386DX/DXL Microprocessor Instructions

Table 2a. Data Transfer

General Purpose	
MOV	Move operand
PUSH	Push operand onto stack
POP	Pop operand off stack
PUSHA	Push all registers on stack
POPA	Pop all registers off stack
XCHG	Exchange operand register
XLAT	Translate
Conversion	
MOVZX	Move byte or Word, Dword with zero extension
MOVSX	Move byte or Word, Dword, sign extended
CBW	Convert byte to Word, or Word to Dword
CWD	Convert Word to Dword
CWDE	Convert Word to Dword extended
CDQ	Convert Dword to Qword
Input/Output	
IN	Input operand from I/O space
OUT	Output operand to I/O space
Address Object	
LEA	Load effective address
LDS	Load pointer into D segment register
LES	Load pointer into E segment register
LFS	Load pointer into F segment register
LGS	Load pointer into G segment register
LSS	Load pointer into S (Stack) segment register
Flag Manipulation	
LAHF	Load A register from Flags
SAHF	Store A register in Flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
PUSHFD	Push EFLAGS onto stack
POPFD	Pop EFLAGS off stack
CLC	Clear Carry Flag
CLD	Clear Direction Flag
CMC	Complement Carry Flag
STC	Set Carry Flag
STD	Set Direction Flag

Table 2b. Arithmetic Instructions

Addition	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
Subtraction	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII adjust for subtraction
Multiplication	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
Division	
DIV	Divide unsigned
IDIV	Integer divide
AAD	ASCII adjust before division

Table 2c. String Instructions

MOVS	Move byte or Word, Dword string
INS	Input string from I/O space
OUTS	Output string to I/O space
CMPS	Compare byte or Word, Dword string
SCAS	Scan Byte or Word, Dword string
LODS	Load byte or Word, Dword string
STOS	Store byte or Word, Dword string
REP	Repeat
REPE/ REPZ	Repeat while equal/zero
RENE/ REPNZ	Repeat while not equal/not zero

Table 2. Am386DX/DXL Microprocessor Instructions (continued)

Table 2d. Logical Instructions

Logicals	
NOT	"NOT" operand
AND	"AND" operands
OR	"Inclusive OR" operands
XOR	"Exclusive OR" operands
TEST	"Test" operands
Shifts	
SHL/SHR	Shift logical left or right
SAL/SAR	Shift arithmetic left or right
SHLD/ SHRD	Double shift left or right
Rotates	
ROL/ROR	Rotate left/right
RCL/RCR	Rotate through carry left/right

Table 2e. Bit Manipulation Instructions

Single Bit Instructions	
BT	Bit Test
BTS	Bit Test and Set
BTR	Bit Test and Reset
BTC	Bit Test and Complement
BSF	Bit Scan Forward
BSR	Bit Scan Reverse

Table 2f. Program Control Instructions

Conditional Transfers	
SETCC	Set byte equal to condition code
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign

Table 2f. Program Control Instructions (continued)

Unconditional Transfers	
CALL	Call procedure/task
RET	Return from procedure
JMP	Jump
Iteration Controls	
LOOP	Loop
LOOPE/ LOOPZ	Loop if equal/zero
LOOPNE/ LOOPNZ	Loop if not equal/not zero
JCXZ	JUMP if register CX = 0
Interrupts	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Return from interrupt/task
CLI	Clear interrupt enable
STI	Set interrupt enable

Table 2g. High Level Language Instructions

BOUND	Check array bounds
ENTER	Setup parameter block for entering procedure
LEAVE	Leave procedure

Table 2h. Protection Model

SGDT	Store global descriptor table
SIDT	Store interrupt descriptor table
STR	Store task register
SLDT	Store local descriptor table
LGDT	Load global descriptor table
LIDT	Load interrupt descriptor table
LTR	Load task register
LLDT	Load local descriptor table
ARPL	Adjust requested privilege level
LAR	Load access rights
LSL	Load segment limit
VERR/ VERW	Verify segment for reading or writing
LMSW	Load machine status word (lower 16 bits of CR0)
SMSW	Store machine status word

Table 2i. Processor Control Instructions

HLT	Halt
WAIT	Wait until $\overline{\text{BUSY}}$ negated
ESC	Escape
LOCK	Lock Bus

Index Mode: An Index register's contents is added to a Displacement to form the operands offset.

Example: ADD EAX, TABLE [ESI]

Scaled Index Mode: An Index register's contents is multiplied by a scaling factor that is added to a Displacement to form the operands offset.

Example: IMUL EBX, TABLE [ESI•4], 7

Based Index Mode: The contents of a Base register is added to the contents of an Index register to form the effective address of an operand.

Example: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an Index register is multiplied by a Scaling factor and the result is added to the contents of a Base register to obtain the operands offset.

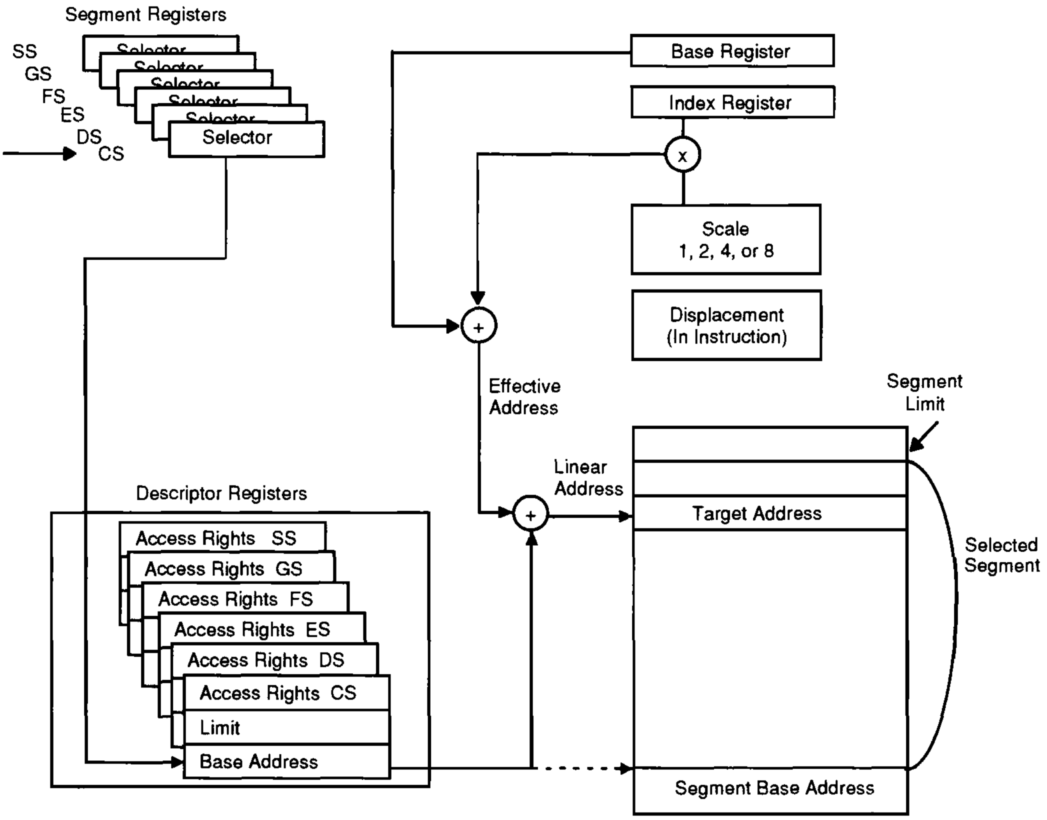
Example: MOV ECX, [EDX•8] [EAX]

Based Index Mode with Displacement: The contents of an Index Register and a Base register's contents and a Displacement are all summed together to form the operand offset.

Example: ADD EDX, [ESI] [EBP+00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an Index register are multiplied by a Scaling factor; the result is added to the contents of a Base register and a Displacement to form the operand's offset.

Example: MOV EAX, LOCALTABLE[EDI•4] [EBP+80]



15021B–012

Figure 9. Addressing Mode Calculations

Differences Between 16- and 32-Bit Addresses

In order to provide software compatibility with the 80286 and the 8086, the Am386DX/DXL microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode, the default size for operands and addresses is 16 bits.

Regardless of the default precision of the operands or addresses, the Am386DX/DXL 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.

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. An assembler automatically determines that an Operand Size Prefix is needed and generates it.

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

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

The Operand Length and Address Length prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kb to be accessed in Real Mode. A memory address exceeding FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional Am386DX/DXL microprocessor addressing modes.

When executing 32-bit code, the Am386DX/DXL microprocessor uses either 8- or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8 or 16 bits, and the base and index register conform to the 80286 model. Table 3 illustrates the differences.

Data Types

The Am386DX/DXL microprocessor supports all data types commonly used in high-level languages.

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits that spans a maximum of four bytes.

Bit String: A set of contiguous bits on the Am386DX/DXL microprocessor bit strings can be up to 4 Gb long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Offset: A 16- or 32-bit offset only quantity that indirectly references another memory location.

Pointer: A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.

Char: A byte representation of an ASCII alphanumeric or control character.

String: A contiguous sequence of bytes, words, or Dwords. A string may contain between 1 byte and 4 Gb.

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

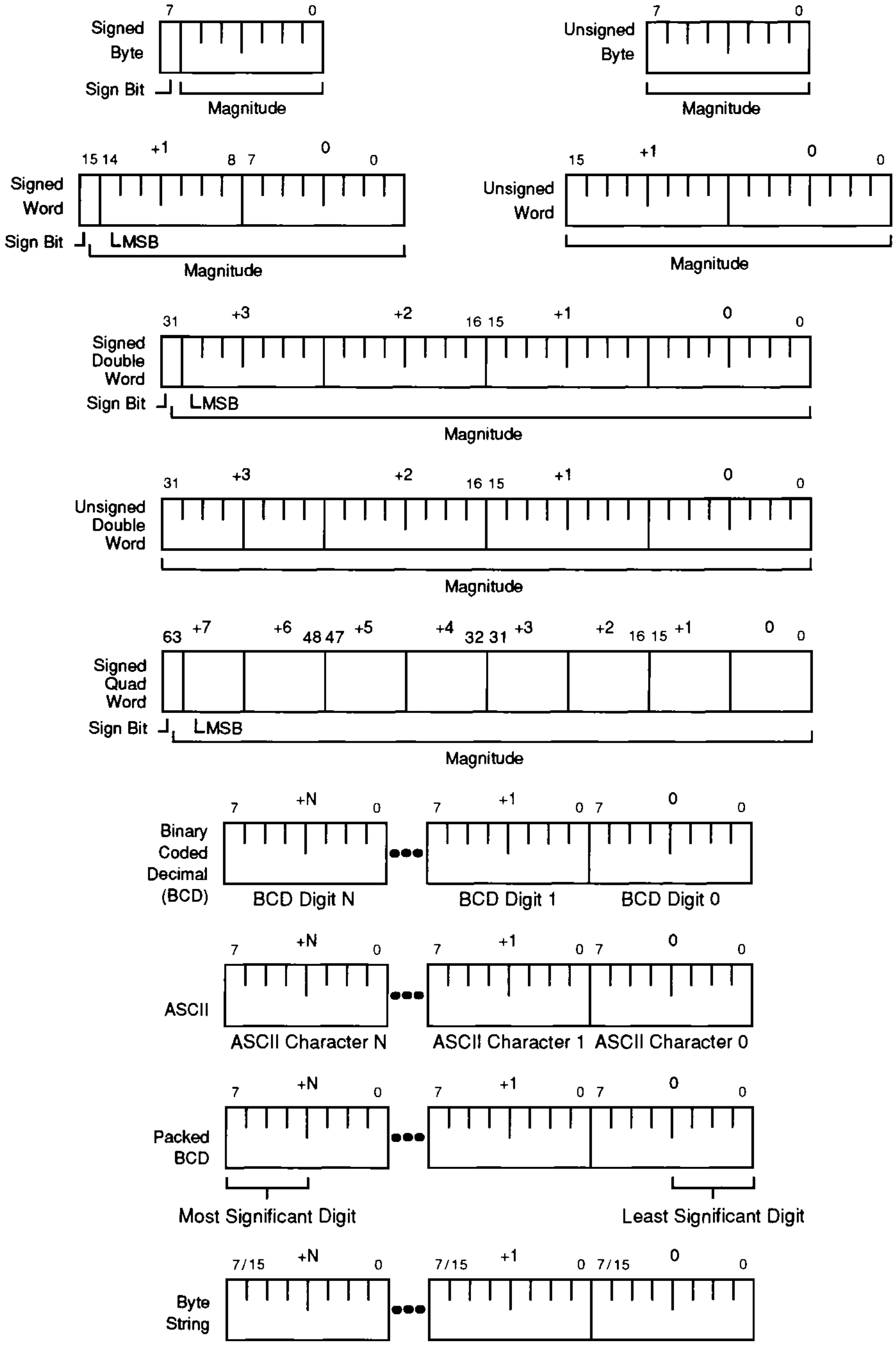
When the Am386DX/DXL microprocessor is coupled with a 387DX math coprocessor then the following common floating point types are supported.

Floating Point: A signed 32-, 64-, or 80-bit real number representation. Floating point numbers are supported by a 387DX compatible math coprocessor.

Figure 10 illustrates the data types supported by the Am386DX/DXL microprocessor and a 387DX compatible math coprocessor.

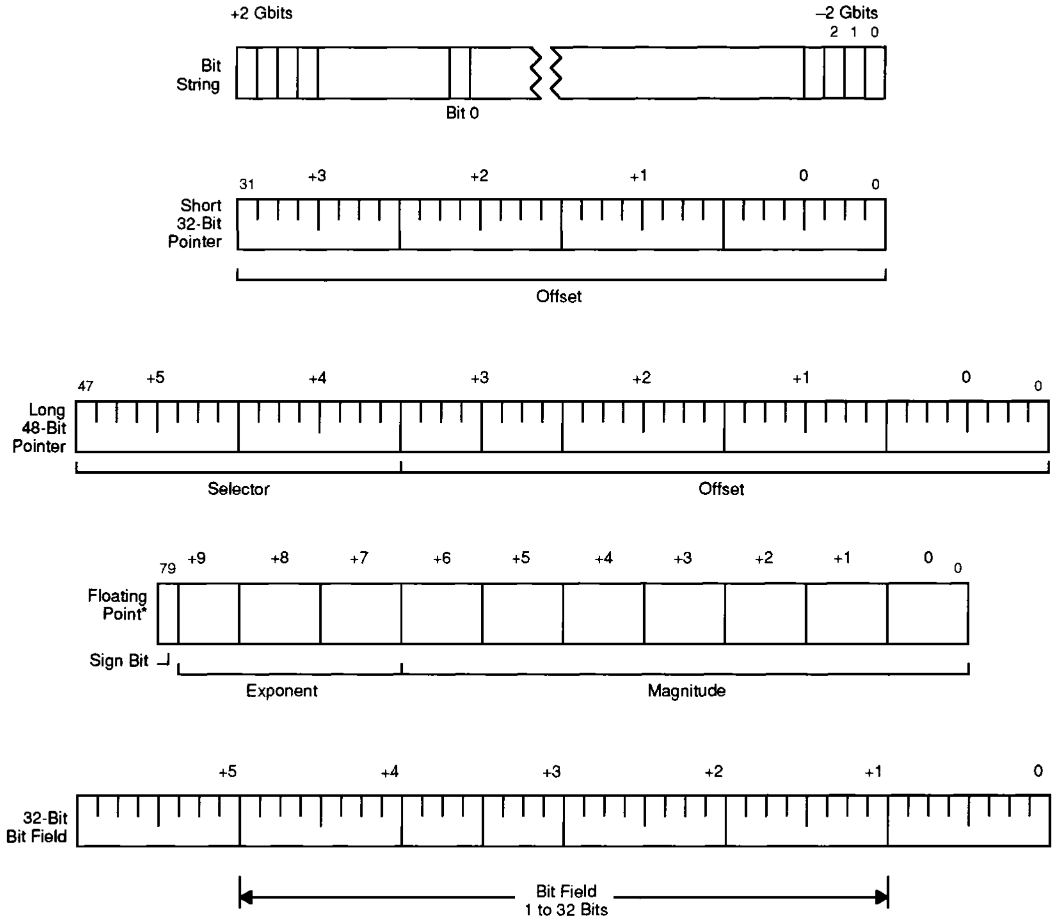
Table 3. Base and Index Registers for 16- and 32-Bit Addresses

	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



15021B–013

Figure 10. Supported Data Types



*Supported by 387DX-compatible math coprocessor.

15021B–013

Figure 10. Supported Data Types (continued)

Memory Organization

Introduction

Memory on the Am386DX/DXL microprocessor is divided up into 8-bit quantities (Bytes), 16-bit quantities (Words), and 32-bit quantities (Dword). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or Dword is the byte address of the low-order byte.

In addition to these basic data types, the Am386DX/DXL microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4-Kb pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The Am386DX/DXL microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

Address Spaces

The Am386DX/DXL microprocessor has three distinct address spaces: logical, linear, and physical. A logical address (also known as a virtual address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (Base, Index, Displacement) discussed in Section Memory Address Modes into an effective address. Since each task on Am386DX/DXL CPU has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 Gb (2^{32} bits), this gives a total of 2^{46} bits or 64 tb 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 11 shows the relationship between the various address spaces.

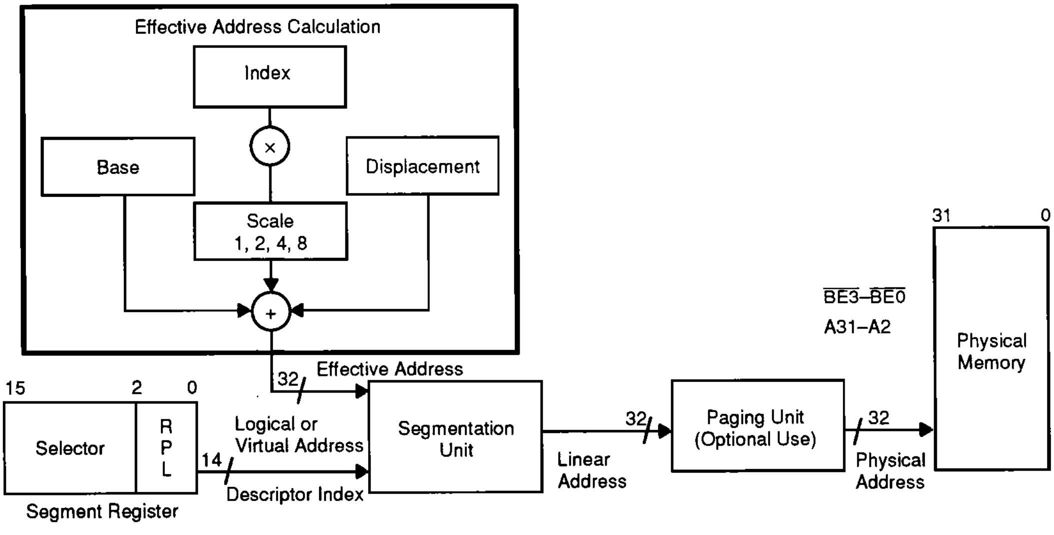


Figure 11. Address Translation

Segment Register Usage

The main data structure used to organize memory is the segment. On the Am386DX/DXL 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 Gb (2^{32} bytes).

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 4 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register; and Instruction fetches use the CS register. The contents of the Instruction Pointer provides the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 4. The override prefixes also allow the use of the ES, FS, and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a 4-Gb 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 Protected Mode Architecture.

I/O Space

The Am386DX/DXL microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the Am386DX/DXL CPU also supports memory-mapped peripherals. The I/O space consists of 64 Kb and can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports that add up to less than 64 Kb. The 64Kb 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/$\overline{\text{IO}}$ 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/$\overline{\text{IO}}$ pin to be driven Low.

I/O port addresses 00F8H through 00FFH are reserved.

Interrupts

Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to

Table 4. Segment Register Selection Rules

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

handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT n instruction, the processor treats software interrupts as exceptions.

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

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. Faults are exceptions that are detected and serviced before the execution of the faulting instruction. A fault 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 then the Am386DX/DXL 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.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 5 summarizes the possible interrupts for the Am386DX/DXL microprocessor and shows where the return address points.

The Am386DX/DXL microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section Real Mode Introduction), 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 Introduction). Of the 256 possible interrupts, 32 are Reserved for Future Use, the remaining 224 are free to be used by the system designer.

Table 5. Interrupt Vector Assignments

Function	Interrupt Number	Instructions Which Can Cause Exceptions	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	Yes	FAULT
Debug Exception	1	Any instruction	Yes	TRAP*
NMI Interrupt	2	INT 2 or NMI	No	NMI
One Byte Interrupt	3	INT	No	TRAP
Interrupt on Overflow	4	INTO	No	TRAP
Array Bounds Check	5	BOUND	Yes	FAULT
Invalid Op-Code	6	Any illegal instruction	Yes	FAULT
Device Not Available	7	ESC, WAIT	Yes	FAULT
Double Fault	8	Any instruction that can generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	No	ABORT
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 for Future Use	15			
Coprocessor Error	16	ESC, WAIT	Yes	FAULT
Reserved for Future Use	17–31			
Two Byte Interrupt	0–255	INT n	No	TRAP

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

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 Am386DX/DXL microprocessor that 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 Am386DX/DXL 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.

Maskable Interrupt

Maskable interrupts are the most common way the Am386DX/DXL 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, which allows interrupts during long string moves). When an interrupt occurs, the processor reads an 8-bit vector supplied by the hardware that identifies the source of the interrupt (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section Functional Data.

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

Non-Maskable Interrupt

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

While executing the NMI servicing procedure, the Am386DX/DXL microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

Software Interrupts

A third type of interrupt/exception for the Am386DX/DXL 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 the Debugging Support section.

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 Am386DX/DXL microprocessor invokes the NMI service routine first. If after the NMI service routine has been invoked, maskable interrupts are still enabled, then the Am386DX/DXL CPU invokes the appropriate interrupt service routine.

Table 6a. Am386DX/DXL Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the Am386DX/DXL microprocessor if in the course of executing an instruction, the Am386DX/DXL CPU detects a problematic condition. The Am386DX/DXL microprocessor then immediately invokes the appropriate exception service routine. The state of the Am386DX/DXL CPU is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

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

As the Am386DX/DXL microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 6b. This cycle is repeated as each instruction is executed and occurs in parallel with instruction decoding and execution.

Instruction Restart

The Am386DX/DXL microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (Exception Categories 4 through 10 in Table 6b), the Am386DX/DXL device invokes the appropriate exception service routine. The Am386DX/DXL microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 6c. Note that all such cases are easily avoided by proper design of the operating system.

Table 6b. Sequence of Exception Checking

Consider the case of the Am386DX/DXL 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 and 13).
5. Check for Paging Faults that prevented fetching the entire next instruction (Exception 14).
6. Check for Faults decoding the next instruction [Exception 6 if illegal op-code; Exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section Protection and I/O Permission Bitmap); or Exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL = 0)].
7. If WAIT op-code, check if TS = 1 and MP = 1 (Exception 7 if both are 1).
8. If ESCAPE op-code for numeric coprocessor, check if EM = 1 or TS = 1 (Exception 7 if either are 1).
9. If WAIT op-code or ESCAPE op-code for numeric coprocessor, check $\overline{\text{ERROR}}$ input signal (Exception 16 if $\overline{\text{ERROR}}$ input 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 that 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.

Table 6c. Conditions Preventing Instruction Restart

1. An instruction causes a task switch to a task whose Task State Segment (TSS) is partially not present. (An entire not present TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kb or less).
2. A coprocessor operand wraps around the top of a 64-Kb segment or a 4-Gb segment and spans three pages; and the page holding the middle portion of the operand is not present. This condition can be avoided by starting any segments containing coprocessor operands at a page boundary if the segments are approximately 64–200 Kb or larger (i.e., large enough for wraparound of the coprocessor operand to possibly occur).

Note that these conditions are avoided by using the operating system designs mentioned in this table.

Double Fault

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

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

Double Page faults however do not raise the Double Fault exception. If a second Page Fault occurs while the processor is attempting to enter the service routine for the first time, then the processor will invoke the Page Fault (Exception 14) handler a second time rather than the Double Fault (Exception 8) handler. A subsequent fault, though, will lead to shutdown.

When a Double Fault occurs, the Am386DX/DXL microprocessor invokes the exception service routine for Exception 8.

Reset and Initialization

When the processor is initialized or Reset, the registers have the values shown in Table 7. The Am386DX/DXL microprocessor will then start executing instructions near the top of physical memory, at location FFFFFFF0H. When the first Inter-Segment Jump or Call is executed, address lines A31–A20 will drop Low for CS-relative memory cycles, and the Am386DX/DXL microprocessor will only execute instructions in the lower 1 Mb 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 Am386DX/DXL microprocessor to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350- and 450-CLK2 periods after Reset becomes inactive, the Am386DX/DXL device will start executing instructions at the top of physical memory.

Table 7. Register Values after Reset

Flag Word	UUUU0002H	Note 1
Machine Status Word (CR0)	UUUUUUU0H	Note 2
Instruction Pointer	0000FFF0H	
Code Segment	F000H	Note 3
Data Segment	0000H	
Stack Segment	0000H	
Extra Segment (ES)	0000H	
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
DX Register	Component and Stepping ID	Note 5
All Other Registers	Undefined	Note 4

Notes:

1. EFLAGS Register. The upper 14 bits of the EFLAGS register are undefined, VM (Bit 17) and RF (Bit 16) and 0 (Bit 15) are all other defined flag bits.
2. CR0: (Machine Status Word). All of the defined fields in the CR0 are 0 (PG Bit 31, TS Bit 3, EM Bit 2, MP Bit 1, and PE Bit 0).
3. The code Segment Register (CS) will have its Base Address set to FFFF0000H and Limit set to 0FFFFH.
4. All undefined bits are Reserved for Future Use and should not be used.
5. DX register always holds component and stepping identifier (see Section Component and Revision Identifiers). EAX register holds self-test signature if self-test was requested (see Section Self-Test Signature).

Testability

Self-Test

The Am386DX/DXL microprocessor has the capability to perform a self-test. The self-test checks the function of all the Control ROM and most of the non-random logic of the part. Approximately one-half of the Am386DX/DXL microprocessor can be tested during self-test.

Self-Test is initiated on the Am386DX/DXL microprocessor when the RESET pin transitions from High to Low, and the $\overline{BUSY}$ pin is Low. The self-test takes about 2^{19} clocks or approximately 26 ms with a 20-MHz Am386DX/DXL device. At the completion of self-test, the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register are zero (0). If the results of EAX are not zero then the self-test has detected a flaw in the part.

TLB Testing

The Am386DX/DXL microprocessor provides a mechanism for testing the Translation Look-Aside Buffer (TLB) if desired. This particular mechanism is unique to the Am386DX/DXL CPU and may not be continued in the same way in future processors. When testing the TLB, paging must be turned off (PG = 0 in CR0) to enable the TLB testing hardware and avoid interference with the test data being written to the TLB.

There are two TLB testing operations:

1. Write entries into the TLB; and,
2. Perform TLB lookups. Two test registers, shown in Figure 12, are provided for the purpose of testing. TR6 is the test command register and TR7 is the test data register. The fields within these registers are defined below.

C: This is the command bit. For a write into TR6 to cause an immediate write into the TLB entry, write a 0 to this bit. For a write into TR6 to cause an immediate TLB lookup, write a 1 to this bit.

Linear Address: This is the tag field of the TLB . On a TLB write, a TLB entry is allocated to this linear address and the rest of that TLB entry is set per the value of TR7 and the value just written into TR6. On a TLB lookup, the TLB is interrogated per this value and if one and only one TLB entry matches, the rest of the fields of TR6 and TR7 are set from the matching TLB entry.

Physical Address: This is the data field of the TLB. On a write to the TLB, the TLB entry allocated to the linear address in TR6 is set to this value. On a TLB lookup, the data field (physical address) from the TLB is read out to here.

PL: On a TLB write, PL = 1 causes the REP field of TR7 to select which of four associative blocks of the TLB is to be written, but PL = 0 allows the internal pointer in the paging unit to select which TLB block is written. On a TLB lookup, the PL bit indicated whether the lookup was a hit (PL gets set to 1) or a miss (PL gets reset to 0).

V: The valid bit for this TLB entry. All valid bits can also be cleared by writing to CR3.

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

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

W, $\overline{W}$: The writable bit for/from the TLB entry.

For D, U, and W, both the attribute and its complement are provided as tag bits to permit the option of a don't care on TLB lookups. The meaning of these pairs of bits is given in the following table.

X	$\overline{X}$	Effect During TLB Lookup	Value of Bit X after TLB Write
0	0	Miss All	Bit X becomes undefined
0	1	Match if X = 0	Bit X becomes 0
1	0	Match if X = 1	Bit X becomes 1
1	1	Match All	Bit X becomes undefined

For writing a TLB entry:

1. Write TR7 for the desired physical address, PL, and REP values; and,
2. Write TR6 with the appropriate linear address, etc., (be sure to write C = 0 for write command).

For looking up (reading) a TLB entry:

1. Write TR6 with the appropriate linear address (be sure to write C = 1 for lookup command); and,
2. Read TR7 and TR6. If the PL bit in TR7 indicates a hit, then the other values reveal the TLB contents. If PL indicates a miss, then the other values in TR7 and TR6 are indeterminate.

Debugging Support

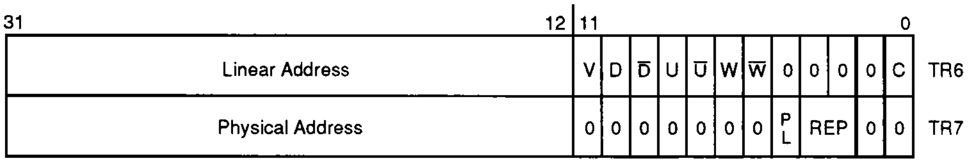
The Am386DX/DXL microprocessor provides several features that simplify the debugging process.

The three categories of on-chip debugging aids are:

1. The code execution breakpoint op-code (0CCH);
2. The single-step capability provided by the TF bit in the flag register; and,
3. The code and data breakpoint capability provided by the Debug Registers DR3–DR0, DR6, and DR7.

Breakpoint Instruction

A single-byte op-code breakpoint instruction is available for use by software debuggers. The breakpoint op-code 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 op-code 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.



Note: 0 indicates "Reserved for Future Use." Do not define; see Section Compatibility.

15021B–015

Figure 12. Test Registers

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 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.

Debug Registers

The Debug Registers are an advanced debugging feature of the Am386DX/DXL 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 op-code.

The Am386DX/DXL 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, breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to Exception 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 13. The breakpoint addresses specified are 32-bit linear addresses. Am386DX/DXL 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. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

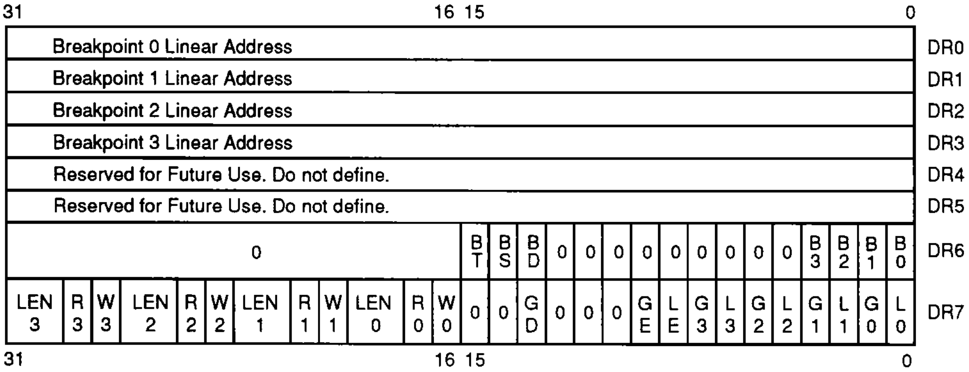
Debug Control Register (DR7)

A Debug Control Register, DR7, shown in Figure 13, allows several debug control functions, such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows.

LENi (Breakpoint Length Specification Bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows.

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i = 0 – 3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	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.



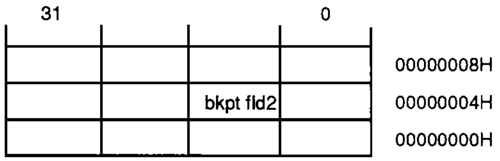
Note: 0 indicates "Reserved for Future Use." Do not define; see Section Compatibility.

Figure 13. Debug Registers 15021B–016

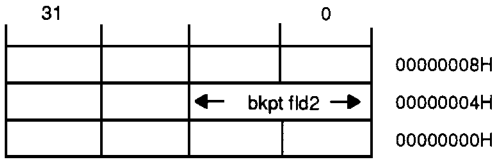
The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries and 4-byte breakpoint fields begin on Dword boundaries.

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

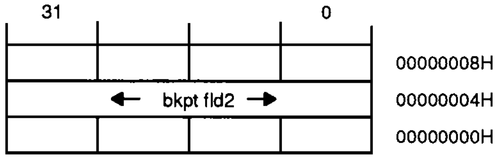
DR2 = 00000005H; LEN2 = 00B



DR2 = 00000005H; LEN2 = 01B



DR2 = 00000005H; LEN2 = 11B



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.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that instruction execution breakpoints are taken as faults (i.e., before the instruction executes), but data breakpoints are taken as traps (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access 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 will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an Exception 1 fault will occur before the instruction is executed.

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

GD (Global Debug Register Access Detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against any Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger (or ICE-386) can have full control over the Debug Register resources when required. The GD bit, when set, causes an Exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the Exception 1 handler is invoked, allowing the Exception 1 handler free access to the debug registers.

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

If either GE or LE is set, any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the Am386DX/DXL microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

If exact data breakpoint match is not selected, data breakpoints may not be reported until several instructions later or may not be reported at all. When enabling a data breakpoint, it is therefore recommended to enable the exact data breakpoint match.

When the Am386DX/DXL 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 Am386DX/DXL microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that instruction execution breakpoints are always reported exactly, whether or not exact data breakpoint match is selected.

Gi and Li (Breakpoint Enable, Global and Local)

If either Gi or Li is set, then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set and the Am386DX/DXL microprocessor detects the breakpoint condition, then the Exception 1 handler is invoked.

When the Am386DX/DXL 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 enabled under software control.

All Am386DX/DXL microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

Debug Status Register (DR6)

A Debug Status Register, DR6, shown in Figure 13, allows the Exception 1 handler to easily determine why it was invoked. Note the Exception 1 handler can be invoked as a result of one of several events.

1. DR0 Breakpoint fault/trap.
2. DR1 Breakpoint fault/trap.
3. DR2 Breakpoint fault/trap.
4. DR3 Breakpoint fault/trap.
5. Single-step (TF) trap.
6. Task switch trap.
7. Fault due to attempted debug register access when GD = 1.

The Debug Status Register contains single-bit flags for each of the possible events invoking Exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of Exception 1.

The fields within the Debug Status Register, DR6 are as follows.

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

Four breakpoint indicator flags, 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 breakpoint is detected, the processor will invoke the Exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred or as a trap if a data breakpoint occurred.

Important Note: A flag, Bi, is set whenever the hardware detects a match condition on enabled breakpoint i. Whenever a match is detected on at least one enabled breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled or not. Therefore, the Exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to enabled breakpoints (Li or Gi set) are true indications of why the Exception 1 handler was invoked.

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

This bit is set if the Exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the Exception 1 handler is invoked, allowing handler access to the debug registers.

BS (Debug Trap Due to Single-Step)

This bit is set if the Exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping). See Section Single-Step Trap.

BT (Debug Trap Due to Task Switch)

This bit is set if the Exception 1 handler was invoked due to a task switch occurring to a task having an Am386DX/DXL microprocessor TSS with the T-bit set. (See Figure 29.) 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.

Use of Resume Flag (RF) In Flag Register

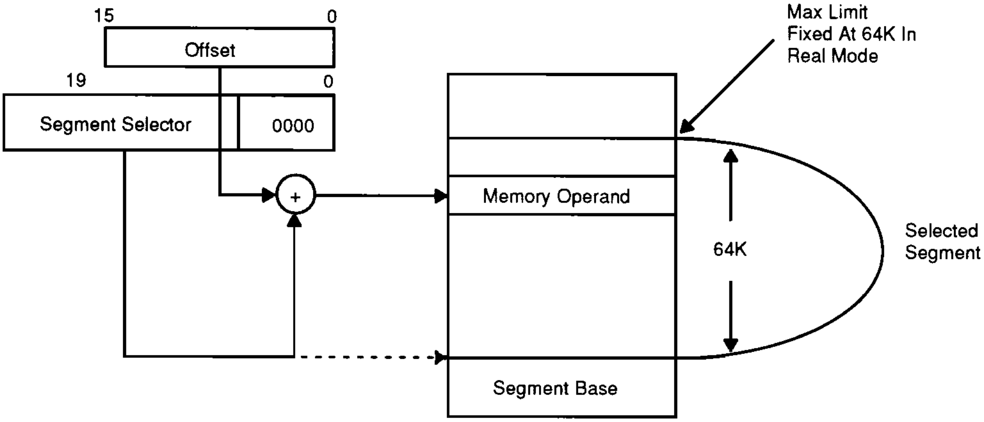
The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the Exception 1 handler returns to a user program at a user address that is also an instruction execution breakpoint. See Section Flags Register.

REAL MODE ARCHITECTURE

Real Mode Introduction

When the processor is reset or powered up, it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the Am386DX/DXL microprocessor. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286.

All of the Am386DX/DXL microprocessor instructions are available in Real Mode (except those instructions listed in Protection and I/O Permission Bitmap). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the Am386DX/DXL CPU in Real Mode is 64 Kb 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.



15021B–017

Figure 14. Real Address Mode Addressing

LOCK Operation

The LOCK prefix on the Am386DX/DXL microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Am386DX/DXL CPU in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The Am386DX/DXL CPU 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.

These are the only instruction forms where the LOCK prefix is legal on the Am386DX/DXL microprocessor.

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

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

Since, on the Am386DX/DXL microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the Am386DX/DXL device. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

Memory Addressing

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

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 that is shifted left by 4 bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, this implies that Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64-Kb long and may be read, written, or executed. The Am386DX/DXL microprocessor will generate an Exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH; for example, a word with a low byte at FFFFH and the high byte at 0000H).

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

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 one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

Interrupts

Many of the exceptions shown in Table 5 and discussed in Section Interrupts are not applicable to Real Mode operation; in particular, Exceptions 10, 11, and 14 will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode. Table 8 identifies these exceptions.

Table 8. Other Exceptions in Real Mode

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

Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, $\overline{\text{FLT}}$, INTR with interrupts enabled (IF = 1), or RESET will force the Am386DX/DXL microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

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

- An interrupt or an exception occur (Exception 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt);
- A CALL, INT, or PUSH instruction attempts to wrap around the stack segment when SP is not even (e.g., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

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

PROTECTED MODE ARCHITECTURE

Introduction

The complete capabilities of the Am386DX/DXL microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to 4 Gb (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 tb or 2^{46} bytes). In addition, Protected Mode allows the Am386DX/DXL CPU to run all of the existing 8086 and 80286 software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the Am386DX/DXL CPU remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main differences between Protected Mode and Real Mode from a programmer's view is the increased address space and a different addressing mechanism.

Addressing Mechanism

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

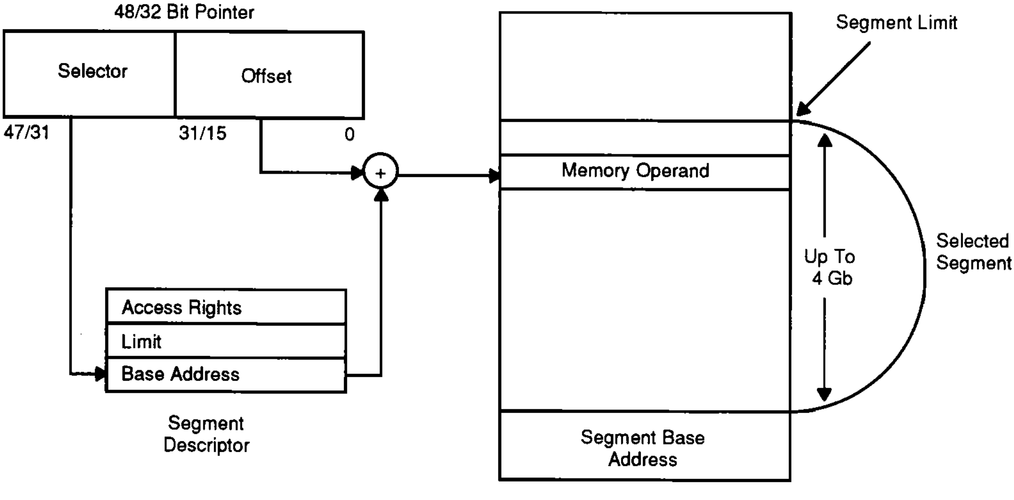
The difference between the two modes lies in calculating the base address. In Protected Mode, the selector is used to specify an index into an operating system defined table (see Figure 15). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base 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 Am386DX/DXL 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 16 shows the complete Am386DX/DXL device addressing mechanism with paging enabled.

Segmentation

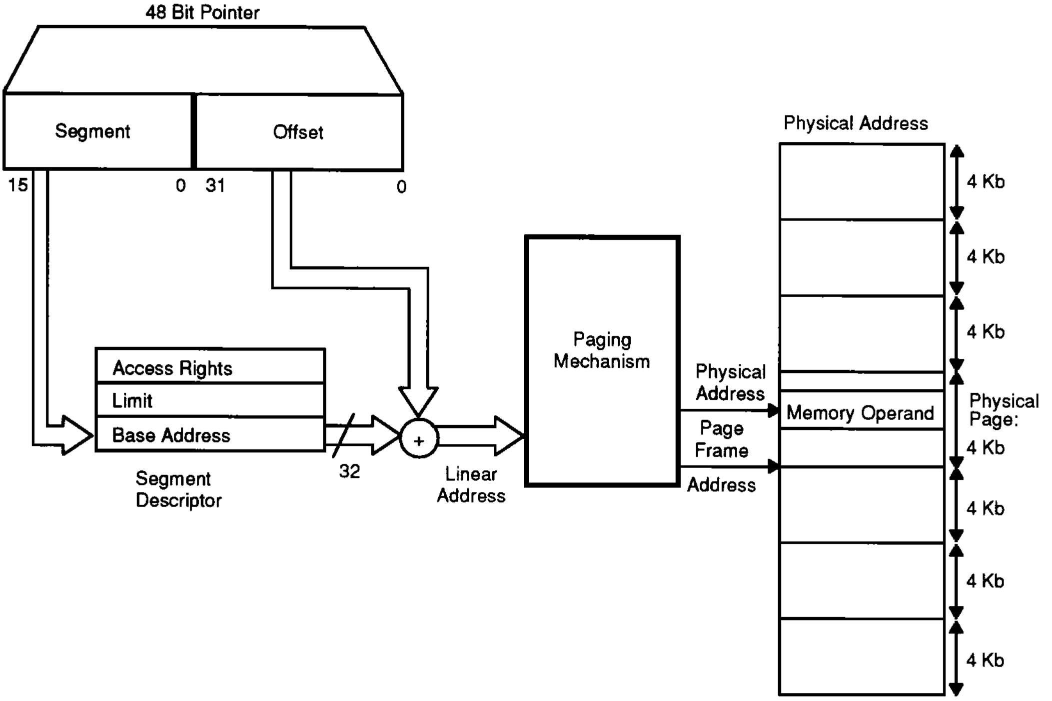
Segmentation Introduction

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



15021B–018

Figure 15. Protected Mode Addressing



15021B–019

Figure 16. Paging and Segmentation

Terminology

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

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

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

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

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

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

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

Descriptor Tables

Descriptor Tables Introduction

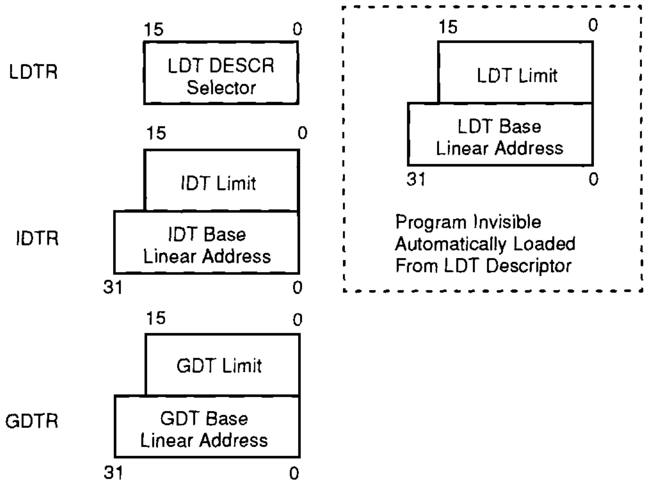
The descriptor tables define all of the segments which are used in an Am386DX/DXL microprocessor system. There are three types of tables on the Am386DX/DXL microprocessor that hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kb. Each table can hold up to 8192 eight 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 of the tables has a register associated with it: the GDTR, LDTR, and the IDTR (see Figure 17). The LGDT, LLDT, and LIDT instructions load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT instructions store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

Global Descriptor Table

The Global Descriptor Table (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 Am386DX/DXL microprocessor contains a GDT. Generally, the GDT contains code and data segments used by the operating systems and task state segments and descriptors for the LDTs in a system.

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



15021B–020

Figure 17. Descriptor Table Registers

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 may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments 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 a Local Descriptor Table descriptor in the GDT.

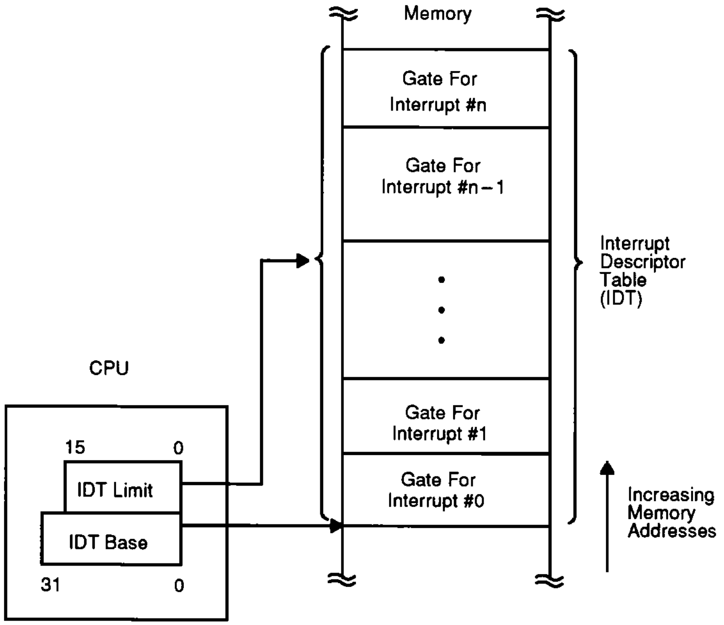
Interrupt Descriptor Table

The third table needed for Am386DX/DXL microprocessor systems is the Interrupt Descriptor Table (see Figure 18). The IDT contains the descriptors that point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32, Reserved for Future Use, 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 Interrupts.)

Descriptors

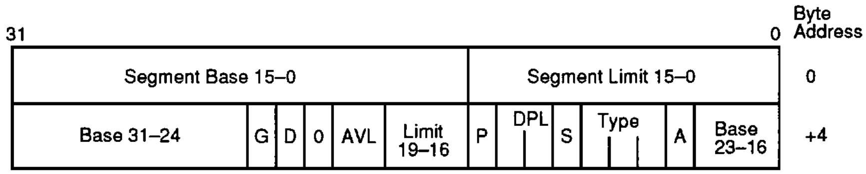
Descriptor Attribute Bits

The object to which the segment selector points 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 19 shows the general format of a descriptor. All segments on the Am386DX/DXL microprocessor have three attribute fields in common: the P bit, the DPL bit, and the S bit. The Present P bit is 1 if the segment is loaded in physical memory; if P = 0 then any attempt to access this segment causes a not present exception (Exception 11). The Descriptor Privilege Level (DPL) is a 2-bit field that specifies the protection levels 0–3 associated with a segment.



15021B–021

Figure 18. Interrupt Descriptor Table Register Use



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

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

15021B–022

Figure 19. General Format of Segment Descriptors

The Am386DX/DXL microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment S bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1, then the segment is either a code or data segment; if it is 0, then the segment is a system segment.

Am386DX/DXL Microprocessor Code and Data Descriptors (S = 1)

Figure 20 shows the general format of a code and data descriptor and Table 9 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 systems to keep usage statistics on a given segment. The G bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. Am386DX/DXL microprocessor segments can be 1 Mb long with byte granularity (G = 0) or 4 Gb with page granularity (G = 1), (i.e., 2^{20} pages—each page is 4 Kb in length). The granularity is totally unrelated to paging. An Am386DX/DXL CPU system can consist of segments with byte granularity and page granularity, whether or not paging is enabled.

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

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

The D bit indicates the default length for operands and effective addresses. If D = 1, then 32-bit operands and 32-bit addressing modes are assumed. If D = 0, then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the Am386DX/DXL 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 Protection).

31							0	
Segment Base 15–0						Segment Limit 15–0		0
Base 31–24	G	D/B	0	AVL	Limit 19–16	Access Rights Byte	Base 23–16	+4

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

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

15021B–023

Figure 20. Code and Data Segment Descriptors

Table 9. Access Rights Byte Definition for Code and Data Descriptions

Bit Position	Name	Function		
7	Present (P)	P = 1 P = 0	Segment is mapped into physical memory. No mapping to physical memory exists, base and limit are not used.	
6–5	Descriptor Privilege Levels (DPL)		Segment privilege attribute used in privilege tests.	
4	Segment Descriptor (S)	S = 1 S = 0	Code or Data (includes stacks) segment descriptor. System Segment Descriptor or Gate Descriptor.	
3	Executable (E)	E = 0	Descriptor type is data segment.	If Data Segment (S = 1, E = 0)
2	Expansion Direction (ED)	ED = 0 ED = 1	Expand up segment, offsets must be ≤ limit. Expand down segment, offsets must be > limit.	
1	Writeable (W)	W = 0 W = 1	Data segment may not be written into. Data segment may be written into.	
3	Executable (E)	E = 1	Descriptor type is code segment.	If Code Segment (S = 1, E = 1)
2	Conforming (C)	C = 1	Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
1	Readable (R)	R = 0 R = 1	Code segment may not be read. Code segment may be read.	
0	Accessed (A)	A = 0 A = 1	Segment has not been accessed. Segment selector has been loaded into segment register or used by selector test instructions.	

Segments identified as data segments (E = 0, S = 1) are used for two types of Am386DX/DXL microprocessor segments: stack and data segments. The expansion direction (ED) bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment, all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

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

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

System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 21 shows the general format of system segment descriptors, and the various types of system segments. The Am386DX/DXL microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zeros.

LDT Descriptors (S = 0, Type = 2)

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

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

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a TSS. A TSS in turn 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 whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The Type field also indicates if the segment contains a 80286 or an Am386DX/DXL microprocessor TSS. The Task Register (TR) contains the selector that points to the current TSS.

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 Protection), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

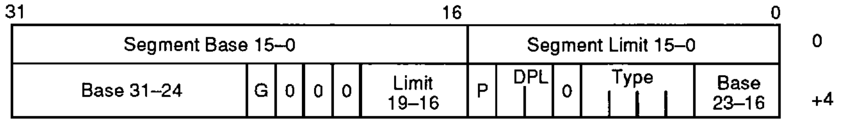
Figure 22 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte; a long pointer (selector and offset) that points to the start of a routine; and a word count that specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

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

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

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

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes Exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 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 22.

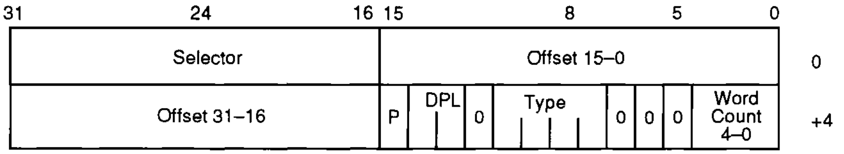


Type	Definition	Type	Definition
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available Am386DX/DXL CPU TSS
2	LDT	A	Undefined (Reserved)
3	Busy 80286 TSS	B	Busy Am386DX/DXL CPU TSS
4	80286 Call Gate	C	Am386DX/DXL CPU Call Gate
5	Task Gate (for 80286 or Am386DX/DXL CPU Task)	D	Undefined (Reserved)
6	80286 Interrupt Gate	E	Am386DX/DXL CPU Interrupt Gate
7	80286 Trap Gate	F	Am386DX/DXL CPU Trap Gate

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

15021B–024

Figure 21. System Segments Descriptors



Gate Descriptors Fields

Name	Value	Description
Type	4	80286 Call Gate
	5	Task Gate (for 80286 or Am386DX/DXL CPU Task)
	6	80286 Interrupt Gate
	7	80286 Trap Gate
	C	Am386DX/DXL CPU Call Gate
	E	Am386DX/DXL CPU Interrupt Gate
	F	Am386DX/DXL 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 Am386DX/DXL 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 state segment for task gate
DESTINATION OFFSET	Offset 16-bit 80286 32-bit Am386DX/DXL CPU	Entry point within the target code segment

15021B–025

Figure 22. Gate Descriptor Formats

Difference Between Am386DX/DXL Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and Am386DX/DXL microprocessor, the Am386DX/DXL CPU supports all of the 80286 segment descriptors. Figure 23 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and Am386DX/DXL device descriptor formats are that the values of the type fields and the limit and base address fields have been expanded for the Am386DX/DXL device. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the Am386DX/DXL microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

By supporting 80286 system segments, the Am386DX/DXL microprocessor is able to execute 80286 application programs on an Am386DX/DXL CPU operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which are Am386DX/DXL microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is an 80286-style descriptor.

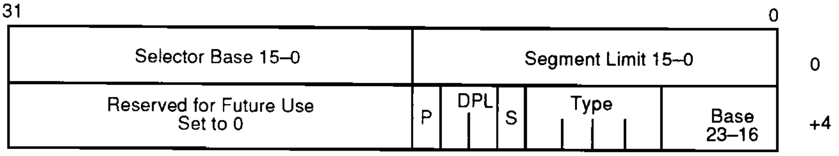
The only other differences between 80286-style descriptors and Am386DX/DXL microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for Am386DX/DXL device call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0, then PUSHes are 16 bits, if B = 1, then PUSHes are 32 bits.

Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 24. 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.

Segment Descriptor Cache

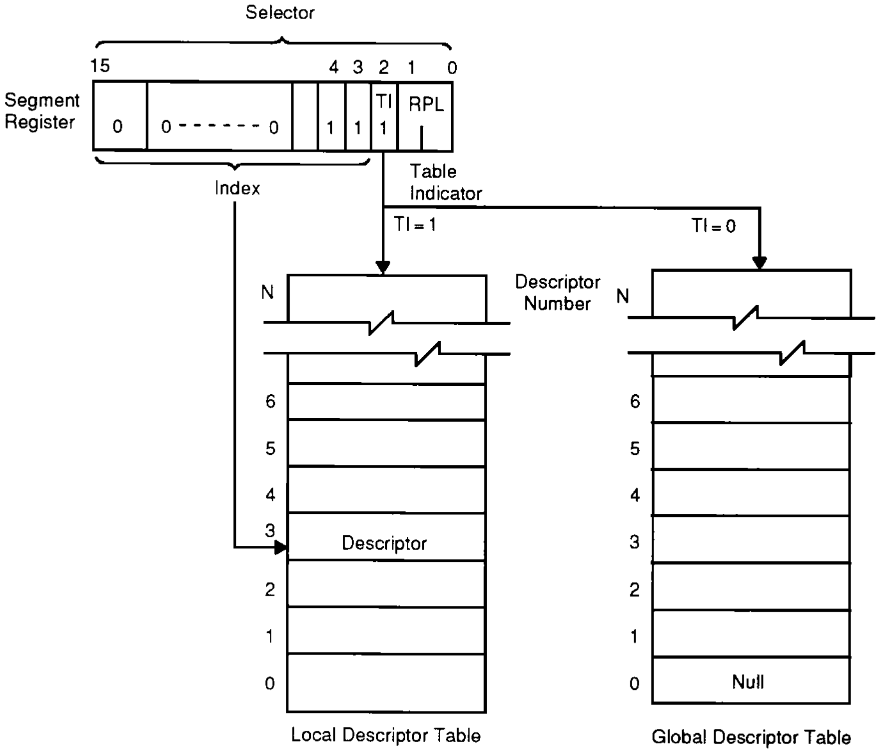
In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that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



Base	Base Address of the Segment
Limit	The length of the Segment
P	Present Bit: 1 = Present, 0 = Not Present
DPL	Descriptor Privilege Levels 0–3
S	System Descriptor: 0 = System, 1 = User
Type	Type of Segment

15021B–026

Figure 23. 80286 Code and Data Segment Descriptors



15021B–027

Figure 24. Example Descriptor Selection

Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the operating mode of the Am386DX/DXL microprocessor. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 25.

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 that 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 op-codes may be executed.

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 26. 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.

Segment Descriptor Cache Register Contents

	32-Bit Base (Updated During Selector Load into Segment Register)	32-Bit Limit (Fixed)	Other Attributes (Fixed)									
	BASE	LIMIT	Present	Privilege Level	Accessed	Granularity	Expansion Direction	Readable	Writeable	Executable	Stack Size	Conforming Privilege
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	–	–

Key: Y = Yes
N = No
0 = Privilege level 0
1 = Privilege level 1
2 = Privilege level 2
3 = Privilege level 3
U = Expand up

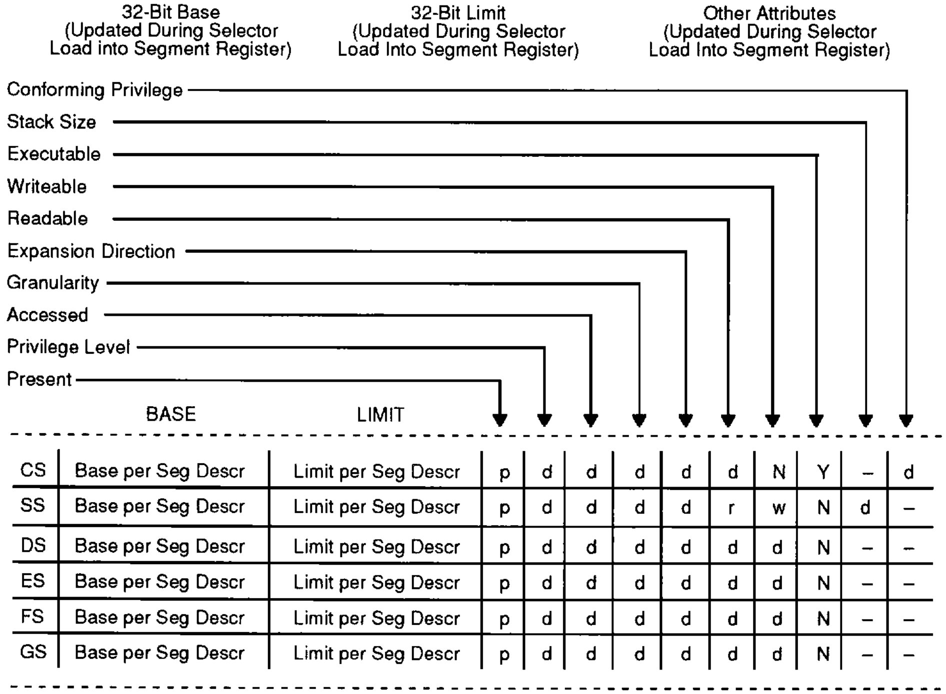
D = Expand down
B = Byte granularity
P = Page granularity
W = Push/pop 16-bit words
F = Push/pop 32-bit Dwords
– = Does not apply to that segment cache register

*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (e.g., intersegment CALL, or intersegment JMP, or INT). (See Figure 27 example.)

15021B–028

Figure 25. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

Segment Descriptor Cache Register Contents



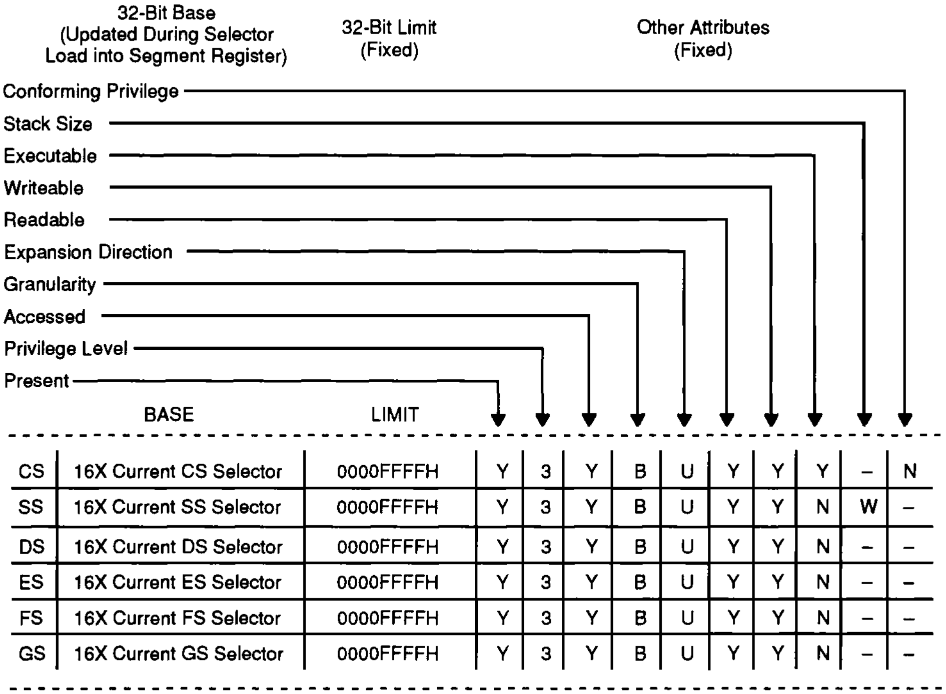
	BASE	LIMIT	Present	Privilege Level	Accessed	Granularity	Expansion Direction	Readable	Writeable	Executable	Stack Size	Conforming Privilege
CS	Base per Seg Descr	Limit per Seg Descr	p	d	d	d	d	d	N	Y	–	d
SS	Base per Seg Descr	Limit per Seg Descr	p	d	d	d	d	r	w	N	d	–
DS	Base per Seg Descr	Limit per Seg Descr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

Key: Y = Fixed Yes
N = Fixed No
d = Per segment descriptor
p = Per segment descriptor; descriptor must indicate "present" to avoid Exception 11 (Exception 12 in case of SS)
r = Per segment descriptor, but descriptor must indicate "readable" to avoid Exception 13 (special case for SS)
w = Per segment descriptor, but descriptor must indicate "writeable" to avoid Exception 13 (special case for SS)
– = Does not apply to that segment cache register

15021B–029

Figure 26. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 27. 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, level 3, to allow trapping of all IOPL-sensitive instructions and level 0 only instructions.



Segment Descriptor Cache Register Contents

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

15021B–030

Figure 27. Segment Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

Protection

Protection Concepts

The Am386DX/DXL microprocessor has four levels of protection that 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 the use of complex external hardware and software, the Am386DX/DXL CPU provides the protection on a page basis when paging is enabled (see Section Page Level Protection).

The four-level hierarchical privilege system is illustrated in Figure 28. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the Am386DX/DXL microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

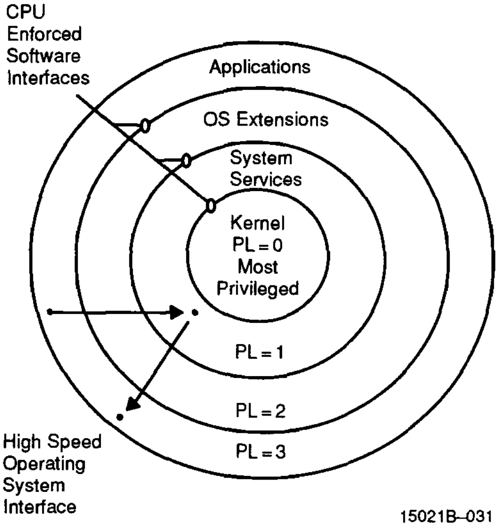


Figure 28. Four-Level Hierarchical Protection

Rules of Privilege

The Am386DX/DXL microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

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

Privilege Levels

Task Privilege

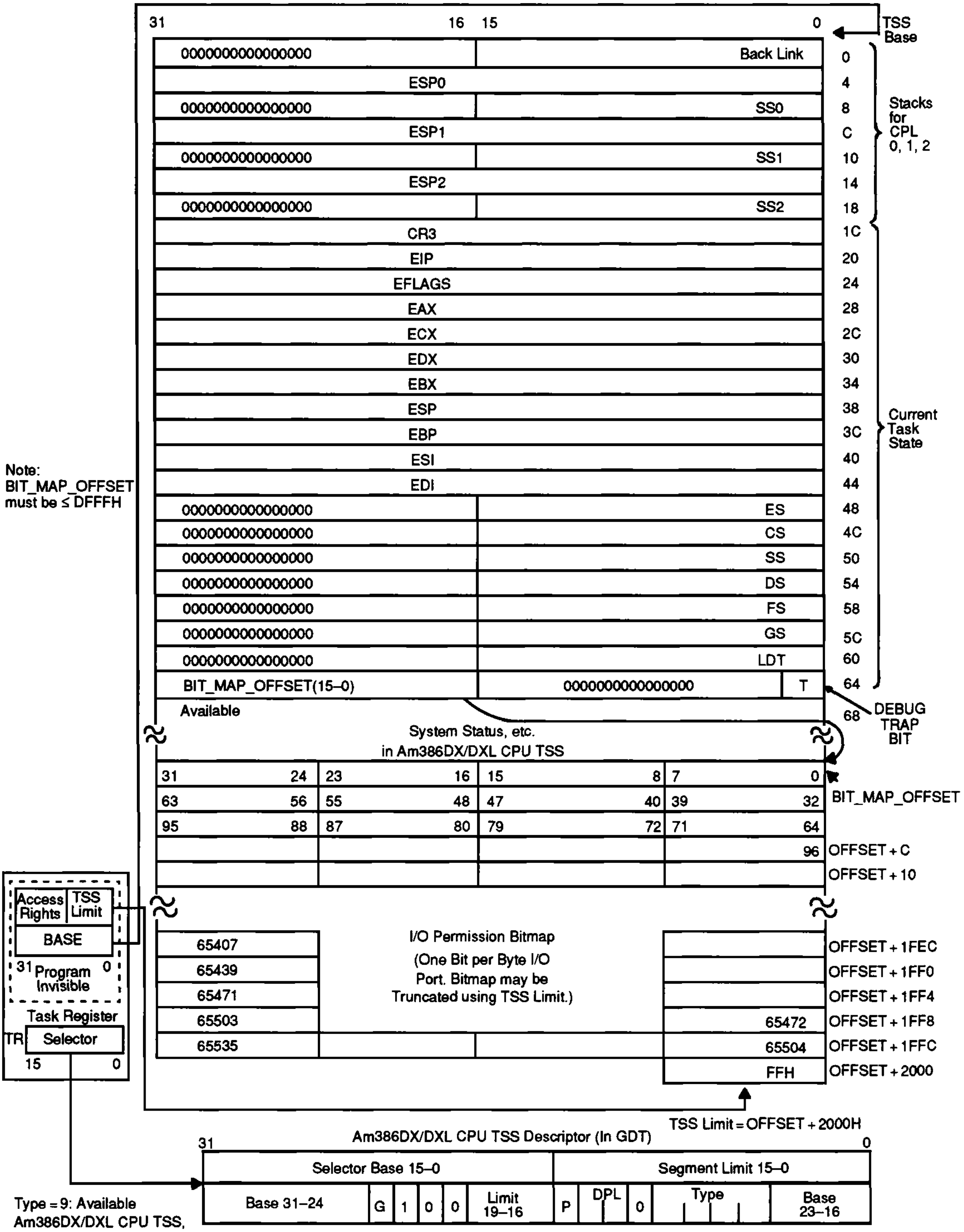
At any point in time, a task on the Am386DX/DXL microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level (see Section Privilege Level Transfers). Thus, an application program running at PL=3 may call an operating system routine at PL=1 (via a gate) that would cause the task's CPL to be set to 1 until operating system routine is finished.

Selector Privilege (RPL)

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

I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAGS 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 Am386DX/DXL CPU TSS, the I/O Permission Bitmap (part of an Am386DX/DXL microprocessor TSS) is consulted on whether I/O to the port is allowed, or an Exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 29a and 29b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section Protection and I/O Permission Bitmap.

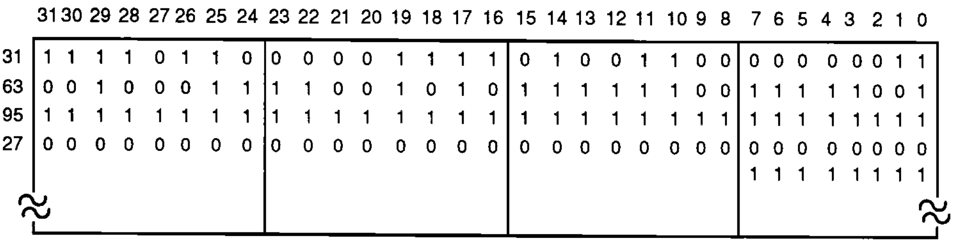


Note:
BIT_MAP_OFFSET must be ≤ DFFFH

Type = 9: Available Am386DX/DXL CPU TSS,
Type = B: Busy Am386DX/DXL CPU TSS

Figure 29a. TSS and TSS Registers

15021B–032a



	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
31	1 1 1 1 0 1 1 0	0 0 0 0 1 1 1 1	0 1 0 0 1 1 0 0	0 0 0 0 0 0 1 1
63	0 0 1 0 0 0 1 1	1 1 0 0 1 0 1 0	1 1 1 1 1 1 0 0	1 1 1 1 1 0 0 1
95	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1
127	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
				1 1 1 1 1 1 1 1

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

15021B–032b

Figure 29b. Sample I/O Permission Bit Map

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an Exception 13 fault instead. These instructions are called IOPL-sensitive instructions and they are CLI and STI. (Note that the LOCK prefix is not IOPL-sensitive on the Am386DX/DXL microprocessor.)

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

Table 10. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level; adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

Privilege Validation

The Am386DX/DXL CPU provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 10 summarizes the selector validation procedures available for the Am386DX/DXL microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling an operating-systems routine at PL = 0 and passing the operating-systems routine a bad pointer that corrupts a data structure belonging to the operating system. If the operating-systems 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.

Descriptor Access

There are basically two types of segment accesses: those involving code segments, such as control transfers; and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used, and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the Am386DX/DXL microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section Rules of Privilege. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally, the privilege validation checks are performed. The CPL is compared to the EPL; if the EPL is more privileged than the CPL, an Exception 13 (General Protection fault) is generated.

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

Table 11. 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	CALL	Call Gate	GDT/LDT
Interrupt within task may change CPL	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (change task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET**, Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

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

Privilege Level Transfers

Intersegment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers, which are summarized in Table 11.

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 will cause an Exception 13 (e.g., JMP through a call gate or IRET from a normal subroutine call).

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

The privilege rules require that:

- Privilege level transitions can only occur via gates.
- JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.
- CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.
- Interrupts handled within the task obey the same privilege rules as CALLs.
- Conforming Code segments are accessible by privilege levels that are the same or less privileged than the conforming-code segment's DPL.
- Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.
- The code segment selected in the gate must be the same or more privileged than the task's CPL.
- Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.
- Task switches can be performed by a CALL, JMP, or INT that references either a task gate or task state segment whose 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 Task Switching). During a JMP or CALL control transfer, the new stack pointer is loaded in the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, use of the lower-privilege 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 intersegment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

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 may only transfer control to a more privileged level.

Call gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an interlevel Am386DX/DXL microprocessor call gate is activated, the following actions occur:

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

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

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

Task Switching

A very important attribute of any multitasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The Am386DX/DXL microprocessor directly supports this operation by providing a task switch instruction in hardware. The Am386DX/DXL CPU task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 ms. Like transfer of control via gates, the task switch operation is invoked by executing an intersegment JMP or CALL instruction that refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 29a) containing the entire Am386DX/DXL microprocessor execution state while a task gate descriptor contains a TSS selector. The Am386DX/DXL CPU supports both 80286 and Am386DX/DXL CPU style TSSs. Figure 30 shows an 80286 TSS. The limit of an Am386DX/DXL microprocessor TSS must be greater than 0064H (002BH for an 80286 TSS) and can be as large as 4 Gb. 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 belonging to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the Am386DX/DXL microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task 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 are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion.

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler.) NT may also be set or cleared by POPF or IRET instructions.

The Am386DX/DXL microprocessor Task State Segment is marked busy by changing the descriptor type field from Type 9H to Type BH. An 80286 TSS is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes an Exception 13.

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

The coprocessor's state is not automatically saved when a task switch occurs, because the incoming task may not use the coprocessor. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the coprocessor's state in a multitasking environment. Whenever the Am386DX/DXL microprocessor switches tasks, it sets the TS bit. The Am386DX/DXL CPU detects the first use of a processor extension instruction after a task switch and causes the processor extension not available Exception 7. The exception handler for Exception 7 may then decide whether to save the state of the coprocessor. A processor extension not present Exception 7 will occur when attempting to execute an ESC or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e., TS = 1 and MP = 1).

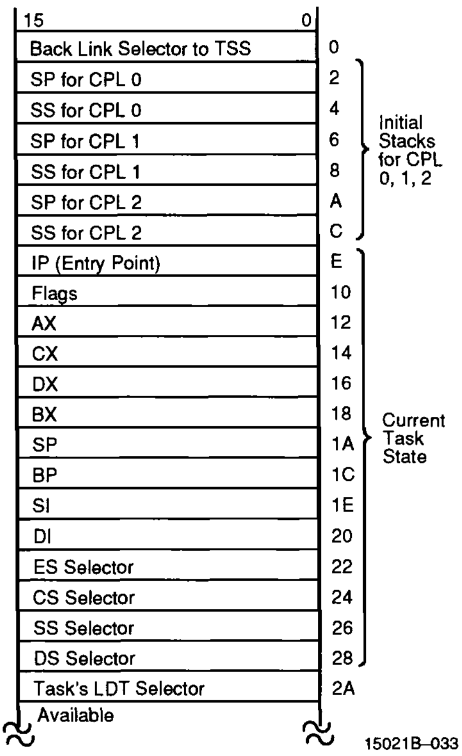


Figure 30. 80286 TSS

The T bit in the Am386DX/DXL microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1, then upon entry to a new task, a debug Exception 1 will be generated.

Initialization and Transition to Protected Mode

Since the Am386DX/DXL microprocessor begins executing in Real Mode immediately after RESET, it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256-bytes long, and GDT must contain descriptors for the initial code and data segments. Figure 31 shows the tables and Figure 32 shows the descriptors needed for a simple Protected Mode Am386DX/DXL microprocessor system. It has a single code and single data/stack segment each 4 Gb long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with PE bit set, via the MOV CR0, R/M instruction. This puts the Am386DX/DXL microprocessor in Protected Mode.

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

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

Paging

Paging Concepts

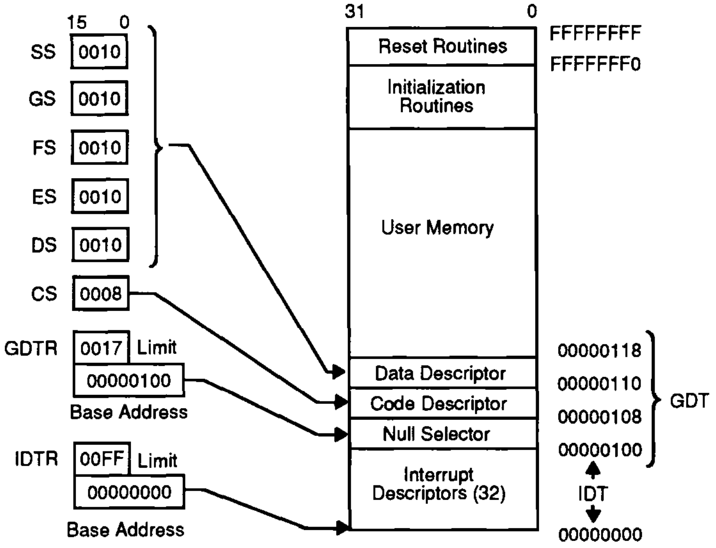
Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation that modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical structure of a program. While segment selectors can be considered the logical name of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

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

Paging Organization

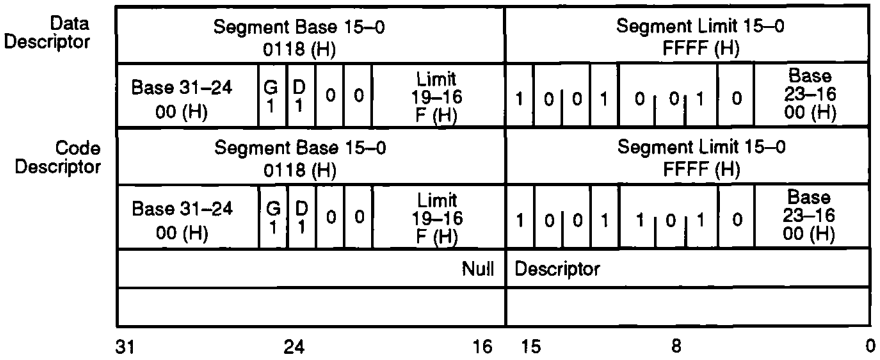
Page Mechanism

The Am386DX/DXL 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 Am386DX/DXL CPU: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Am386DX/DXL CPU paging mechanism are the same size, namely, 4 Kb. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 33 shows how the paging mechanism works.



15021B–034

Figure 31. Simple Protected System



	31					24 / 16	15							8 / 0
Data Descriptor	Segment Base 15–0 0118 (H)						Segment Limit 15–0 FFFF (H)							
	Base 31–24 00 (H)	G 1	D 1	0	0	Limit 19–16 F (H)	1	0	0	1	0	0	1	0	Base 23–16 00 (H)
Code Descriptor	Segment Base 15–0 0118 (H)						Segment Limit 15–0 FFFF (H)							
	Base 31–24 00 (H)	G 1	D 1	0	0	Limit 19–16 F (H)	1	0	0	1	1	0	1	0	Base 23–16 00 (H)
						Null	Descriptor							

31 24 16 15 8 0

15021B–035

Figure 32. GDT Descriptors for Simple System

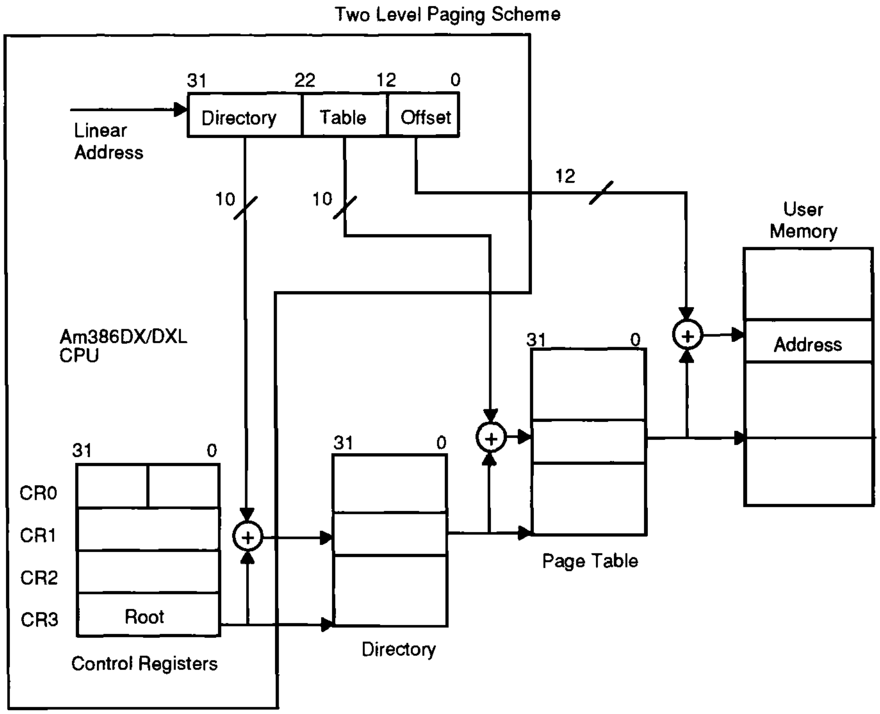
Page Descriptor Base Register

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 will a task switch through a TSS that changes the value of CR0. (See Translation Look-Aside Buffer.)

Page Directory

The Page Directory is 4-Kb 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 34. The upper 10 bits of the linear address (A31–A22) are used as an index to select the correct Page Directory entry.



15021B–036

Figure 33. Paging Mechanism

31 … 12	11 10 9	8	7	6	5	4	3	2	1	0
Page Table Address 31–12	OS Reserved	0	0	D	A	0	0	U – S	R – W	P

Figure 34. Page Directory Entry (Points to Page Table)

15021B–037

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	0	0	U – S	R – W	P

Figure 35. Page Table Entry (Points to Page)

15021B–038

Page Tables

Each Page Table is 4 Kb 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 35). 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.

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. Note that the present bit of the page table entry that points to the page where code is currently being executed should always be set. Code that marks its own page not present should not be written. All of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The A (Accessed) bit 5 is set by the Am386DX/DXL 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 Am386DX/DXL CPU, the microprocessor 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 multi-master systems.

The three bits marked OS Reserved in Figures 34 and 35 (bits 11–9) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the OS Reserved bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

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

Page Level Protection (R/W, U/S Bits)

The Am386DX/DXL microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: user, which corresponds to level 3 of the segmentation based protection, and supervisor, which encompasses all of the other protection levels (0, 1, 2). Programs executing at level 0, 1, or 2 bypass the page protection, although segmentation based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory entry. The U/S and R/W bits in the first level Page Directory Table apply to all pages described by 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 U/S and R/W bits for a given page are obtained by taking the most restrictive of the U/S and R/W bits from the Page Directory Table entries and the Page Table entries and using these bits to address the page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 and the U/S and R/W bits for the Page Table entry were 01, the access rights for the page would be 01, the numerically smaller of the two. Table 12 shows the effect of the U/S and R/W bits on accessing memory.

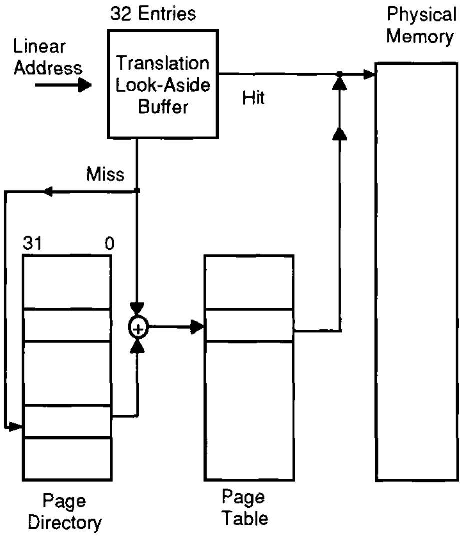
Table 12. Protection Provided by R/W and U/S

U/S	R/W	Permitted Level 3	Permitted Access Levels 0, 1, or 2
0	0	None	Read/Write
0	1	None	Read/Write
1	0	Read-Only	Read/Write
1	1	Read/Write	Read/Write

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

Translation Look-Aside Buffer

The Am386DX/DXL microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the Am386DX/DXL device keeps a cache of the most recently accessed pages, this cache is called the Translation Look-Aside 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 Kb of memory addresses. For many common multitasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 36 illustrates how the TLB complements the Am386DX/DXL microprocessor's paging mechanism.



• 98% Hit Rate

15021B–039

Figure 36. Translation Look-Aside Buffer

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 will be placed on the address bus.

However, if the Page Table entry is not in the TLB, the Am386DX/DXL microprocessor will read the appropriate Page Directory entry. If P = 1 on the Page Directory entry indicating that the page table is in memory, then the Am386DX/DXL device will read the appropriate Page Table entry and set the Access bit. If P = 1 on the Page Table entry indicating that the page is in memory, the Am386DX/DXL device will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory entry or the Page Table Entry, then the processor will generate a Page Fault, an Exception 14.

The processor will also generate an Exception 14, Page Fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W; trying to write to a read-only page). CR2 will hold 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 will invoke the Page Fault (Exception 14) handler a second time, rather than the Double Fault (Exception 8) handler. Since Exception 14 is classified as a fault, CS:EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the Page Fault.

The 16-bit error code is used by the operating system to determine how to handle the Page Fault. Figure 37 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, R/W, and P) have similar names as the bits in the Page Directory/Table entries, the interpretation of the error code bits is different. Figure 38 indicates what type of access caused the Page Fault.

15												3	2	1	0
U	U	U	U	U	U	U	U	U	U	U	U	U	U S	W R	P

15021B–040

Figure 37. Page Fault Error Code Format

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

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

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

U: Undefined.

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

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

15021B–041

Figure 38. Type of Access Causing Page Fault

Operating System Responsibilities

The Am386DX/DXL 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 is also required to invalidate (i.e., flush) the TLB when any changes are made to any of the Page Table entries. The operating system must reload CR3 to cause the TLB to be flushed.

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

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

Virtual 8086 Environment

Executing 8086 Programs

The Am386DX/DXL 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 Am386DX/DXL device protection mechanism. In particular, the Am386DX/DXL CPU allows the simultaneous execution of 8086 operating systems and its applications, and an Am386DX/DXL CPU operating system and both 80286 and Am386DX/DXL microprocessor applications. Thus, in a multiuser Am386DX/DXL CPU computer, one person could be running a 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 they had the computer completely to themself. Figure 39 illustrates this concept.

Virtual 8086 Mode Addressing Mechanism

One of the major differences between Am386DX/DXL 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 shifted left 4 bits and added to the offset to form the segment base linear address.

The Am386DX/DXL 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-Mb address space of the Virtual Mode task can be mapped to anywhere in the 4-Gb linear address space of the Am386DX/DXL device. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kb will cause an Exception 13. However, these restrictions should not prove to be important because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

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 Mb.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4-Gb physical address space of the Am386DX/DXL microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 39 shows how the Am386DX/DXL device paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

Protection and I/O Permission Bitmap

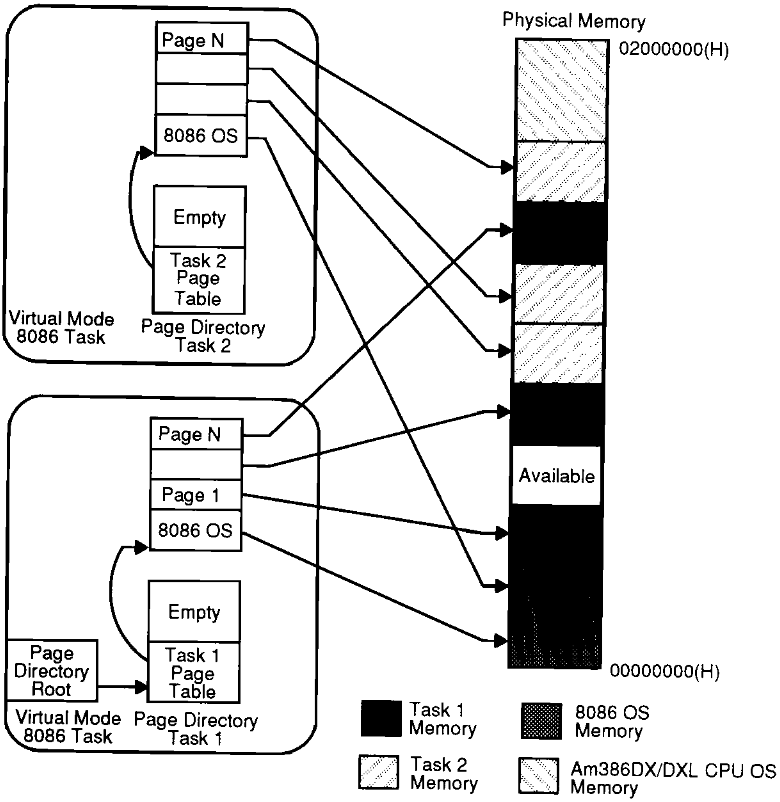
All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest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

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

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

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

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



15021B–042

Figure 39. Virtual 8086 Environment Memory Management

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 (op-code 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 Bitmap contained in the Am386DX/DXL CPU TSS. The I/O Permission Bitmap, automatically used by the Am386DX/DXL microprocessor in Virtual 8086 Mode, is illustrated by Figures 29a and 29b.

The I/O Permission Bitmap can be viewed as a 0–64K bit string, that 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 offset ≤ 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 29a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-side I/O port, as illustrated in Figure 29a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an Exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a Word-wide or Dword-wide port must be 0 for the Word-wide or Dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an Exception 13 fault.

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

Example of Bitmap for I/O Ports 0–255: Setting the TSS limit to {Bit_Map_Offset + 31 +1**} [**see note below] will allow a 32-byte bitmap for the I/O ports 0–255, plus a terminator byte of all 1s [**see note below]. This allows the I/O bitmap 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.

****Important Implementation Note**: Beyond the last byte of I/O mapping, information in the I/O Permission Bitmap must be a byte containing all 1s. The byte of all 1s must be within the limit of the Am386DX/DXL CPU TSS segment (see Figure 29a).

Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode, all interrupts and exceptions involve a privilege change back to the host Am386DX/DXL CPU operating system. The Am386DX/DXL 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 EFLAGS image on the stack.

The Am386DX/DXL microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Am386DX/DXL CPU operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Am386DX/DXL microprocessor operating system. The Am386DX/DXL CPU operating system could emulate the 8086 operating system's call. Figure 40 shows how the Am386DX/DXL microprocessor operating system could intercept an 8086 operating system's call to Open a File.

The Am386DX/DXL 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.

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 Am386DX/DXL microprocessor task whose Am386DX/DXL microprocessor TSS has an EFLAGS 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 Am386DX/DXL device 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.

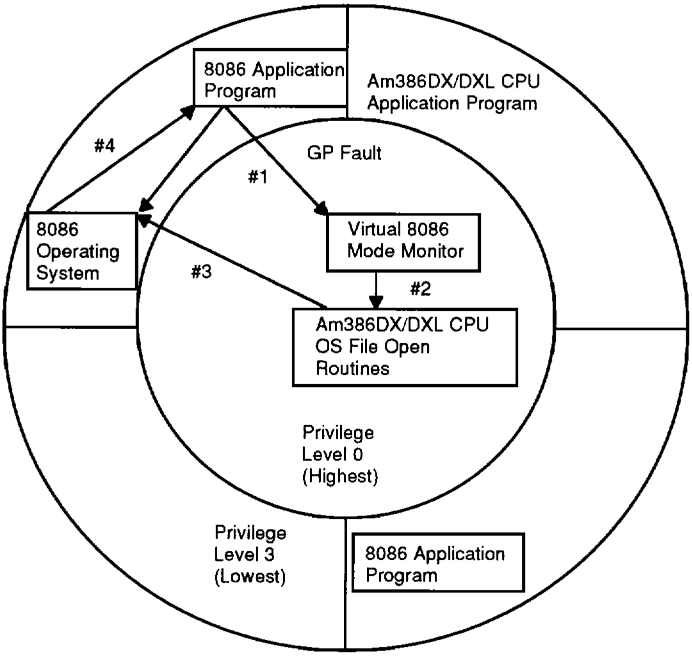
The VM bit can be set by executing an IRET instruction only at privilege level 0 or by any instruction or interrupt that causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed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

The transition out of Virtual 8086 Mode to Am386DX/DXL 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 Am386DX/DXL CPU Protected 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 interlevel interrupt only to level 0. Interrupt or Trap gates through conforming segments or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

Task Switches To/From Virtual 8086 Mode

Tasks which can execute in Virtual 8086 Mode must be described by a TSS with the new Am386DX/DXL microprocessor format (Type 9 or 11 descriptor).



8086 Application makes "Open File Call" → causes General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls Am386DX/DXL CPU OS (Arrow #2)
Am386DX/DXL CPU OS "Opens File" returns control to 8086 OS (Arrow #3)
8086 OS returns control to application (Arrow #4)
Transparent to Application

15021B–043

Figure 40. Virtual 8086 Environment Interrupt and Call Handling

A task switch out of Virtual 8086 Mode will operate exactly the same as any other task switch out of a task with an Am386DX/DXL CPU TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by an Am386DX/DXL microprocessor TSS will have an additional check to determine if the incoming task should be resumed in Virtual 8086 Mode. Tasks described by 80286 format TSSs cannot be resumed in Virtual 8086 Mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low-order 16 FLAGS bits). Before loading the segment register images from an Am386DX/DXL CPU 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.

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 exist 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 Am386DX/DXL microprocessor Trap gate (Type 14) or Interrupt gate (Type 15) that 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. Am386DX/DXL device gates must be used, since 80286 gates save only the lower 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for an Am386DX/DXL microprocessor Trap or Interrupt gate if an interrupt occurs while the task is executing in Virtual 8086 Mode is given by the following sequence.

1. Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt gate, turn off IF bit, also.
2. Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load since the VM bit was turned off above.
3. Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).
4. Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bit, 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 Am386DX/DXL Microprocessor Mode.

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

The IRET instruction will perform the inverse of the above sequence. Only the extended Am386DX/DXL microprocessor IRET instruction (operand size = 32) can be used and must be executed at level 0 to change the VM bit to 1.

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

   Otherwise, continue with the following sequence.
2. Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
3. Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped 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 will be done as an 8086 segment load.
4. Increment the ESP register by 4 to bypass the FLAGS image which was popped in step 1.
5. If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP + 20], respectively, where the new value of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

   Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.
6. If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a Protected Mode segment register load. If VM = 1, an 8086 segment register load is used.
7. Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected Mode of Virtual 8086 Mode.

FUNCTIONAL DATA

Introduction

The Am386DX/DXL microprocessor features a straight forward functional interface to the external hardware. The Am386DX/DXL CPU has separate parallel buses for data and address. The data bus is 32 bits in width and is bidirectional. The address bus outputs 32-bit address values in the most directly usable form for the high-speed local bus: 4 individual Byte Enable signals and the 30 upper-order bits as a binary value. The data and address buses are interpreted and controlled with their associated control signals.

A dynamic data bus sizing feature allows the processor to handle a mix of 32- and 16-bit external buses on a cycle-by-cycle basis (see Data Bus Sizing). If 16-bit bus size is selected, the Am386DX/DXL microprocessor automatically makes any adjustment needed, even performing another 16-bit bus cycle to complete the transfer if that is necessary. Any 8-bit peripheral devices may be connected to 32- or 16-bit buses with no loss of performance. A new address pipelining option is provided and applies to 32- and 16-bit buses for substantially improved memory utilization, especially for the most heavily used memory resources.

The address pipelining option, when selected, typically allows a given memory interface to operate with one less wait state than would otherwise be required (see Address Pipelining). The pipelined bus is also well suited to interleaved memory designs. When address pipelining is requested by the external hardware, the Am386DX/DXL microprocessor will output the address and bus cycle definition of the next bus cycle (if it is internally available) even while waiting for the current cycle to be acknowledged.

Non-pipelined address timing, however, is ideal for external cache designs, since the cache memory will typically be fast enough to allow non-pipelined cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor or from processor to system. Am386DX/DXL microprocessor bus cycles perform data transfer in a minimum of only two clock periods. On a 32-bit data bus, the maximum Am386DX/DXL device transfer at 20-MHz bandwidth is therefore 40 Mb/s, at 25-MHz bandwidth is 50 Mb/s, at 33-MHz bandwidth is 66 Mb/s, and at 40-MHz bandwidth is 80 Mb/s. Any bus cycle will be extended for more than two clock periods, however, if external hardware withholds acknowledgment of the cycle. At the appropriate time, acknowledgment is signaled by asserting the Am386DX/DXL microprocessor $\overline{\text{READY}}$ input.

The Am386DX/DXL CPU can relinquish control of its local buses to allow mastership by other devices, such as direct memory access channels. When relinquished, HLDA is the only output pin driven by the Am386DX/DXL microprocessor providing near-complete isolation of the processor from its system. The near-complete isolation characteristic is ideal when driving the system from test equipment and in fault-tolerant applications.

Functional data covered in this section describes the processor's hardware interface. First, the set of signals available at the processor pins is described (see Signal Description). Following that are the signal waveforms occurring during bus cycles (see Bus Transfer Mechanism, Bus Functional Description, and Other Functional Descriptions).

Signal Description

Introduction

Ahead is a brief description of the Am386DX/DXL CPU input and output signals arranged by functional groups (see Figure 41).

Example signal:

M/$\overline{\text{IO}}$ —High voltage indicates Memory selected
—Low voltage indicates I/O selected

The signal descriptions sometimes refer to AC timing parameters, such as t25 RESET Setup Time and t26 RESET Hold Time.

Clock (CLK2)

CLK2 provides the fundamental timing for the Am386DX/DXL microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, phase one and phase two. Each CLK2 period is a phase of the internal clock. Figure 42 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the RESET signal falling edge meets its applicable setup and hold times, t25 and t26.

Data Bus (D31–D0)

These three-state, bidirectional signals provide the general purpose data path between the Am386DX/DXL microprocessor and other devices. Data bus inputs and outputs indicate 1 when High. The data bus can transfer data on 32- and 16-bit buses using a data bus sizing feature controlled by the $\overline{\text{BS16}}$ input. See Section Bus Control. Data bus reads require that read data setup and hold times, t21 and t22, be met for correct operation. In addition, the Am386DX/DXL microprocessor requires that all data bus pins be at a valid logic state (High or Low) at the end of each read cycle, when $\overline{\text{READY}}$ is asserted. During any write operation (and during halt cycles and shut down cycles), the Am386DX/DXL microprocessor always drives all 32 signals of the data bus even if the current bus size is 16 bits.

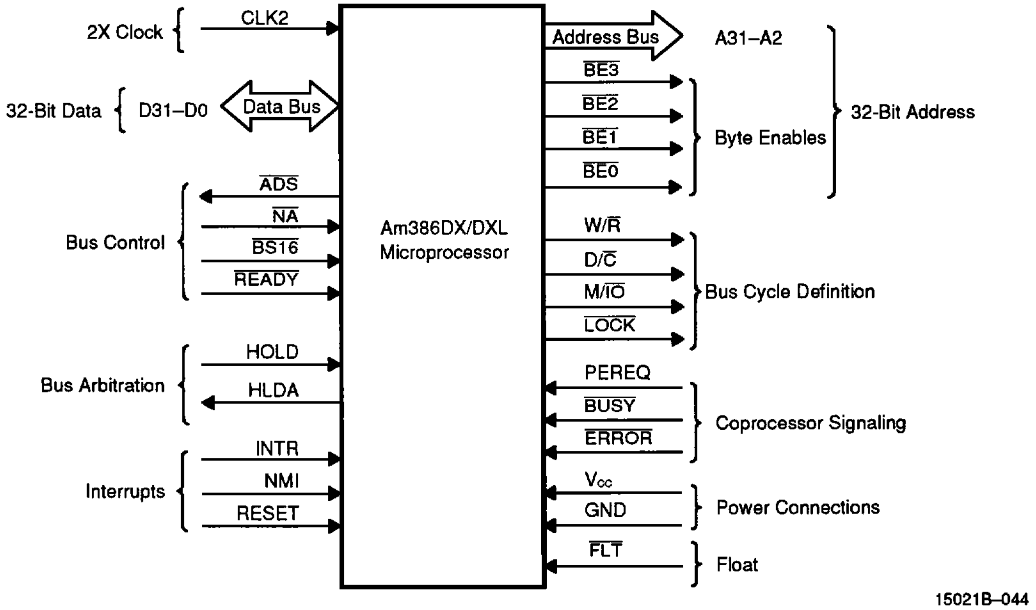


Figure 41. Functional Signal Groups

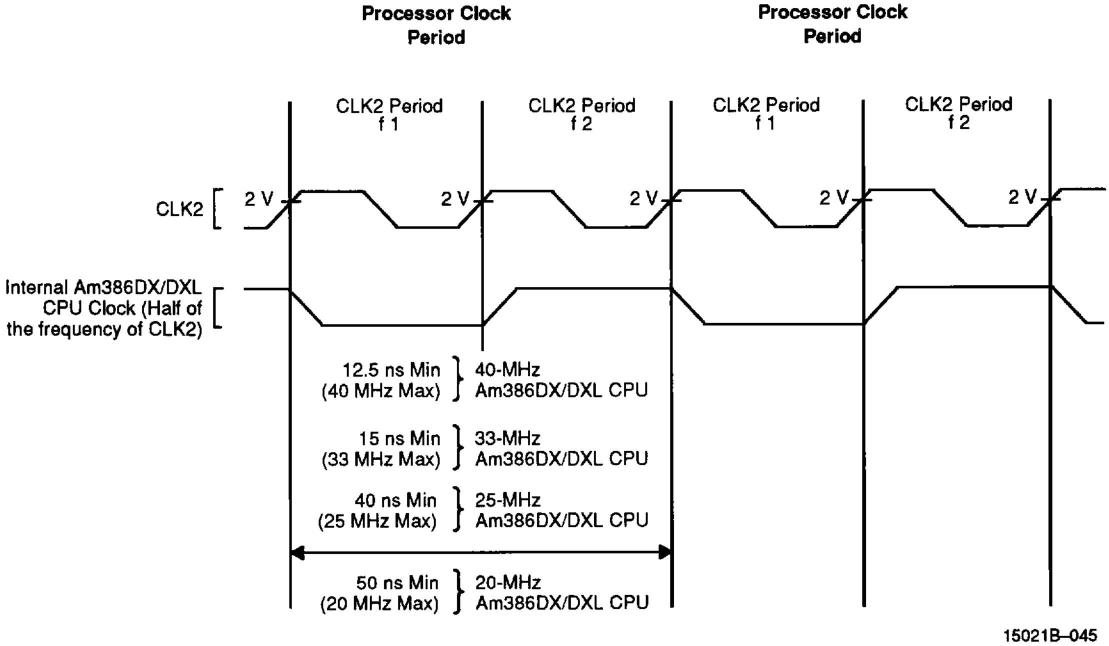


Figure 42. CLK2 Signal and Internal Processor Clock

Address Bus ($\overline{\mathrm{BE3}}$–$\overline{\mathrm{BE0}}$, A31–A2)

These three-state outputs provide physical memory addresses or I/O port addresses. The address bus is capable of addressing 4 Gb of physical memory space (00000000H–FFFFFFFFH), and 64 Kb of I/O address space (00000000H–0000FFFFH) for programmed I/O. I/O transfers automatically generated for Am386DX/DXL microprocessor-to-coprocessor communication use I/O addresses 800000F8H–800000FFH, so A31 is High in conjunction with M/$\overline{\mathrm{IO}}$ Low allows simple generation of the coprocessor select signal.

The Byte Enable outputs, $\overline{\mathrm{BE3}}$–$\overline{\mathrm{BE0}}$, directly indicate which bytes of the 32-bit data bus are involved with the current transfer. This is most convenient for external hardware.

$\overline{\mathrm{BE0}}$ applies to D7–D0
$\overline{\mathrm{BE1}}$ applies to D15–D8
$\overline{\mathrm{BE2}}$ applies to D23–D16
$\overline{\mathrm{BE3}}$ applies to D31–D24

The number of Byte Enables asserted indicates the physical size of the operand being transferred (1, 2, 3, or 4 bytes). Refer to Section Operand Alignment.

When a memory write cycle or I/O write cycle is in progress and the operand being transferred occupies only the upper 16 bits of the data bus (D31–D16), duplicate data is simultaneously presented on the corresponding lower 16 bits of the data bus (D15–D0). This duplication is performed for optimum write performance on 16 bit buses. The pattern of write data duplication is a function of the Byte Enables asserted during the write cycle. Table 13 lists the write data present on D31–D0, as a function of the asserted Byte Enable outputs $\overline{\mathrm{BE3}}$–$\overline{\mathrm{BE0}}$.

Bus Cycle Definition Signals (W/$\overline{\mathrm{R}}$, D/$\overline{\mathrm{C}}$, M/$\overline{\mathrm{IO}}$, $\overline{\mathrm{LOCK}}$)

These three-state outputs define the type of bus cycle being performed. W/$\overline{\mathrm{R}}$ distinguishes between write and read cycles, D/$\overline{\mathrm{C}}$ between data and control cycles, M/$\overline{\mathrm{IO}}$ between memory and I/O cycles, and $\overline{\mathrm{LOCK}}$ between locked and unlocked bus cycles.

The primary bus cycle definition signals are W/$\overline{\mathrm{R}}$, D/$\overline{\mathrm{C}}$, and M/$\overline{\mathrm{IO}}$, since these are the signals driven valid as the $\overline{\mathrm{ADS}}$ (Address Status output) is driven asserted. The $\overline{\mathrm{LOCK}}$ is driven valid at the same time as the first locked bus cycle begins, which due to address pipelining, could be later than $\overline{\mathrm{ADS}}$ is driven asserted. See Pipelined Address. The $\overline{\mathrm{LOCK}}$ is negated when the $\overline{\mathrm{READY}}$ input terminates the last bus cycle that was locked.

Exact bus cycle definitions, as a function of W/$\overline{\mathrm{R}}$, D/$\overline{\mathrm{C}}$, and M/$\overline{\mathrm{IO}}$, are given in Table14. Note one combination of W/$\overline{\mathrm{R}}$, D/$\overline{\mathrm{C}}$, and M/$\overline{\mathrm{IO}}$ is never given when $\overline{\mathrm{ADS}}$ is asserted (however, that combination, which is listed as does not occur, may occur during idle bus states when $\overline{\mathrm{ADS}}$ is not asserted). If M/$\overline{\mathrm{IO}}$, D/$\overline{\mathrm{C}}$, and W/$\overline{\mathrm{R}}$ are qualified by $\overline{\mathrm{ADS}}$ asserted, then a decoding scheme may be simplified by using this definition.

Table 13. Write Data Duplication as a Function of $\overline{\mathrm{BE3}}$–$\overline{\mathrm{BE0}}$

Am386DX/DXL CPU Byte Enables				Am386DX/DXL CPU Write Data				Automatic Duplication?
$\overline{\mathrm{BE3}}$	$\overline{\mathrm{BE2}}$	$\overline{\mathrm{BE1}}$	$\overline{\mathrm{BE0}}$	D31–D24	D23–D16	D15–D8	D7–D0	
High	High	High	Low	Undef	Undef	Undef	A	No
High	High	Low	High	Undef	Undef	B	Undef	No
High	Low	High	High	Undef	C	Undef	C	Yes
Low	High	High	High	D	Undef	D	Undef	Yes
High	High	Low	Low	Undef	Undef	B	A	No
High	Low	Low	High	Undef	C	B	Undef	No
Low	Low	High	High	D	C	D	C	Yes
High	Low	Low	Low	Undef	C	B	A	No
Low	Low	Low	High	D	C	B	Undef	No
Low	Low	Low	Low	D	C	B	A	No

Key: D = Logical Write Data D31–D24
C = Logical Write Data D23–D16
B = Logical Write Data D15–D8
A = Logical Write Data D7–D0

Table 14. Bus Cycle Definition

M/$\overline{\text{IO}}$	D/$\overline{\text{C}}$	W/$\overline{\text{R}}$	Bus Cycle Type	Locked?
Low	Low	Low	Interrupt Acknowledge	Yes
Low	Low	High	Does Not Occur	—
Low	High	Low	I/O Data Read	No
Low	High	High	I/O Data Write	No
High	Low	Low	Memory Code Read	No
High	Low	High	Halt: Address = 2 $\overline{\text{BE0}}$ High $\overline{\text{BE1}}$ High $\overline{\text{BE2}}$ Low $\overline{\text{BE3}}$ High A31–A2 Low Shutdown: Address = 0 $\overline{\text{BE0}}$ Low $\overline{\text{BE1}}$ High $\overline{\text{BE2}}$ High $\overline{\text{BE3}}$ High A31–A2 Low	No
High	High	Low	Memory Data Read	Some Cycles
High	High	High	Memory Data Write	Some Cycles

Bus Control Signals ($\overline{\text{ADS}}$, $\overline{\text{READY}}$, $\overline{\text{NA}}$, $\overline{\text{BS16}}$)

Introduction

The following signals allow the processor to indicate when bus cycle has begun and allow other system hardware to control address pipelining, data bus width, and bus cycle termination.

Address Status ($\overline{\text{ADS}}$)

This three-state output indicates that a valid bus cycle definition and address (W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$, $\overline{\text{BE3}}$–$\overline{\text{BE0}}$, and A31–A2) is being driven at the Am386DX/DXL microprocessor pins. It is asserted during T1 and T2P bus states (see Non-pipelined Address and Pipelined Address for additional information on bus states).

Transfer Acknowledge ($\overline{\text{READY}}$)

This input indicates the current bus cycle is complete, and the active bytes indicated by $\overline{\text{BE3}}$–$\overline{\text{BE0}}$ and $\overline{\text{BS16}}$ are accepted or provided. When $\overline{\text{READY}}$ is sampled asserted during a read cycle or interrupt acknowledge cycle, the Am386DX/DXL microprocessor latches the input data and terminates the cycle. When $\overline{\text{READY}}$ is sampled asserted during a write cycle, the processor terminates the bus cycle.

$\overline{\text{READY}}$ is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. $\overline{\text{READY}}$ must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, $\overline{\text{READY}}$ must always meet setup and hold times, t19 and t20, for correct operation. See all sections of Bus Functional Description.

Next Address Request ($\overline{\text{NA}}$)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of $\overline{\text{BE3}}$–$\overline{\text{BE0}}$, A31–A2, W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, and M/$\overline{\text{IO}}$ from the Am386DX/DXL microprocessor even if the end of the current cycle is not being acknowledged on $\overline{\text{READY}}$. If this input is asserted when sampled, the next address is driven onto the bus provided the next bus request is already pending internally. See Address Pipelining and Read and Write Cycles. $\overline{\text{NA}}$ must always meet setup and hold times, t15 and t16, for correct operation.

Bus Size 16 ($\overline{\text{BS16}}$)

The $\overline{\text{BS16}}$ feature allows the Am386DX/DXL microprocessor to directly connect to 32- and 16-bit data buses. Asserting this input constrains the current bus cycle to use only the lower-order half (D15–D0) of the data bus, corresponding to $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$. Asserting $\overline{\text{BS16}}$ has no additional effect if only $\overline{\text{BE0}}$ and/or $\overline{\text{BE1}}$ are asserted in the current cycle. However, during bus cycles asserting $\overline{\text{BE2}}$ or $\overline{\text{BE3}}$, asserting $\overline{\text{BS16}}$ will automatically cause the Am386DX/DXL microprocessor to make adjustments for correct transfer of the upper byte(s) using only physical data signals D15–D0.

If the operand spans both halves of the data bus and $\overline{\text{BS16}}$ is asserted, the Am386DX/DXL microprocessor will automatically perform another 16-bit bus cycle. $\overline{\text{BS16}}$ must always meet setup and hold times, t17 and t18, for correct operation.

Am386DX/DXL CPU I/O cycles are automatically generated for coprocessor communication. Since the Am386DX/DXL microprocessor must transfer 32-bit quantities between itself and a 387DX math coprocessor, $\overline{\text{BS16}}$ must not be asserted during 387DX math coprocessor communication cycles.

Bus Arbitration Signals (HOLD, HLDA)

Introduction

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See Entering and Exiting Hold Acknowledge for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the Am386DX/DXL CPU requires bus mastership.

HOLD must remain asserted as long as any other device is a local bus master. HOLD is not recognized while RESET is asserted. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high impedance) state. HOLD is level-sensitive and is a synchronous input. HOLD signals must always meet setup and hold times, t23 and t24, for correct operation.

Bus Hold Acknowledge (HLDA)

Assertion of this output indicates the Am386DX/DXL microprocessor has relinquished control of its local bus in response to HOLD asserted, and is in the Bus Hold Acknowledge state.

The Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the Am386DX/DXL microprocessor. The other output signals or bidirectional signals (D31–D0, BE3–BE0, A31–A2, W/R, D/C, M/IO, LOCK, and ADS) are in a high-impedance state so the requesting bus master may control them. Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See Resistor Recommendations. Also, one rising edge occurring on the NMI input during Hold Acknowledge is remembered for processing after the HOLD input is negated.

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near-complete isolation has particular attractiveness during system test when test equipment drives the system and in hardware-fault-tolerant applications.

Coprocessor Interface Signals (PEREQ, BUSY, ERROR)

Introduction

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the Am386DX/DXL microprocessor and its 387DX math coprocessor extension.

Coprocessor Request (PEREQ)

When asserted, this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the Am386DX/DXL microprocessor. In response, the Am386DX/DXL CPU transfers information between the coprocessor and memory. Because Am386DX/DXL microprocessor has internally stored the coprocessor op-code being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

Coprocessor Busy (BUSY)

When asserted, this input indicates the coprocessor is still executing an instruction and is not yet able to accept another. When the Am386DX/DXL microprocessor encounters any coprocessor instruction that operates on the numeric stack (e.g., load, pop, or arithmetic operation) or the WAIT instruction, this input is first automatically sampled until it is seen to be negated. This sampling of the BUSY input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT and FNCLEX coprocessor instructions are allowed to execute even if BUSY is asserted, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

BUSY serves an additional function. If BUSY is sampled Low at the falling edge of RESET, the Am386DX/DXL microprocessor performs an internal self-test (see Bus Activity During and Following Reset). If BUSY is sampled High, no self-test is performed.

Coprocessor Error (ERROR)

This input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the Am386DX/DXL microprocessor when a coprocessor instruction is encountered, and if asserted, the Am386DX/DXL device generates Exception 16 to access the error-handling software.

Several coprocessor instructions, generally those that clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the Am386DX/DXL microprocessor generating Exception 16 even if ERROR is asserted. These instructions are FNINIT, FNCLEX, FSTSW, FSTSWAX, FSTCW, FSTENV, FSAVE, FESTENV, and FESAVE.

ERROR is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

Interrupt Signals (INTR, NMI, RESET)

Introduction

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the Am386DX/DXL CPU Flag Register IF bit. When the Am386DX/DXL microprocessor responds to the INTR input, it performs two interrupt acknowledge bus cycles, and at the end of the second, latches an 8-bit interrupt vector on D17–D0 to identify the source of the interrupt.

INTR is level-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of an INTR request, INTR should remain asserted until the first interrupt acknowledge bus cycle begins.

Non-Maskable Interrupt Request (NMI)

This input indicates a request for interrupt service, which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is rising edge-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of NMI, it must be negated for at least eight CLK2 periods, and then be asserted for at least eight CLK2 periods.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Reset (RESET)

This input signal suspends any operation in progress and places the Am386DX/DXL microprocessor in a known reset state. The Am386DX/DXL device is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is asserted, all other input pins, except $\overline{FLT}$, are ignored, and all other bus pins are driven to an idle bus state as shown in Table 15. If RESET and HOLD are both asserted at a point in time, RESET takes priority even if the Am386DX/DXL device was in a Hold Acknowledge state prior to RESET asserted.

RESET is level-sensitive and must be synchronous to the CLK2 signal. If desired, the phase of the internal processor clock and the entire Am386DX/DXL CPU state can be completely synchronized to external circuitry by ensuring the RESET signal falling edge meets its applicable setup and hold times, t25 and t26. The signal summary is shown in Table 16.

Table 15. Pin State (Idle Bus) During Reset

Pin Name	Signal Level During Reset
$\overline{ADS}$	High
D31–D0	High Impedance
$\overline{BE3}$–$\overline{BE0}$	Low
A31–A2	High
W/$\overline{R}$	Low
D/$\overline{C}$	High
M/$\overline{IO}$	Low
$\overline{LOCK}$	High
HLDA	Low

Table 16. Am386DX/DXL Microprocessor Signal Summary

Signal Name	Function	Active State	Input/ Output	Input Synch or Asynch to CLK2	Output High Impedance During HLDA?
CLK2	Clock	—	I	—	—
D31–D0	Data Bus	High	I/O	S	Yes
$\overline{BE3}$–$\overline{BE0}$	Byte Enables	Low	O	—	Yes
A31–A2	Address Bus	High	O	—	Yes
W/$\overline{R}$	Write-Read Indication	High	O	—	Yes
D/$\overline{C}$	Data-Control Indication	High	O	—	Yes
M/$\overline{IO}$	Memory-I/O Indication	High	O	—	Yes
$\overline{LOCK}$	Bus Lock Indication	Low	O	—	Yes
$\overline{ADS}$	Address Status	Low	O	—	Yes
$\overline{NA}$	Next Address Request	Low	I	S	—
$\overline{BS16}$	Bus Size 16	Low	I	S	—
$\overline{READY}$	Transfer Acknowledge	Low	I	S	—
HOLD	Bus Hold Request	High	I	S	—
HLDA	Bus Hold Acknowledge	High	O	—	No
PEREQ	Coprocessor Request	High	I	A	—
$\overline{BUSY}$	Coprocessor Busy	Low	I	A	—
$\overline{ERROR}$	Coprocessor Error	Low	I	A	—
INTR	Maskable Interrupt Request	High	I	A	—
NMI	Non-Maskable Intrpt Request	High	I	A	—
RESET	Reset	High	I	S	—

Bus Transfer Mechanism

Introduction

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word, and Dword lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two or even three physical bus cycles are performed as required for unaligned operand transfers. See Dynamic Data Bus Sizing and Operand Alignment.

The Am386DX/DXL microprocessor address signals are designed to simplify external system hardware. Higher-order address bits are provided by A31–A2. Lower-order address in the form of $\overline{BE3}$–$\overline{BE0}$ directly provides linear selects for the four bytes of the 32-bit data bus. Physical operand size information is thereby implicitly provided for each bus cycle in the most usable form.

Byte Enable outputs, $\overline{BE3}$–$\overline{BE0}$, are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 17. During a bus cycle, any possible pattern of contiguous asserted Byte Enable outputs can occur, but never patterns having a negated Byte Enable separating two or three asserted Enables.

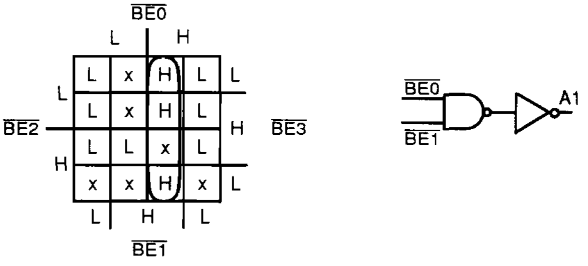
Address bits A0 and A1 of the physical operand's base address can be created when necessary (for instance, for MULTIBUS I or MULTIBUS II interface), as a function of the lowest-order asserted Byte Enable. This is shown by Table 18. Logic to generate A0 and A1 is given by Figure 43.

Table 17. Byte Enables and Associated Data and Operand Bytes

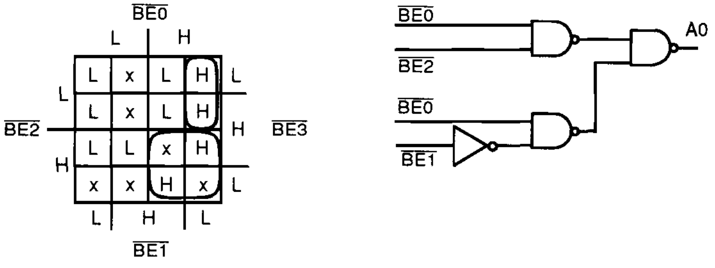
Byte Enable Signal	Associated Data Bus Signals
$\overline{BE0}$	D7–D0 (Byte 0—least significant)
$\overline{BE1}$	D15–D8 (Byte 1)
$\overline{BE2}$	D23–D16 (Byte 2)
$\overline{BE3}$	D31–D24 (Byte 3—most significant)

Each bus cycle is composed of at least two bus states and each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See Bus Functional Description.

Since a bus cycle requires a minimum of two bus states (equal to two processor clock periods), data can be transferred between external devices and the Am386DX/DXL CPU at a maximum rate of one 4-byte Dword every two processor clock periods, for a maximum bus bandwidth of 80 Mb/s (Am386DX/DXL microprocessor operating at 40-MHz processor clock rate).



K – Map for A1 Signal



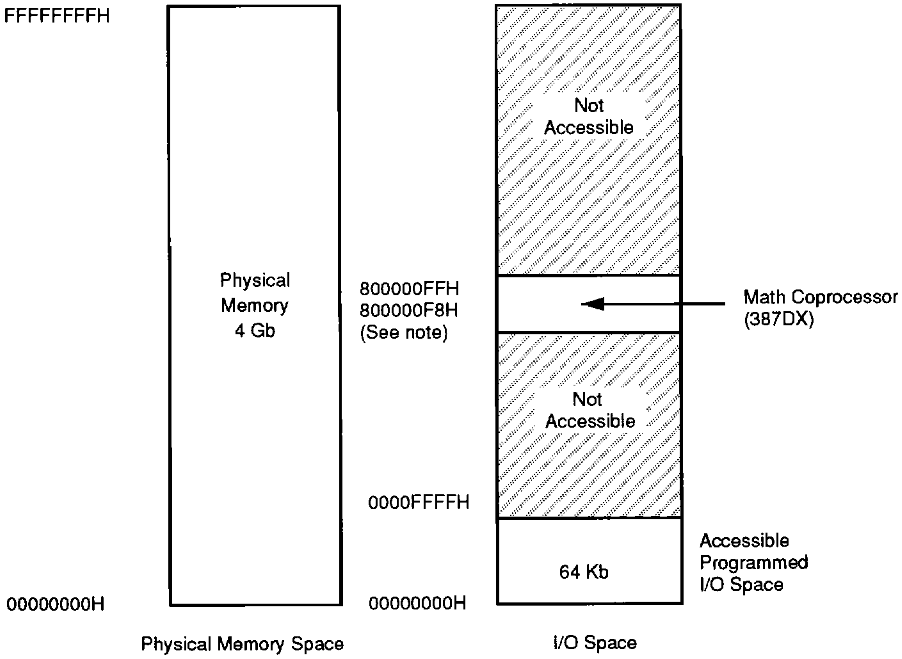
K – Map for A0 Signal

15021B–046

Figure 43. Logic to Generate A0, A1 from $\overline{BE3}$–$\overline{BE0}$

Table 18. Generating A31–A0 from $\overline{BE3}$–$\overline{BE0}$ and A31–A2

Am386DX/DXL 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



Note: Since A31 is High during automatic communication with coprocessor, A31 High and M/$\overline{IO}$ Low can be used to easily generate a coprocessor select signal.

15021B–047

Figure 44. Physical Memory and I/O Spaces

Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 44, physical memory addresses range from 00000000H to FFFFFFFFH (4 Gb) and I/O addresses from 00000000H to 0000FFFFH (64 Kb) for programmed I/O. Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 800000F8H to 800000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A31 and M/$\overline{\text{IO}}$ signals.

Memory and I/O Organization

The Am386DX/DXL microprocessor datapath to memory and I/O spaces can be 32- or 16-bits wide. When 32-bits wide, memory and I/O spaces are organized naturally as arrays of physical 32-bit Dwords. Each memory or I/O Dword has four individually addressable bytes at consecutive byte addresses. The lowest-addressed byte is associated with data signals D17–D0; the highest-addressed byte with D31–D24.

The Am386DX/DXL microprocessor includes a bus control input, $\overline{\text{BS16}}$, that also allows direct connection to 16-bit memory or I/O spaces organized as a sequence of 16-bit word. Cycles to 32- and 16-bit memory or I/O devices may occur in any sequence, since the $\overline{\text{BS16}}$ control is sampled during each bus cycle. (See Dynamic Data Bus Sizing.) The Byte Enable signals, $\overline{\text{BE3}}$–$\overline{\text{BE0}}$, allow byte granularity when addressing any memory or I/O structure, whether 32- or 16-bits wide.

Dynamic Data Bus Sizing

Dynamic Data Bus Sizing is a feature allowing direct processor connection to 32- or 16-bit data buses for memory or I/O. A single processor may connect to both size buses. Transfers to or from 32- or 16-bit ports are supported by dynamically determining the bus width during each bus cycle. During each bus cycle an address decoding circuit or the slave device itself may assert $\overline{\text{BS16}}$ for 16-bit ports, or negate $\overline{\text{BS16}}$ for 32-bit ports.

With $\overline{\text{BS16}}$ asserted, the processor automatically converts operand transfers larger than 16 bits, or misaligned 16-bit transfers, into two or three transfers as required. All operand transfers physically occur on D15–D0 when $\overline{\text{BS16}}$ is asserted. Therefore, 16-bit memories or I/O devices only connect on data signals D15–D0. No extra transceivers are required.

Asserting $\overline{\text{BS16}}$ only affects the processor when $\overline{\text{BE2}}$ and/or $\overline{\text{BE3}}$ are asserted during the current cycle. If only D15–D0 are involved with the transfer, asserting $\overline{\text{BS16}}$ has no affect since the transfer can proceed normally over a 16-bit bus whether $\overline{\text{BS16}}$ is asserted or not. In other words, asserting $\overline{\text{BS16}}$ has no effect when only the lower half of the bus is involved with the current cycle.

There are two types of situations where the processor is affected by asserting $\overline{\text{BS16}}$, depending on which Byte Enables are asserted during the current bus cycle.

Upper Half Only:

Only $\overline{\text{BE2}}$ and/or $\overline{\text{BE3}}$ asserted.

Upper and Lower Half:

At least $\overline{\text{BE1}}$, $\overline{\text{BE2}}$ asserted (and perhaps also $\overline{\text{BE0}}$ and/or $\overline{\text{BE3}}$).

Effect of asserting $\overline{\text{BS16}}$ during Upper Half Only read cycles:

Asserting $\overline{\text{BS16}}$ during Upper Half Only reads causes the Am386DX/DXL microprocessor to read data on the lower 16 bits of the data bus and ignore data on the upper 16 bits of the data bus. Data that would have been read from D31–D16 (as indicated by $\overline{\text{BE2}}$ and $\overline{\text{BE3}}$) will instead be read from D15–D0, respectively.

Effect of asserting $\overline{\text{BS16}}$ during Upper Half Only write cycles:

Asserting $\overline{\text{BS16}}$ during Upper Half Only writes does not affect the Am386DX/DXL microprocessor. When only $\overline{\text{BE2}}$ and/or $\overline{\text{BE3}}$ are asserted during a Write cycle, the Am386DX/DXL microprocessor always duplicates data signals D31–D16 onto D15–D0 (see Table 13). Therefore, no further Am386DX/DXL CPU action is required to perform these writes on 32- or 16-bit buses.

Effect of asserting $\overline{\text{BS16}}$ during Upper and Lower Half read cycles:

Asserting $\overline{\text{BS16}}$ during Upper and Lower Half reads causes the processor to perform two 16-bit read cycles for complete physical operand transfer. Bytes 0 and 1 (as indicated by $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$) are read on the first cycle using D15–D0. Bytes 2 and 3 (as indicated by $\overline{\text{BE2}}$ and $\overline{\text{BE3}}$) are read during the second cycle, again using D15–D0. D31–D16 are ignored during both 16-bit cycles. $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$ are always negated during the second 16-bit cycle. See Figure 54 Cycles 2 and 2a.

Effect of asserting $\overline{\text{BS16}}$ during Upper and Lower Half write cycles:

Asserting $\overline{\text{BS16}}$ during Upper and Lower Half writes causes the Am386DX/DXL microprocessor to perform two 16-bit write cycles for complete physical operand transfer. All bytes are available the first write cycle allowing external hardware to receive Bytes 0 and 1 (as indicated by $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$) using D15–D0. On the second cycle the Am386DX/DXL microprocessor duplicates Bytes 2 and 3 on D15–D0 and Bytes 2 and 3 (as indicated by $\overline{\text{BE2}}$ and $\overline{\text{BE3}}$) are written using D15–D0. $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$ are always negated during the second 16-bit cycle. $\overline{\text{BS16}}$ must be asserted during the second 16-bit cycle. See Figure 54 Cycles 1 and 1a.

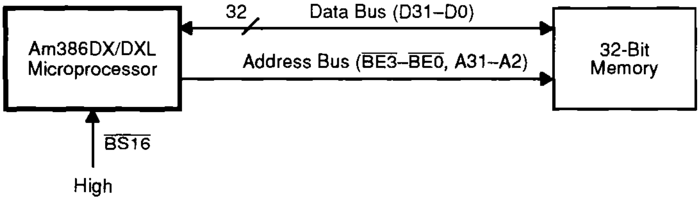
Interfacing with 32- and 16-Bit Memories

In 32-bit-wide physical memories such as Figure 45, each physical Dword begins at a byte address that is a multiple of 4. A31–A2 are directly used as a Dword selects and $\overline{\text{BE3}}$–$\overline{\text{BE0}}$ as byte selects. $\overline{\text{BS16}}$ is negated for all bus cycles involving the 32-bit array.

When 16-bit-wide physical arrays are included in the system, as in Figure 46, each 16-bit physical word begins at an address that is a multiple of 2. Note the address is decoded to assert $\overline{BS16}$ only during bus cycles involving the 16-bit array. If desiring to use pipelined address with 16-bit memories, then $\overline{BE3}$–$\overline{BE0}$ and W/$\overline{R}$ are also decoded to determine when $\overline{BS16}$ should be asserted. (See Pipelined Address with Dynamic Data Bus Sizing.)

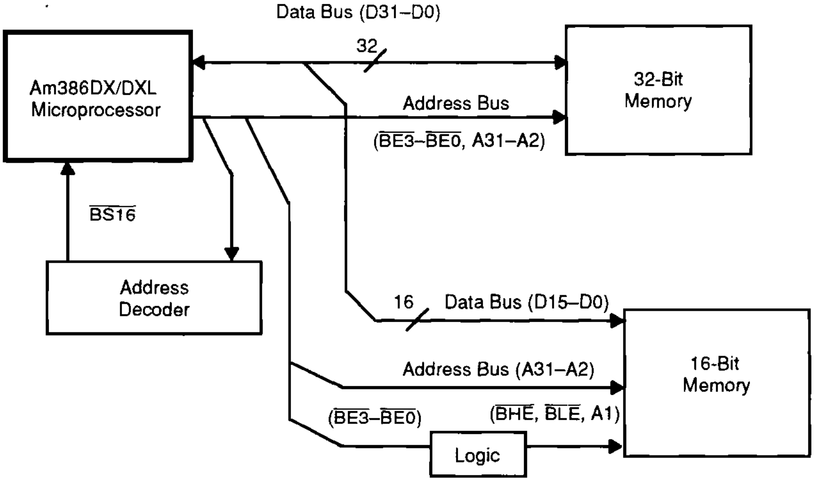
A31–A2 are directly usable for addressing 32- and 16-bit devices. To address 16-bit devices, A1 and two Byte Enable signals are also needed.

To generate an A1 signal and two Byte Enable signals for 16-bit access, $\overline{BE3}$–$\overline{BE0}$ should be decoded as in Table 19. Note that certain combinations of $\overline{BE3}$–$\overline{BE0}$ are never generated by the Am386DX/DXL microprocessor, leading to "don't care" conditions in the decoder. Any $\overline{BE3}$–$\overline{BE0}$ decoder, such as shown in Figure 47, may use the non-occurring $\overline{BE3}$–$\overline{BE0}$ combinations to its best advantage.



15021B–048

Figure 45. Am386DX/DXL Microprocessor with 32-Bit Memory



15021B–049

Figure 46. Am386DX/DXL Microprocessor with 32-Bit and 16-Bit Memory

Table 19. Generating A1, $\overline{\text{BHE}}$, and $\overline{\text{BLE}}$ for Addressing 16-Bit Devices

Am386DX/DXL CPU Signals				16-Bit Bus Signals			
$\overline{\text{BE3}}$	$\overline{\text{BE2}}$	$\overline{\text{BE1}}$	$\overline{\text{BE0}}$	A1	$\overline{\text{BHE}}$	$\overline{\text{BLE}}$ (A0)	Comments
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	

$\overline{\text{BLE}}$ asserted when D7–D0 of 16-bit bus is active.
$\overline{\text{BHE}}$ 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 are asserted or the pattern has Byte Enables asserted for non-contiguous bytes.

Operand Alignment

With the flexibility of memory addressing on the Am386DX/DXL microprocessor, it is possible to transfer a logical operand that spans more than one physical Dword or Word of memory or I/O. Examples are 32-bit Dword operands beginning at addresses not evenly divisible by 4- or a 16-bit Word operand split between two physical Dwords of memory array.

Operand alignment and data bus size dictates when multiple bus cycles are required. Table 20 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple bus cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first (but if $\overline{\text{BS16}}$ asserted requires two 16-bit cycles be performed, that part of the transfer is lowest-order first).

Bus Functional Description

Introduction

The Am386DX/DXL microprocessor has separate, parallel buses for data and address. The data bus is 32 bits in width and is bidirectional. The address bus provides a 32-bit value using 30 signals for the 30 upper-order address bits and 4 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled via several associated definition or control signals.

The definition of each bus cycle is given by three definition signals: M/$\overline{\text{IO}}$, W/$\overline{\text{R}}$, and D/$\overline{\text{C}}$. At the same time, a valid address is present on the Byte Enable signals $\overline{\text{BE3}}$–$\overline{\text{BE0}}$ and other address signals, A31–A2. A status signal, $\overline{\text{ADS}}$, indicates when the Am386DX/DXL CPU issues a new bus cycle definition and address.

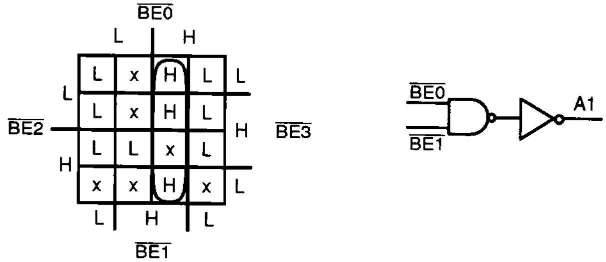
Collectively, the address bus, data bus, and all associated control signals are referred to simply as the bus.

When active, the bus performs one of the bus cycles below.

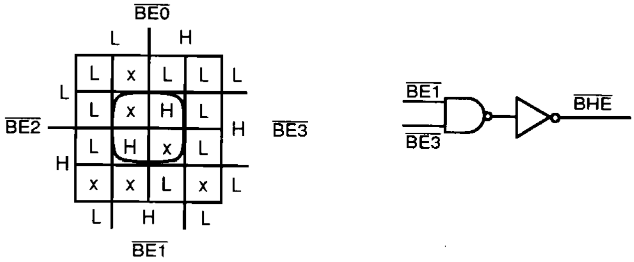
1. Read from memory space.
2. Locked read from memory space.
3. Write to memory space.
4. Locked write to memory space.
5. Read from I/O space (or coprocessor).
6. Write to I/O space (or coprocessor).
7. Interrupt acknowledge.
8. Indicate halt or indicate shutdown.

Table 14 shows the encoding of the bus cycle definition signals for each bus cycle. See Section Bus Cycle Definition.

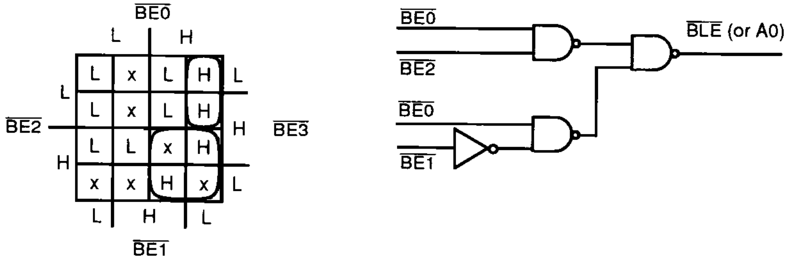
The data bus has a dynamic sizing feature supporting 32- and 16-bit bus size. Data bus size is indicated to the Am386DX/DXL microprocessor using its Bus Size 16 ($\overline{\text{BS16}}$) input. All bus functions can be performed with either data bus size.



K – Map for A1 Signal (same as Figure 43)



K – Map for 16-bit $\overline{\text{BHE}}$ signal



K – Map for 16-bit $\overline{\text{BLE}}$ signal (same as A0 signal in Figure 43).

15021B–050

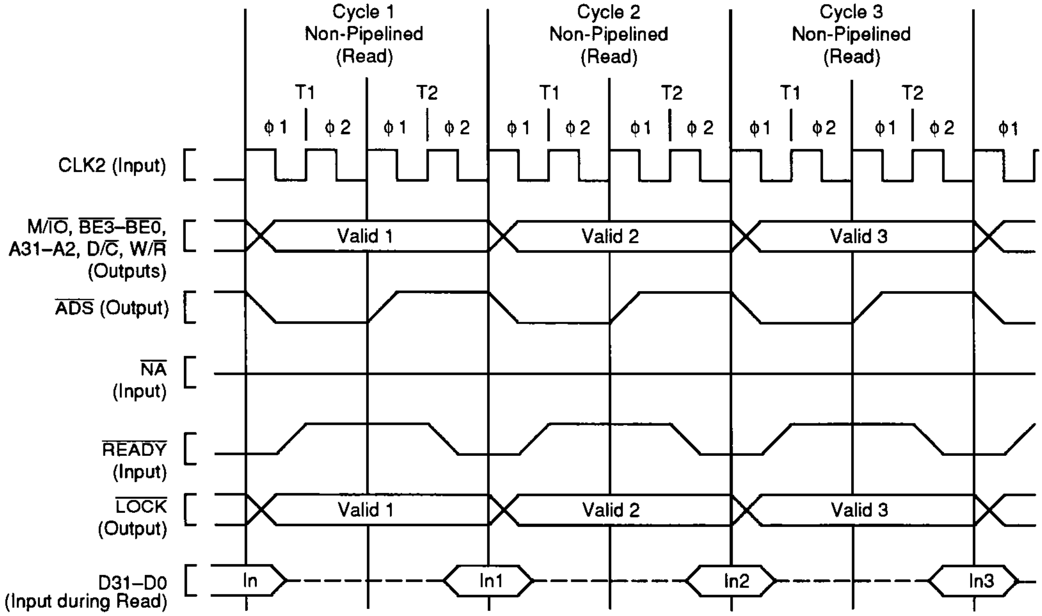
Figure 47. Logic to Generate A1, $\overline{\text{BHE}}$, and $\overline{\text{BLE}}$ for 16-Bit Buses

Table 20. 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 Data Bus	b	w	w	w	hb,* lb	d	hd l3	hw, lw	h3, lb
Transfer Cycles over 16-bit Data Bus	b	w	lb, hb	w	hb, lb	lw, hw	hb, lb, mw	hw, lw	mw, hb, lb

Key:
b = Byte transfer
w = Word transfer
l = low-order portion
m = mid-order portion
3 = 3-byte transfer
d = Dword transfer
h = high-order portion
x = Don't care
= $\overline{\text{BS16}}$ asserted causes second bus cycle.

*For this case, 8086, 8088, 80186, 80188, 80286 transfer lb first, then hb.



Fastest non-pipelined bus cycles consist of T1 and T2

15021B–051

Figure 48. Fastest Read Cycles with Non-Pipelined Address Timing

When the Am386DX/DXL CPU bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The Idle state can be identified by the Am386DX/DXL microprocessor giving no further assertions on its address strobe output (ADS) since the beginning of its most recent bus cycle, and the most recent bus cycle has been terminated. The Hold Acknowledge state is identified by the Am386DX/DXL CPU asserting its Hold Acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

The fastest Am386DX/DXL microprocessor bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 48. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough. The high-bandwidth, two-clock bus cycle realizes the full potential of fast main memory, or cache memory.

Every bus cycle continues until it is acknowledged by the external system hardware, using the Am386DX/DXL microprocessor READY input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If READY is not immediately asserted, however, T2 states are repeated indefinitely until the READY input is sampled asserted.

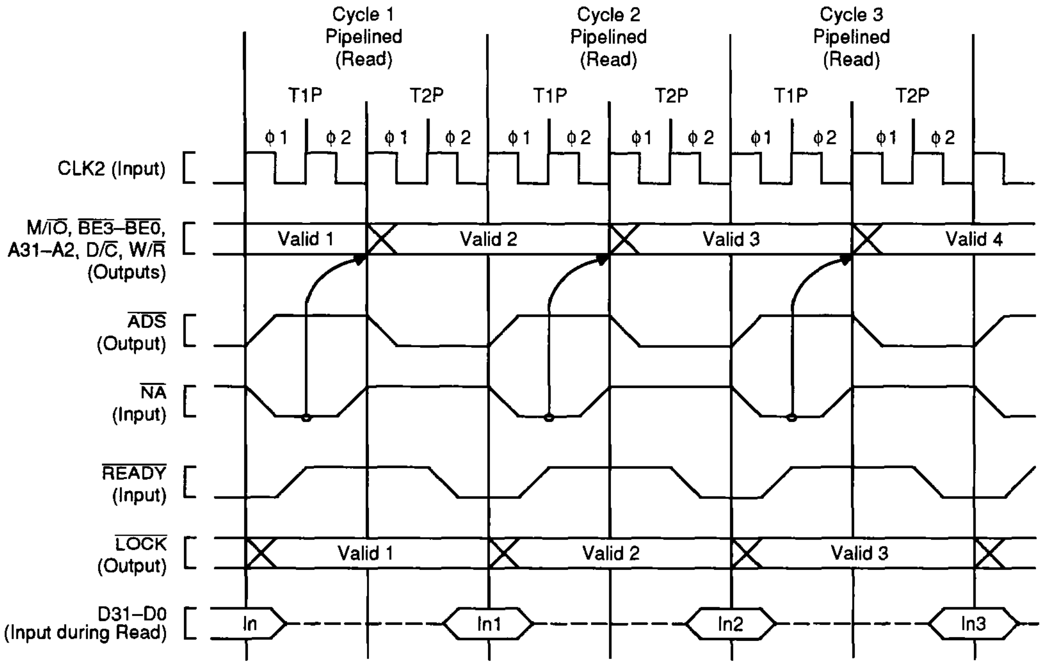
Address Pipelining

The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address (NA) input.

When address pipelining is not selected, the current address and bus cycle definition remain stable throughout the bus cycle.

When address pipelining is selected, the address (BE3–BE0, A31–A2) and definition (W/R, D/C, and M/IO) of the next cycle are available before the end of the current cycle. To signal their availability, the Am386DX/DXL microprocessor address status output (ADS) is also asserted. Figure 49 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 49, the fastest bus cycles using pipelined address require only two bus states, named T1P and T2P. Therefore, cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased compared to that of a non-pipelined cycle.



Fastest pipelined bus cycles consist of T1P and T2P

15021B–052

Figure 49. Fastest Read Cycles with Pipelined Address Timing

By increasing the address-to-data access time, pipelined address timing reduces wait state requirements. For example, if one wait state is required with non-pipelined address timing, no wait states would be required with pipelined address.

Pipelined address timing is useful in typical systems having address latches. In those systems, once an address has been latched, pipelined availability of the next address allows decoding circuitry to generate chip selects (and other necessary select signals) in advance, so selected devices are accessed immediately when the next cycle begins. In other words, the decode time for the next cycle can be overlapped with the end of the current cycle.

If a system contains a memory structure of two or more interleaved memory banks, pipelined address timing potentially allows even more overlap of activity. This is true when the interleaved memory controller is designed to allow the next memory operation to begin in one memory bank while the current bus cycle is still activating another memory bank. Figure 50 shows the general structure of the Am386DX/DXL microprocessor with two-bank and four-bank interleaved memory. Note each memory bank of the interleaved memory has full data bus width (32-bit data width typically, unless 16-bit bus size is selected).

Further details of pipelined address timing are given in Pipelined Address; Initiating and Maintaining Pipelined Address; Pipelined Address with Dynamic Bus Sizing; and, Maximum Pipelined Address Usage With 16-bit Bus Size.

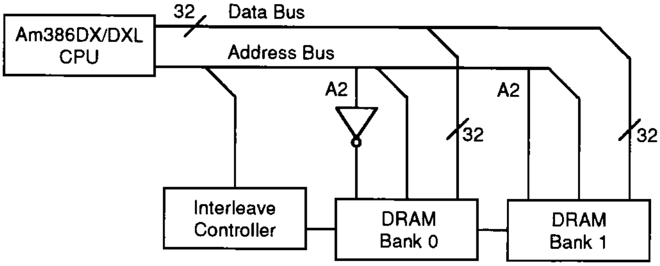
Read and Write Cycles

Introduction

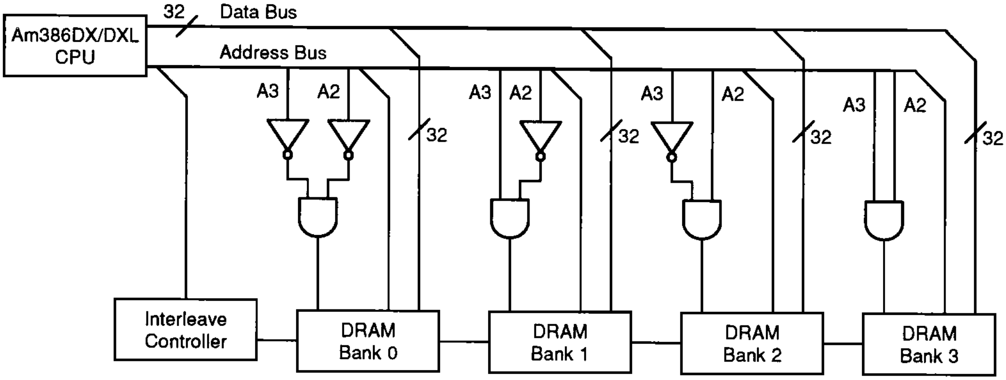
Data transfers occur as a result of bus cycles, classified as Read or Write cycles. During Read cycles, data is transferred from an external device to the processor. During Write cycles, data is transferred in the other direction, from the processor to an external device.

Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After a bus idle state, the processor always uses non-pipelined address timing. However, the NA (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the Am386DX/DXL microprocessor has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY. Generally, the NA input is sampled each bus cycle to select the desired address timing for the next bus cycle.

Two-Bank Interleaved Memory:
a. Address signal A2 selects bank
b. 32-bit datapath to each bank



Four-Bank Interleaved Memory:
a. Address signals A3 and A2 select bank
b. 32-bit datapath to each bank



15021B–053

Figure 50. Two-Bank and Four-Bank Interleaved Memory Structure

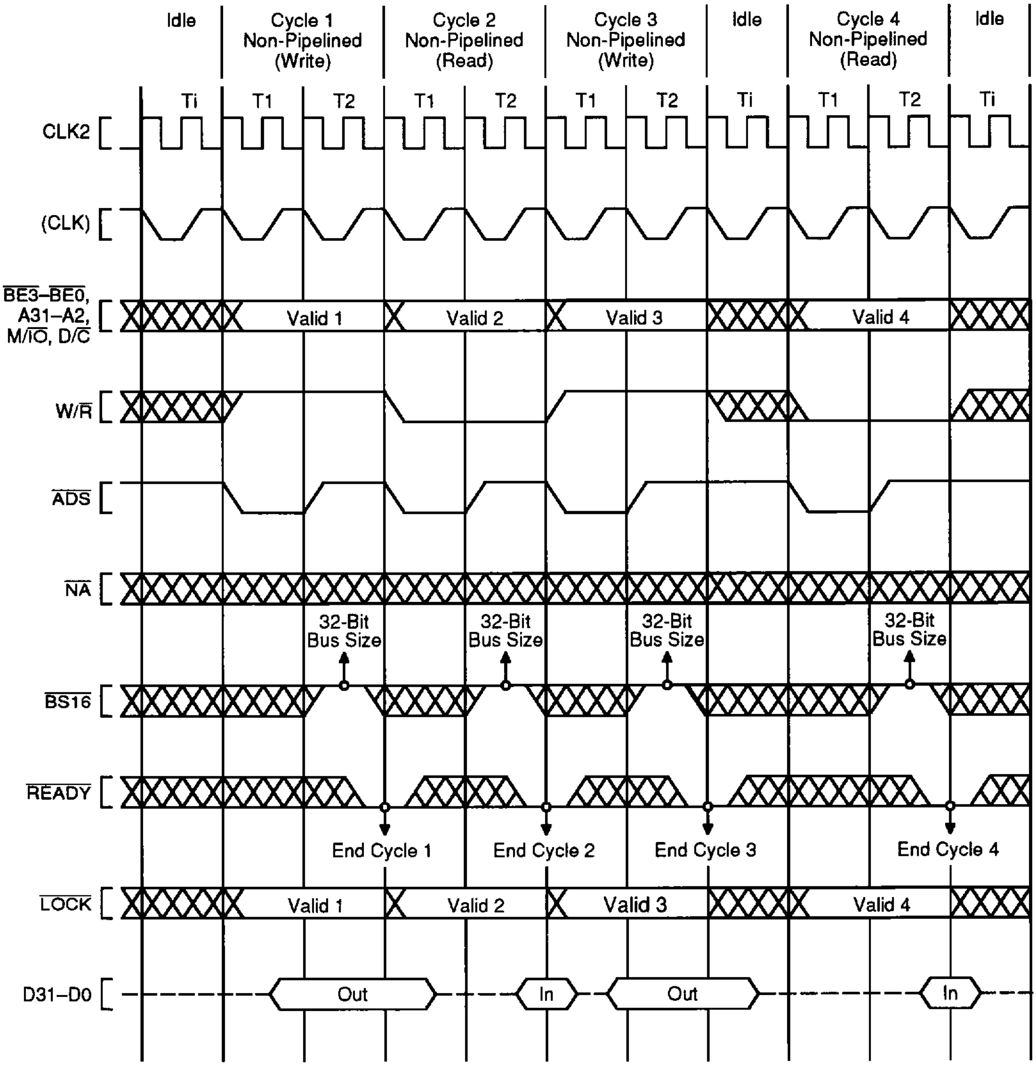
Two choices of physical data bus width are dynamically selectable: 32 bits or 16 bits. Generally, the $\overline{\text{BS16}}$ (Bus Size 16) input is sampled near the end of the bus cycle to confirm the physical data bus size applicable to the current cycle. Negation of $\overline{\text{BS16}}$ indicates a 32-bit size and assertion indicates a 16-bit bus size.

If 16-bit bus size is indicated, the Am386DX/DXL CPU automatically responds as required to complete the transfer on a 16-bit data bus. Depending on the size and alignment of the operand, another 16-bit bus cycle may be required. Table 19 provides all details. When necessary, the Am386DX/DXL microprocessor performs an additional 16-bit bus cycle, using D15–D0 in place of D31–D16.

Terminating a Read cycle or Write cycle, like any bus cycle, requires acknowledging the cycle by asserting the $\overline{\text{READY}}$ input. Until acknowledged, the processor inserts wait states into the bus cycle to allow adjustment for the speed of any external device. External hardware, that has decoded the address and bus cycle type asserts the $\overline{\text{READY}}$ input at the appropriate time.

At the end of the second bus state within the bus cycle, $\overline{\text{READY}}$ is sampled. At that time, if external hardware acknowledges the bus cycle by asserting $\overline{\text{READY}}$, the bus cycle terminates as shown in Figure 51. If $\overline{\text{READY}}$ is negated as in Figure 52, the cycle continues another bus state (a wait state) and $\overline{\text{READY}}$ is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by $\overline{\text{READY}}$ asserted.

When the current cycle is acknowledged, the Am386DX/DXL microprocessor terminates it. When a Read cycle is acknowledged, the Am386DX/DXL CPU latches the information present at its data pins. When a Write cycle is acknowledged, the Am386DX/DXL CPU write data remains valid throughout phase one of the next bus state to provide write data hold time.



Note: Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the write cycle.

15021B–054

Figure 51. Various Bus Cycles and Idle States with Non-Pipelined Address (Zero Wait States)

Non-Pipelined Address

Any bus cycle may be performed with non-pipelined address timing. For example, Figure 51 shows a mixture of Read and Write cycles with non-pipelined address timing. Figure 51 shows that the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of the T1, the address signals and bus cycle definition signals are driven valid, and to signal their availability, address status (ADS) is simultaneously asserted.

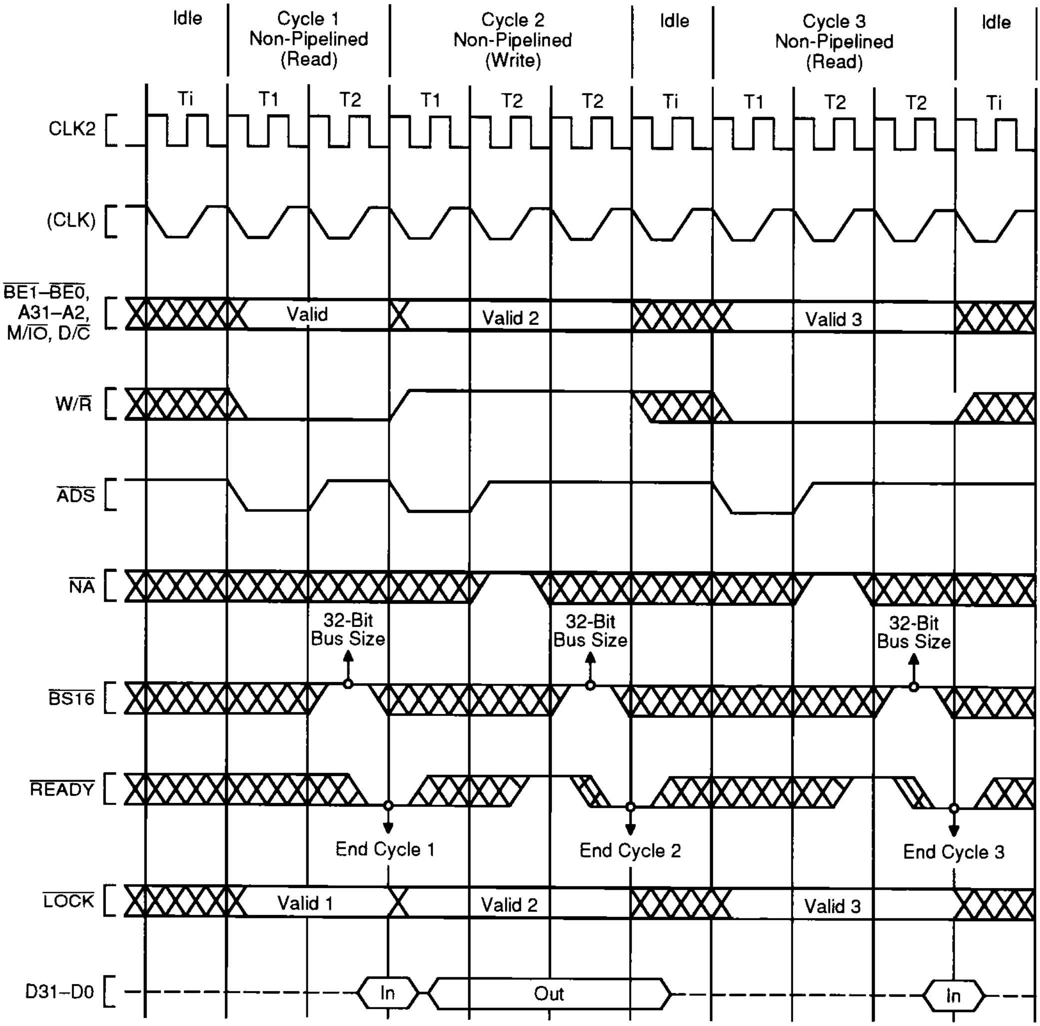
During Read or Write cycles, the data bus behaves as follows. If the cycle is a read, the Am386DX/DXL microprocessor floats its data signals to allow driving by the external device being addressed. The Am386DX/DXL device requires that all data bus pins be at a valid logic state (High or Low) at the end of each read cycle, when READY is asserted, even if all byte enables are not asserted. The system **must** be designed to meet this requirement. If the cycle is a write, data signals are driven by the Am386DX/DXL device beginning in phase two of

T1 until phase one of the bus state following cycle acknowledgment.

Figure 52 illustrates non-pipelined bus cycles with one wait added to Cycles 2 and 3. $\overline{\text{READY}}$ is sampled negated at the end of the first T2 in Cycles 2 and 3. Therefore, Cycles 2 and 3 have T2 repeated. At the end of the second T2, $\overline{\text{READY}}$ is sampled asserted.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states.

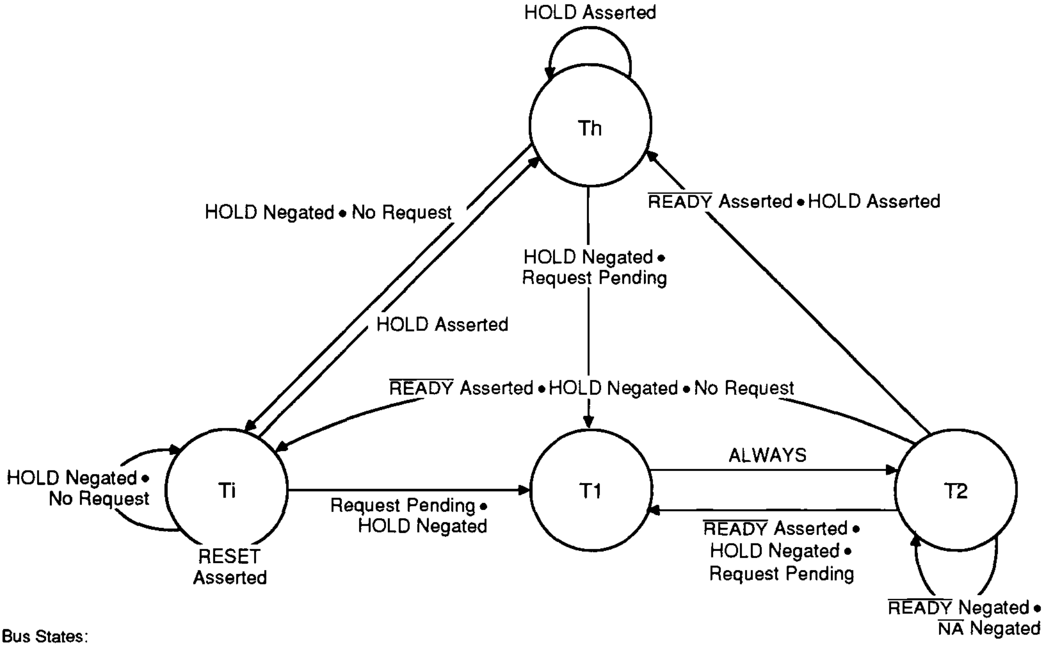
When wait states are added and you desire to maintain non-pipelined address timing, it is necessary to negate $\overline{\text{NA}}$ during each T2 state except the last one, as shown in Figure 52 Cycles 2 and 3. If $\overline{\text{NA}}$ is sampled asserted during a T2 other than the last one, the next state would be T2I (for pipelined address) or T2P (for pipelined address) instead of another T2 (for non-pipelined address).



Note: Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the Write cycle.

15021B–055

Figure 52. Various Bus Cycles and Idle States with Non-Pipelined Address (Various Number of Wait States)



Bus States:
T1— First clock of a non-pipelined bus cycle (Am386DX/DXL microprocessor drives new address and asserts ADS).
T2— Subsequent clocks of a bus cycle when NA has not been sampled asserted in the current bus cycle.
Ti — Idle state.
Th— Hold Acknowledge state (Am386DX/DXL microprocessor asserts HLDA).

15022B–017

The fastest bus cycle consists of two states: T1 and T2.

Four basic bus states describe bus operation when not using pipelined address. These states do include BS16 usage for 32-bit and 16-bit bus size. If asserting BS16 requires second 16-bit bus cycle to be performed, it is performed before HOLD asserted acknowledged.

Figure 53. Bus States (Not Using Pipelined Address)

Figure 53 illustrates the bus states and transitions when address pipelining is not used. The bus transitions between four possible states: T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise, the bus may be idle in the Ti state, or in hold acknowledge, the Th state.

When address pipelining is not used, the bus state diagram is as shown in Figure 53. When the bus is idle, it is in state Ti. Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA is negated, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or Ti if there is no bus request pending, or Th if the HOLD input is being asserted.

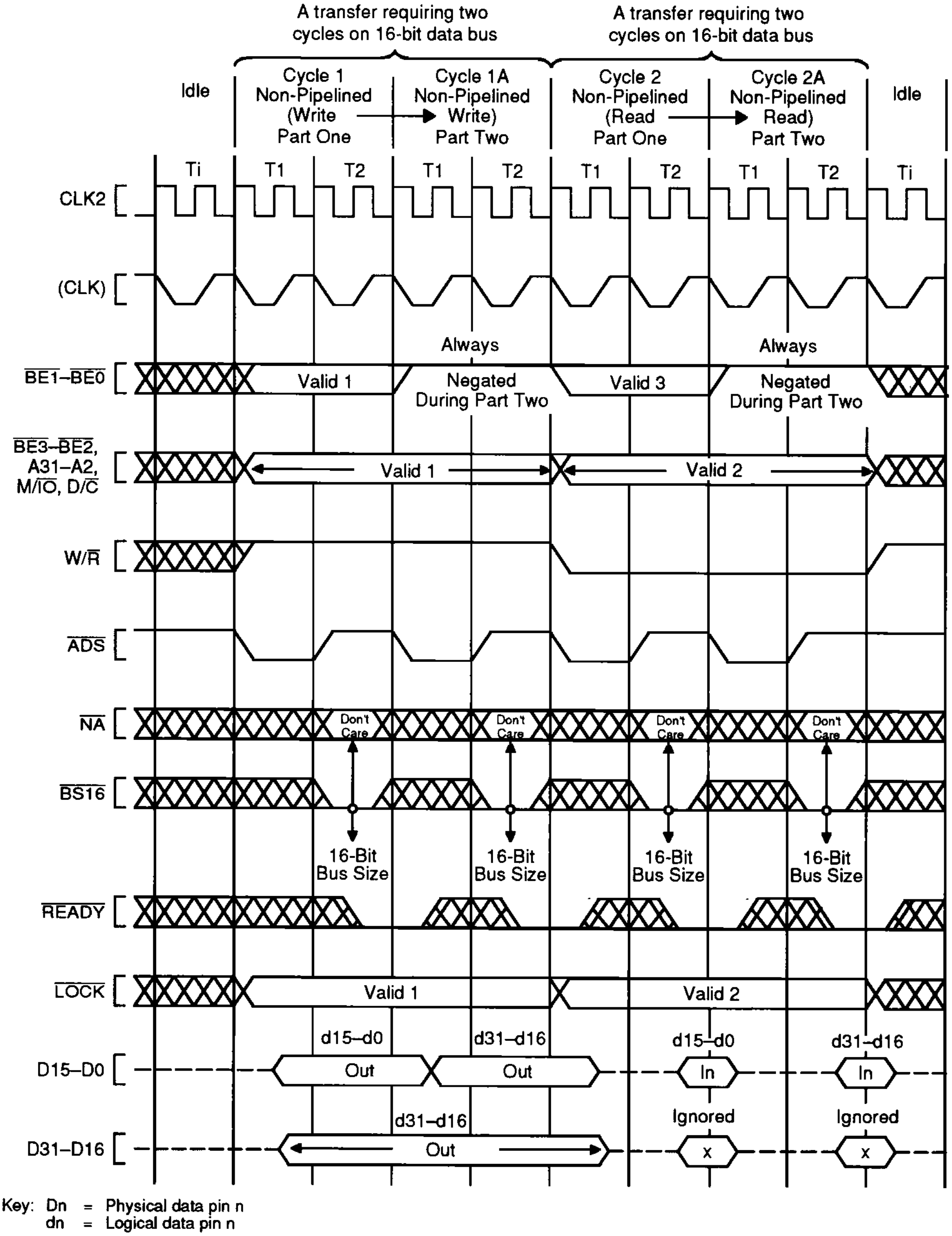
The bus state diagram in Figure 53 also applies to the use of BS16. If the Am386DX/DXL microprocessor makes internal adjustments for 16-bit bus size, the adjustments do not affect the external bus states. If an additional 16-bit bus cycle is required to complete a transfer on a 16-bit bus, it also follows the state transitions shown in Figure 53.

Use of pipelined address allows the Am386DX/DXL CPU to enter three additional bus states not shown in Figure 53. Figure 59 in Pipelined Address is the complete bus state diagram, including pipelined address cycles.

Non-Pipelined Address With Dynamic Data Bus Sizing

The physical data bus width for any non-pipelined bus cycle can be either 32 or 16 bits. At the beginning of the bus cycle, the processor behaves as if the data bus is 32-bits wide. When the bus cycle is acknowledged by asserting READY at the end of a T2 state, the most recent sampling of BS16 determines the data bus size for the cycle being acknowledged. If BS16 was most recently negated, the physical data bus size is defined as 32 bits. If BS16 was most recently asserted, the size is defined as 16 bits.

When BS16 is asserted and two 16-bit bus cycles are required to complete the transfer, BS16 must be asserted during the second cycle; 16-bit bus size is not assumed. Like any bus cycle, the second 16-bit cycle must be acknowledged by asserting READY.



15021B–057

Figure 54. Asserting BS16 (Zero-Wait-States, Non-Pipelined Address)

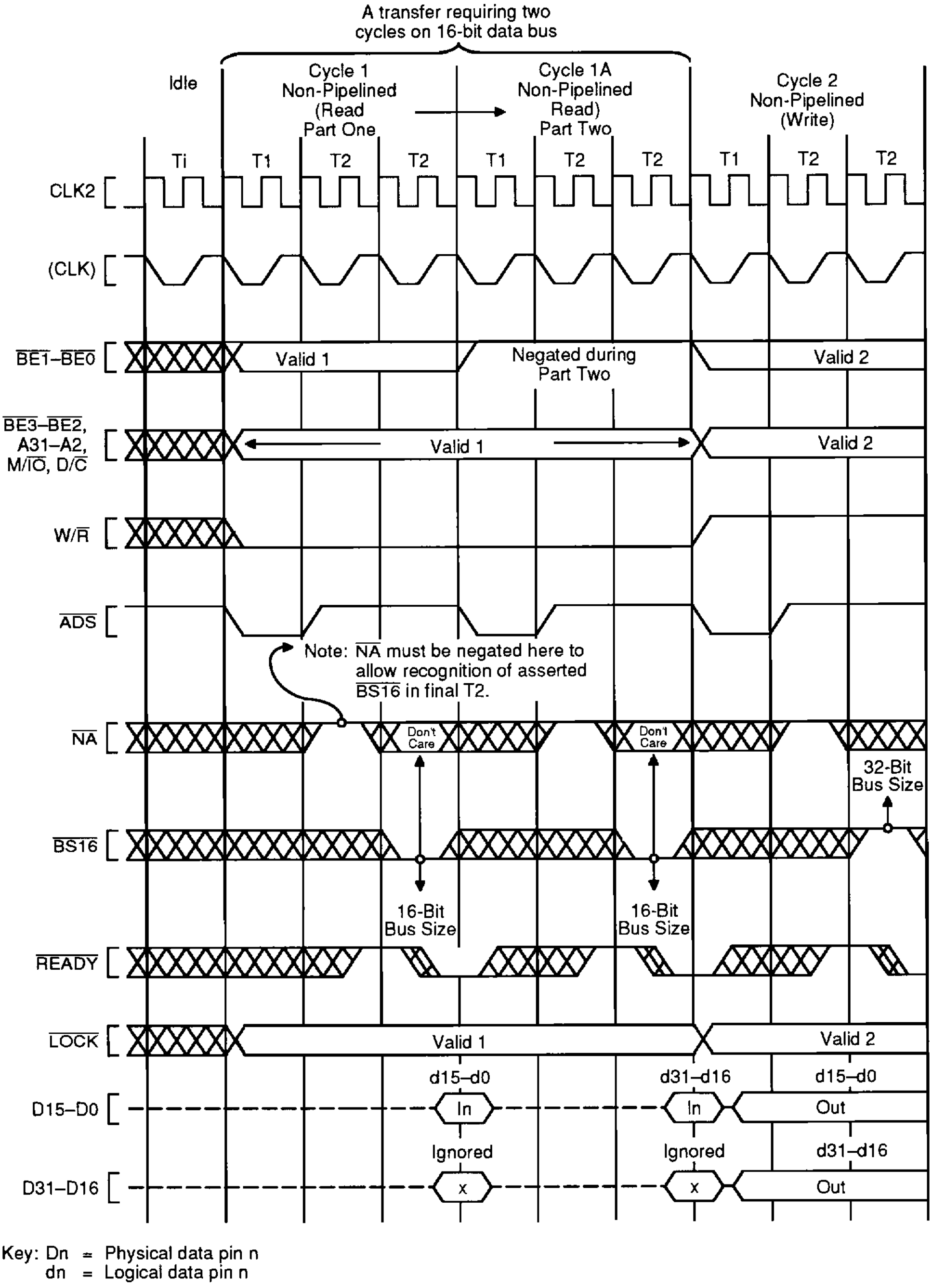


Figure 55. Asserting $\overline{BS16}$ (One-Wait-State, Non-Pipelined Address)

When a second 16-bit bus cycle is required to complete the transfer over a 16-bit bus, the addresses generated for the two 16-bit bus cycles are closely related to each other. The addresses are the same, except $\overline{\text{BE0}}$ and $\overline{\text{BE1}}$ are always negated for the second cycle. This is because data on D15–D0 was already transferred during the first 16-bit cycle.

Figures 54 and 55 show cases where assertion of $\overline{\text{BS16}}$ requires a second 16-bit cycle for complete operand transfer. Figure 54 illustrates cycles without wait states. Figure 55 illustrates cycles with one wait state. In Figure 55 Cycle 1, the bus cycle during which $\overline{\text{BS16}}$ is asserted, note that $\overline{\text{NA}}$ must be negated in the T2 state(s) prior to the last T2 state. This is to allow the recognition of $\overline{\text{BS16}}$ asserted in the final T2 state. The relation of $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ is given fully in Pipelined Address, but Figure 55 illustrates this only precaution you need to know when using $\overline{\text{BS16}}$ with non-pipelined address.

Pipelined Address

Address pipelining is the option of requesting the address and the bus cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with $\overline{\text{READY}}$ asserted. $\overline{\text{ADS}}$ is asserted by the Am386DX/DXL microprocessor when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the $\overline{\text{NA}}$ input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the $\overline{\text{NA}}$ input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles, therefore, $\overline{\text{NA}}$ is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 56, during which $\overline{\text{NA}}$ is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).

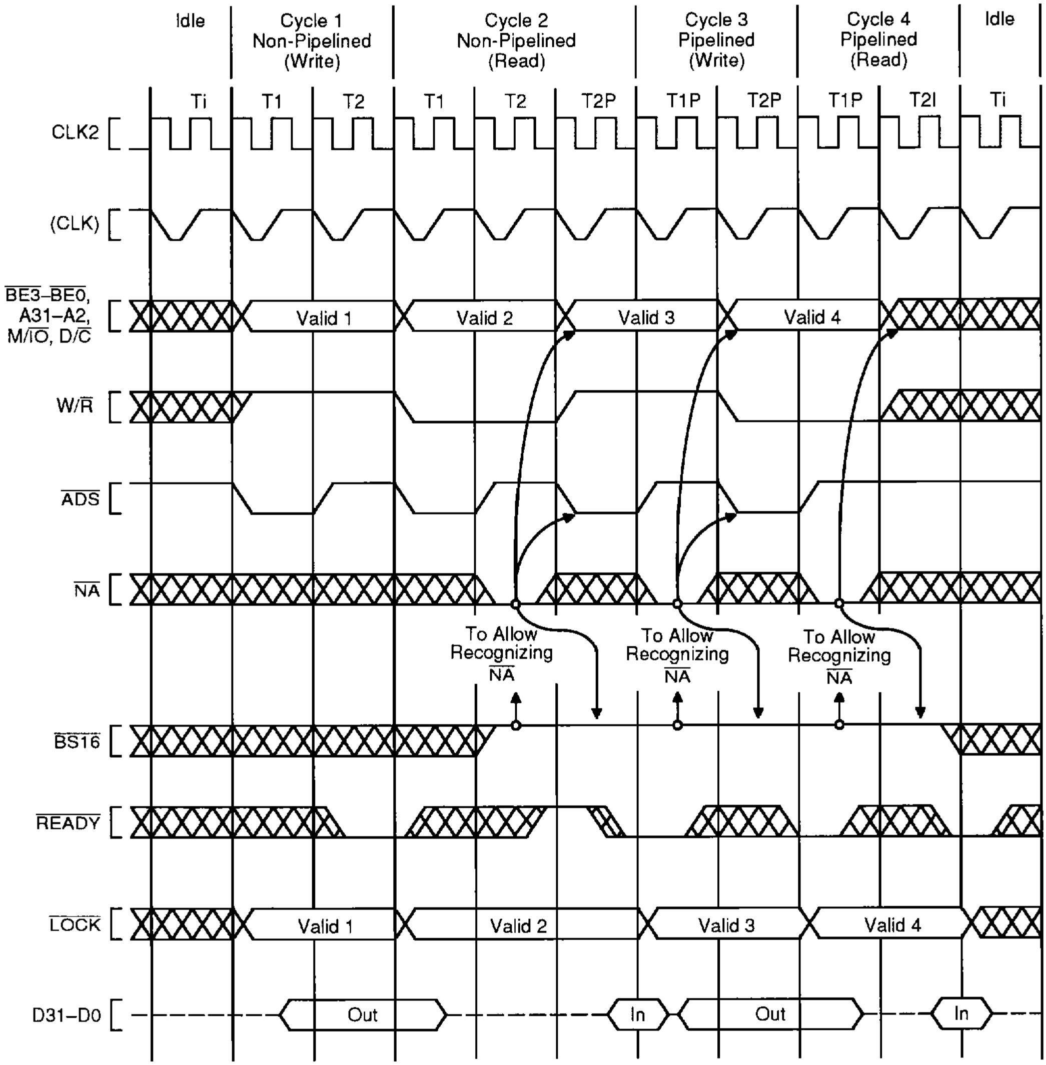
If $\overline{\text{NA}}$ is sampled asserted, the Am386DX/DXL microprocessor is free to drive the address and bus cycle definition of the next bus cycle, and assert $\overline{\text{ADS}}$, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the Am386DX/DXL CPU has the following characteristics.

1. For $\overline{\text{NA}}$ to be sampled asserted, $\overline{\text{BS16}}$ must be negated at the sampling window (see Figure 56 Cycles 2 through 4, and Figure 57 Cycles 1 through 4). If $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ are both sampled asserted during the last T2 period of a bus cycle, $\overline{\text{BS16}}$ asserted has priority. Therefore, if both are asserted, the current bus size is taken to be 16 bits and the next address is not pipelined.
2. The next address may appear as early as the bus state after $\overline{\text{NA}}$ was sampled asserted (see Figure 56 or 57). In that case, state T2P is entered immediately, However, when there is not an internal bus request already pending, the next address will not be available immediately after $\overline{\text{NA}}$ is asserted and T2I is entered instead of T2P (see Figure 58 Cycle 3). Provided the current bus cycle is not yet acknowledged by $\overline{\text{READY}}$ asserted, T2P will be entered as soon as the Am386DX/DXL microprocessor does drive the next address. External hardware should therefore observe the $\overline{\text{ADS}}$ output as confirmation the next address is actually being driven on the bus.
3. Once $\overline{\text{NA}}$ is sampled asserted, the Am386DX/DXL microprocessor commits itself to the highest priority bus request that is pending internally. It can no longer perform another 16-bit transfer to the same address should $\overline{\text{BS16}}$ be asserted externally, so thereafter must assume the current bus size is 32 bits. Therefore, if $\overline{\text{NA}}$ is sampled asserted within a bus cycle, $\overline{\text{BS16}}$ must be negated thereafter in that bus cycle (see Figures 56, 57, 58). Consequently, do not assert $\overline{\text{NA}}$ during bus cycles that must have $\overline{\text{BS16}}$ driven asserted. See Dynamic Bus Sizing with Pipelined Address.
4. Any address which is validated by a pulse on the Am386DX/DXL CPU $\overline{\text{ADS}}$ output will remain stable on the address pins for at least two processor clock periods. The Am386DX/DXL microprocessor cannot produce a new address more frequently than every two processor clock periods (see Figures 56, 57, 58).
5. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 58 Cycle 1).

The complete bus state transition diagram, including operation with pipelined address is given by Figure 59. Note it is a superset of the diagram for non-pipelined address only and the three additional bus states for pipelined address are drawn in bold.

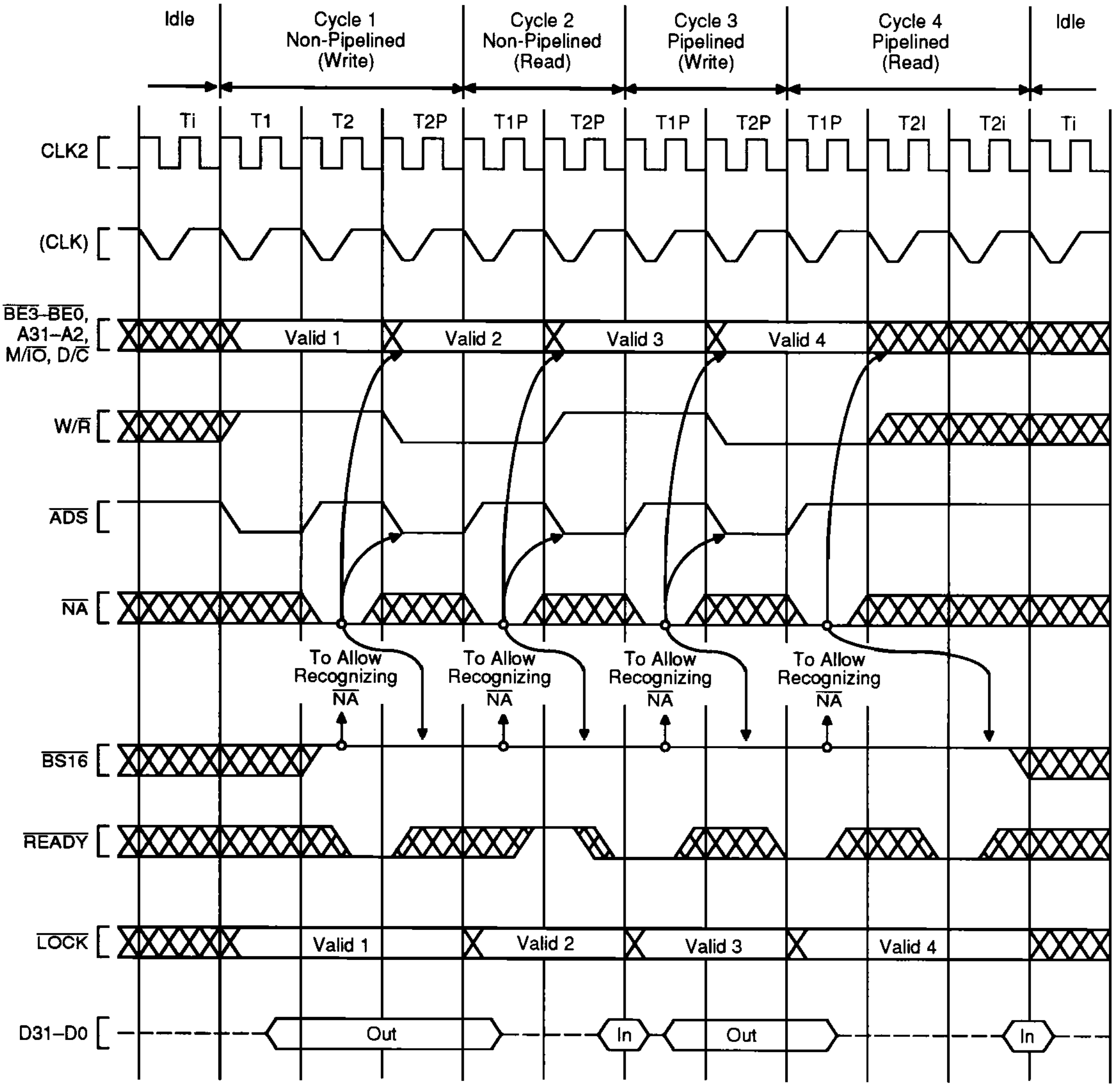
The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.



Note: Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, NA is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

15021B–059

Figure 56. Transitioning to Pipelined Address During Burst of Bus Cycles



Note: Following any idle bus state (Ti), the address is always non-pipelined and NA is only sampled during wait states. To start address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (Cycle 1 above). The pipelined cycles (2, 3, 4 above) are shown with various numbers of wait states.

15021B–060

Figure 57. Fastest Transition to Pipelined Address Following Idle Bus State

Initiating and Maintaining Pipelined Address

Using the state diagram Figure 59, observe the transitions from an idle state, Ti, to the beginning of a pipelined bus cycle, T1P. From an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided $\overline{NA}$ is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in below:

$$\underbrace{\text{Ti, Ti, Ti}}_{\substack{\text{Idle}\\ \text{States}}} \quad \underbrace{\text{T1-T2-T2P}}_{\substack{\text{Non-Pipelined}\\ \text{Cycle}}} \quad \underbrace{\text{T1P-T2P}}_{\substack{\text{Pipelined}\\ \text{Cycle}}}$$

T1-T2-T2P are the states of the bus cycle that establishes address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

$$\underbrace{\text{Th, Th, Th}}_{\substack{\text{Hold}\\ \text{Acknowledge}\\ \text{States}}} \quad \underbrace{\text{T1-T2-T2P}}_{\substack{\text{Non-Pipelined}\\ \text{Cycle}}} \quad \underbrace{\text{T1P-T2P}}_{\substack{\text{Pipelined}\\ \text{Cycle}}}$$

The transition to pipelined address is shown functionally by Figure 57 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3, and 4 that are pipelined. The $\overline{NA}$ input is asserted at the appropriate time to select address pipelining for Cycles 2, 3, and 4.

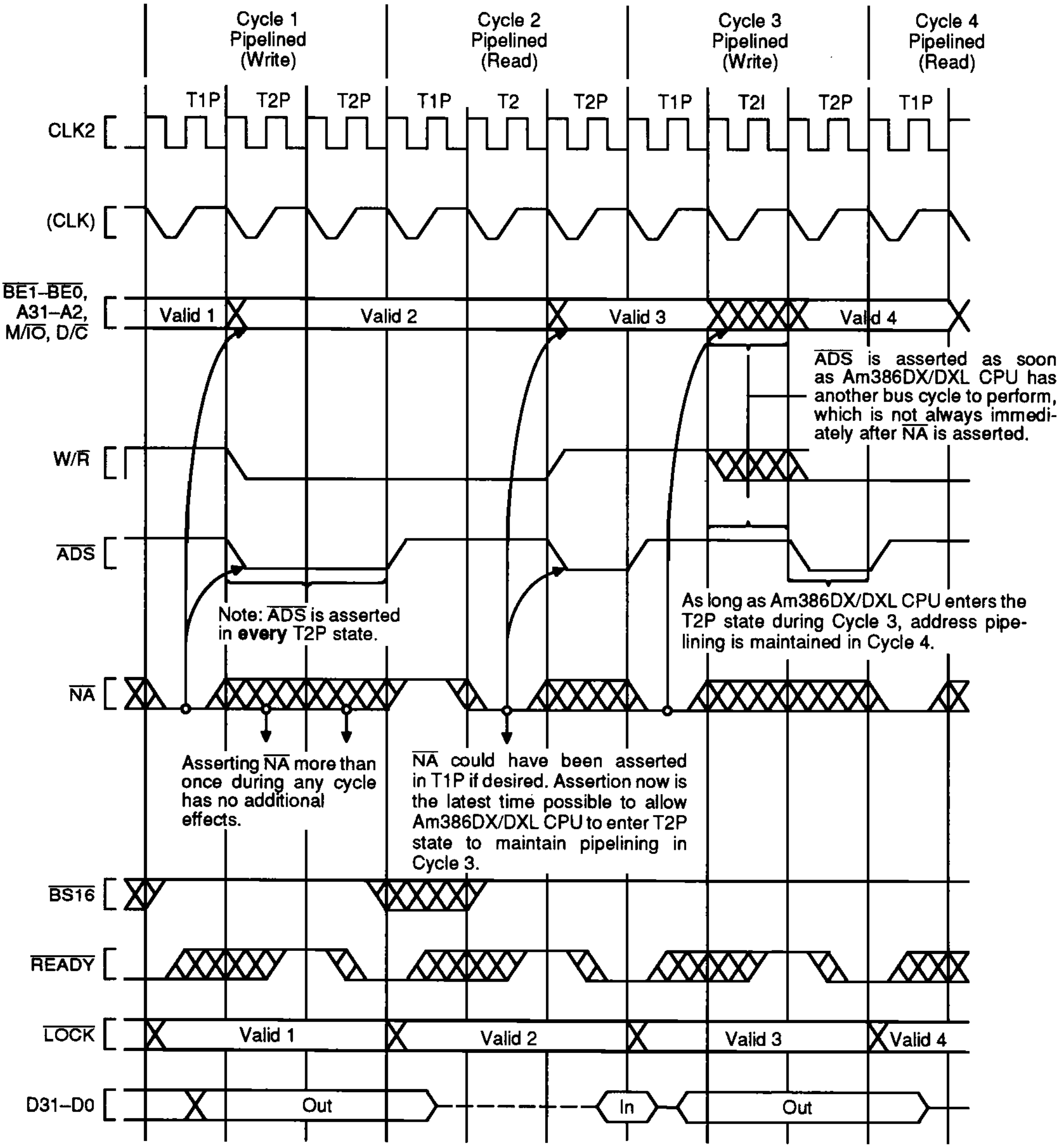
Once a bus cycle is in progress and the current address has become valid, the $\overline{NA}$ input is sampled at the end of every phase one, beginning with the next bus state, until the bus cycle is acknowledged. During Figure 57 Cycle 1 therefore, sampling begins in T2. Once $\overline{NA}$ is sampled asserted during the current cycle, the Am386DX/DXL microprocessor is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 56 Cycle 1 for example, the next address is driven during state T2P. Thus, Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as $\overline{READY}$ asserted terminates Cycle 1.

Example transition bus cycles are Figure 57 Cycle 1 and Figure 56 Cycle 2. Figure 57 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 56 Cycle 2, shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (you assert $\overline{NA}$ at that time), and T2P (provided the Am386DX/DXL microprocessor has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note three states (T1, T2, and T2P) are only required in a bus cycle performing a transition from non-pipelined address into pipelined address timing; for example, Figure 57 Cycle 1. Figure 57 Cycles 2, 3, and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting $\overline{NA}$ and detecting that the Am386DX/DXL CPU enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of $\overline{ADS}$. Figures 56 and 57 however, show pipelining ending after Cycle 4, because Cycle 4 ends in T2P. This indicates the Am386DX/DXL CPU did not have an internal bus request prior to the acknowledgment of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

Realistically, address pipelining is almost always maintained as long as $\overline{NA}$ is sampled asserted. This is so, because in the absence of any other request a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore, address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD negated) and $\overline{NA}$ is sampled asserted in each of the bus cycles.



ADS is asserted as soon as Am386DX/DXL CPU has another bus cycle to perform, which is not always immediately after NA is asserted.

Note: ADS is asserted in **every** T2P state.

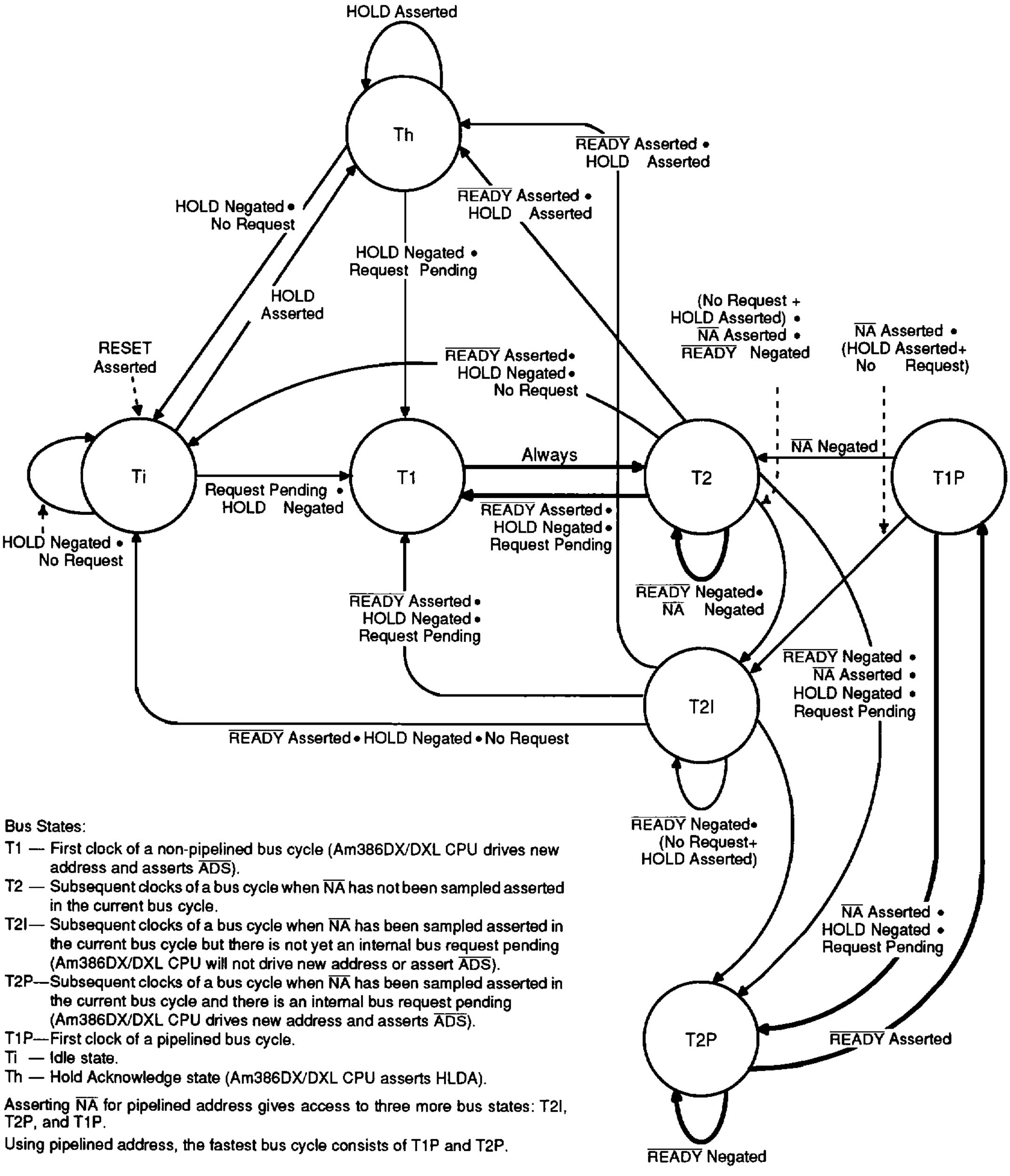
As long as Am386DX/DXL CPU enters the T2P state during Cycle 3, address pipelining is maintained in Cycle 4.

Asserting NA more than once during any cycle has no additional effects.

NA could have been asserted in T1P if desired. Assertion now is the latest time possible to allow Am386DX/DXL CPU to enter T2P state to maintain pipelining in Cycle 3.

15021B–061

Figure 58. Details of Address Pipelining During Cycles with Wait States



Bus States:

T1 — First clock of a non-pipelined bus cycle (Am386DX/DXL CPU drives new address and asserts $\overline{\text{ADS}}$).

T2 — Subsequent clocks of a bus cycle when $\overline{\text{NA}}$ has not been sampled asserted in the current bus cycle.

T2I— Subsequent clocks of a bus cycle when $\overline{\text{NA}}$ has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (Am386DX/DXL CPU will not drive new address or assert $\overline{\text{ADS}}$).

T2P—Subsequent clocks of a bus cycle when $\overline{\text{NA}}$ has been sampled asserted in the current bus cycle and there is an internal bus request pending (Am386DX/DXL CPU drives new address and asserts $\overline{\text{ADS}}$).

T1P—First clock of a pipelined bus cycle.

Ti — Idle state.

Th — Hold Acknowledge state (Am386DX/DXL CPU asserts HLDA).

Asserting $\overline{\text{NA}}$ for pipelined address gives access to three more bus states: T2I, T2P, and T1P.

Using pipelined address, the fastest bus cycle consists of T1P and T2P.

15021B–062

Figure 59. Am386DX/DXL Microprocessor Complete Bus States (Including Pipelined Address)

Pipelined Address With Dynamic Data Bus Sizing

The $\overline{\text{BS16}}$ feature allows easy interface to 16-bit data buses. When asserted, the Am386DX/DXL microprocessor bus interface hardware performs appropriate action to make the transfer using a 16-bit data bus connected on D15–D0.

There is a degree of interaction, however, between the use of Address Pipelining and the use of Bus Size 16. The interaction results from the multiple bus cycles required when transferring 32-bit operands over a 16-bit bus. If the operand requires both 16-bit halves of the 32-bit bus, the appropriate Am386DX/DXL microprocessor action is a second bus cycle to complete the operand's transfer. This necessity conflicts with $\overline{\text{NA}}$ usage.

When $\overline{\text{NA}}$ is sampled asserted, the Am386DX/DXL microprocessor commits itself to perform the next internally pending bus request, and is allowed to drive the next internally pending address onto the bus. Asserting $\overline{\text{NA}}$ therefore makes it impossible for the next bus cycle to again access the current address on A31–A2, such as may be required when $\overline{\text{BS16}}$ is asserted by the external hardware.

To avoid conflict, the Am386DX/DXL microprocessor is designed with following two provisions.

1. To avoid conflict, $\overline{\text{BS16}}$ must be negated in the current bus cycle if $\overline{\text{NA}}$ has already been sampled asserted in the current cycle. If $\overline{\text{NA}}$ is sampled asserted, the current data bus size is assumed to be 32 bits.
2. Also to avoid conflict, if $\overline{\text{NA}}$ and $\overline{\text{BS16}}$ are both asserted during the same sampling window, $\overline{\text{BS16}}$ asserted has priority and the Am386DX/DXL microprocessor acts as if $\overline{\text{NA}}$ was negated at that time.

Certain types of 16- or 8-bit operands require no adjustment for correct transfer on a 16-bit bus. Those are read or write operands using only the lower half of the data bus, and write operands using only the upper half of the bus, since the Am386DX/DXL CPU simultaneously duplicates the write data on the lower half of the data bus. For these patterns of Byte Enables and the W/$\overline{\text{R}}$ signals, $\overline{\text{BS16}}$ need not be asserted at the Am386DX/DXL CPU allowing $\overline{\text{NA}}$ to be asserted during the bus cycle if desired.

Interrupt Acknowledge (INTA) Cycles

In response to an interrupt request on the INTR input when interrupts are enabled, the Am386DX/DXL microprocessor performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by $\overline{\text{READY}}$ sampled asserted.

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, $\overline{\text{BE3}}$–$\overline{\text{BE1}}$ High, and $\overline{\text{BE0}}$ Low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 Low, $\overline{\text{BE3}}$–$\overline{\text{BE1}}$ High, $\overline{\text{BE0}}$ Low).

The $\overline{\text{LOCK}}$ output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, Ti, are inserted by the Am386DX/DXL microprocessor between the two interrupt acknowledge cycles, allowing for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D31–D0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the Am386DX/DXL microprocessor will read an external interrupt vector from D7–D0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.

Halt Indication Cycle

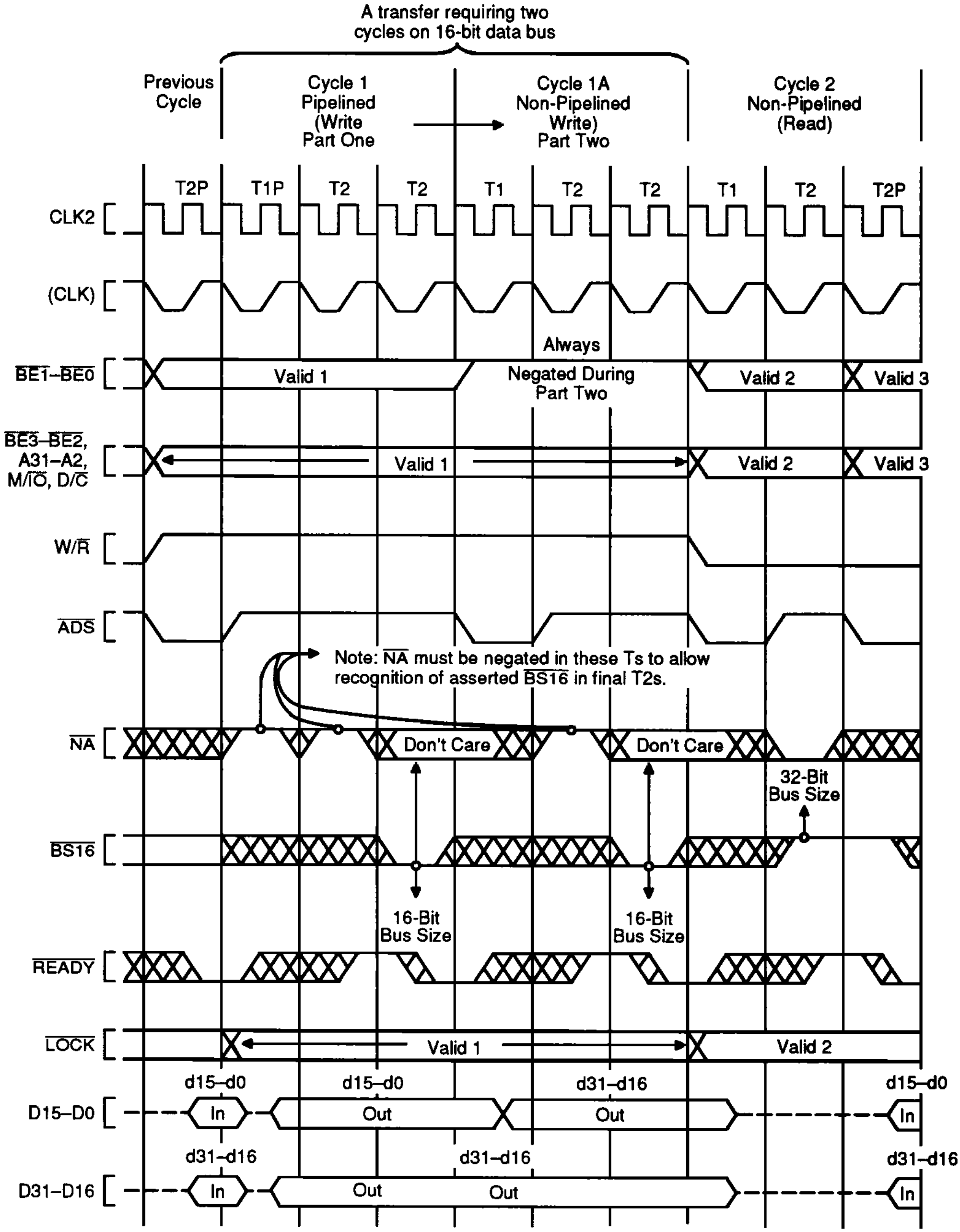
The Am386DX/DXL microprocessor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown in Bus cycle Definition and a byte address of 2. $\overline{\text{BE0}}$ and $\overline{\text{BE2}}$ are the only signals distinguishing halt indication from shutdown indication, that drives an address of 0. During the halt cycle undefined data is driven on D31–D0. The halt indication cycle must be acknowledged by $\overline{\text{READY}}$ asserted.

A halted Am386DX/DXL CPU resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

Shutdown Indication Cycle

The Am386DX/DXL microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in Bus Cycle Definition and a byte address of 0. $\overline{\text{BE0}}$ and $\overline{\text{BE2}}$ are the only signals distinguishing shutdown indication from halt indication, which drives an address of 2. During the shutdown cycle, undefined data is driven on D31–D0. The shutdown indication cycle must be acknowledged by $\overline{\text{READY}}$ asserted.

A shutdown Am386DX/DXL microprocessor resumes execution when NMI or RESET is asserted.

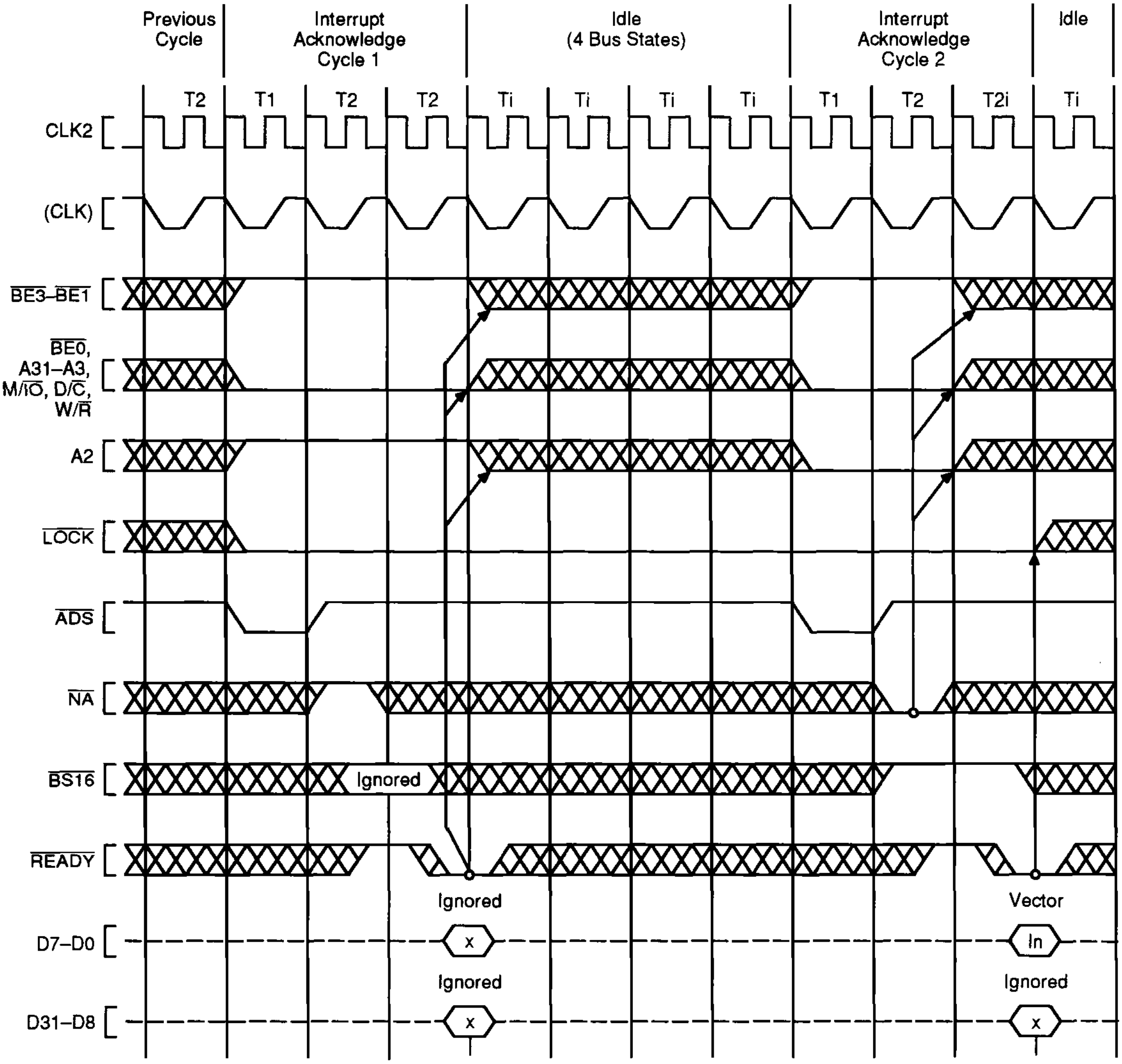


Key: Dn = Physical data pin n
dn = Logical data pin n

Cycle 1 is pipelined. Cycle 1A cannot be pipelined, but its address can be inferred from that of Cycle 1, to externally simulate address pipelining during Cycle 1A.

15021B–063

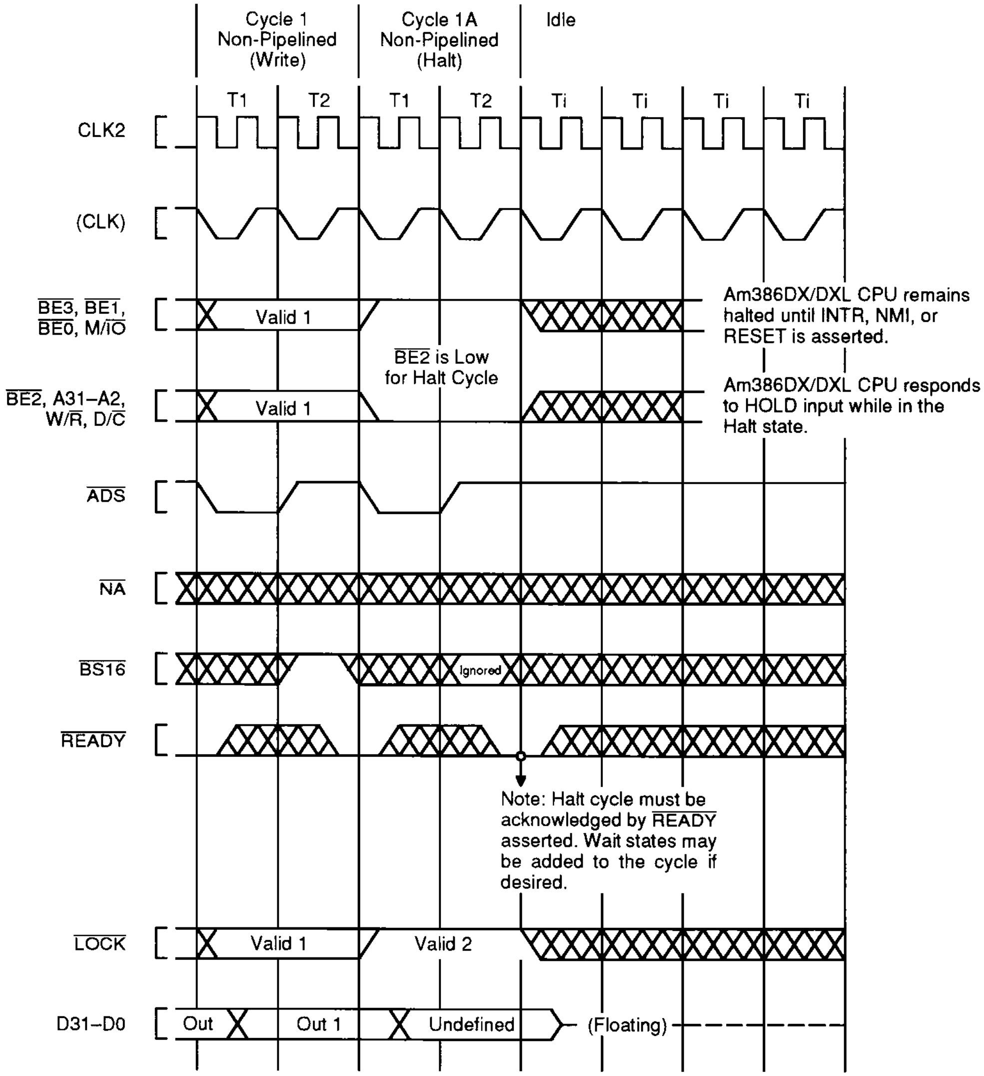
Figure 60. Using NA and BS16



Interrupt Vector (0–255) is read on D7–D0 at end of second Interrupt Acknowledge bus cycle.
Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting NA has no practical effect. Choose the approach that is simplest for your system hardware design.

15021B–064

Figure 61. Interrupt Acknowledge Cycles



15021B–065

Figure 62. Halt Indication Cycle

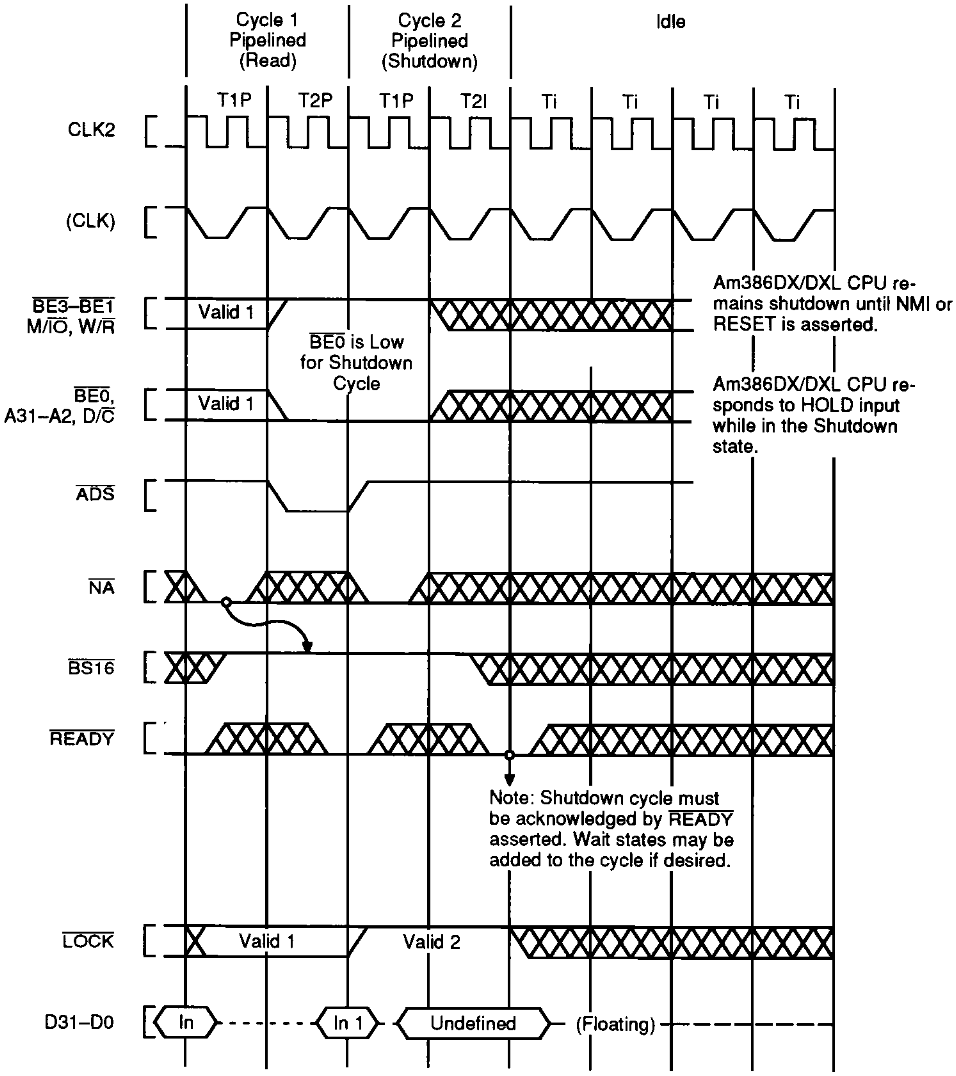
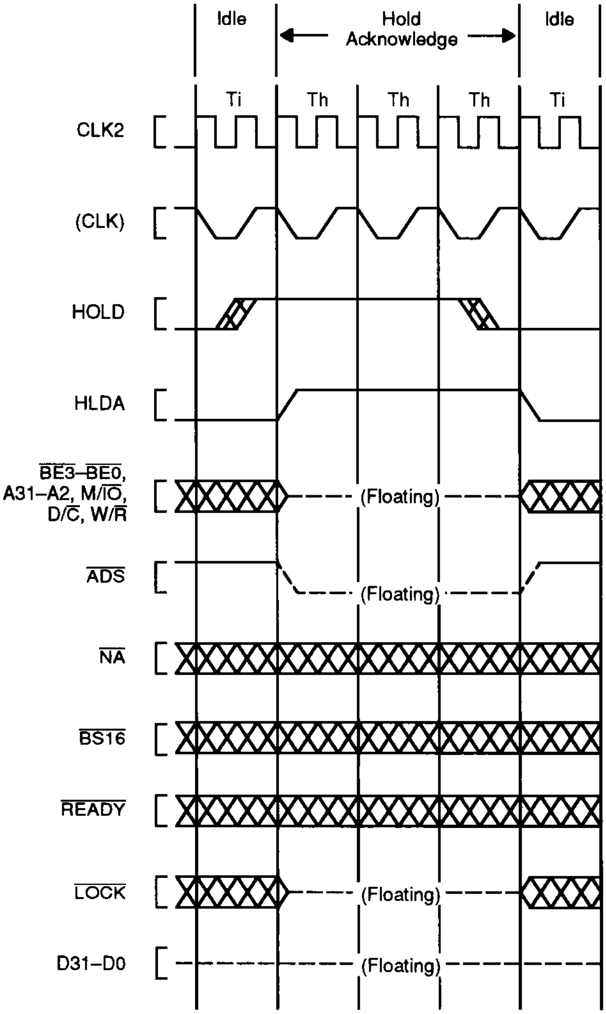


Figure 63. Shutdown Indication Cycle

Other Functional Descriptions

Entering and Exiting Hold Acknowledge

The Bus Hold Acknowledge State, Th, is entered in response to the HOLD input being asserted. In the Bus Hold Acknowledge state, the Am386DX/DXL microprocessor floats all output or bidirectional signals, except for HLDA. HLDA is asserted as long as the Am386DX/DXL CPU remains in the bus hold acknowledge state. In the Bus Hold Acknowledge state, all inputs except HOLD, $\overline{FLT}$, RESET, $\overline{BUSY}$, $\overline{ERROR}$, and PEREQ are ignored (also up to one rising edge on NMI is remembered for processing when HOLD is no longer asserted).



Note: For maximum design flexibility, the Am386DX/DXL CPU has no internal pullup resistors on its outputs. The design may require an external pullup on $\overline{ADS}$ and other Am386DX/DXL CPU outputs to keep them negated during float periods.

15021B–067

Figure 64. Requesting Hold from Idle Bus

Th may be entered from a bus idle state, as in Figure 64, or after the acknowledgment of the current physical bus cycle if the $\overline{\text{LOCK}}$ signal is not asserted, as in Figures 65 and 66. If HOLD is asserted during a locked bus cycle, the Am386DX/DXL microprocessor may execute one unlocked bus cycle before acknowledging HOLD. If asserting $\overline{\text{BS16}}$ requires a second 16-bit bus cycle to complete a physical operand transfer, it is performed before HOLD is acknowledged, although the bus state diagrams in Figures 53 and 59 do not indicate that detail.

Th is exited in response to the HOLD input being negated. The following state will be Ti as in Figure 64 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 65 and 66.

Th is also exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in Th, the event is remembered as a non-maskable interrupt 2 and is serviced when Th is exited, unless of course, the Am386DX/DXL microprocessor is reset before Th is exited.

RESET During HOLD Acknowledge

RESET being asserted takes priority over HOLD being asserted. Therefore, Th is exited in response to the RESET input being asserted. If RESET is asserted while HOLD remains asserted, the Am386DX/DXL microprocessor drives its pins to defined states during reset, as in Table 15 Pin State During RESET, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is negated, the Am386DX/DXL microprocessor enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the Am386DX/DXL microprocessor would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is negated, the $\overline{\text{BUSY}}$ input is still sampled as usual to determine whether a self test is being requested, and $\overline{\text{ERROR}}$ is still sampled as usual to determine whether a 387DX math coprocessor versus an 80287 (or none) is present.

Float

Activating the $\overline{\text{FLT}}$ input floats all Am386DX/DXL CPU bidirectional and output signals, including HLDA. Asserting $\overline{\text{FLT}}$ isolates the Am386DX/DXL CPU from the surrounding circuitry.

As the Am386DX/DXL microprocessor is packaged in a surface mount PQFP, it cannot be removed from the motherboard when In-Circuit Emulation (ICE) is needed. The $\overline{\text{FLT}}$ input allows the Am386DX/DXL CPU to be electrically isolated from the surrounding circuitry. This allows connection of an emulator to the Am386DX/DXL microprocessor PQFP without removing it from the PCB. This method of emulation is referred to as ON-Circuit Emulation (ONCE).

Entering and Exiting Float

$\overline{\text{FLT}}$ is an asynchronous, active Low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the Am386DX/DXL microprocessor (Figure 68). $\overline{\text{FLT}}$ must be held Low for a minimum of 16-CLK2 cycles. Reset should be asserted and held asserted until after $\overline{\text{FLT}}$ is deasserted. This will ensure that the Am386DX/DXL CPU will exit Float in a valid state.

Asserting the $\overline{\text{FLT}}$ input unconditionally aborts the current bus cycle and forces the Am386DX/DXL microprocessor into the Float mode. Since activating $\overline{\text{FLT}}$ unconditionally forces the Am386DX/DXL CPU into Float mode, the Am386DX/DXL CPU is not guaranteed to enter Float in a valid state. After deactivating $\overline{\text{FLT}}$, the Am386DX/DXL CPU is not guaranteed to exit Float mode in a valid state. This is not a problem, as the $\overline{\text{FLT}}$ pin is meant to be used only during ONCE. After exiting Float, the Am386DX/DXL CPU must be reset to return it to a valid state. Reset should be asserted before $\overline{\text{FLT}}$ is deasserted. This will ensure that the Am386DX/DXL CPU will exit Float in a valid state.

$\overline{\text{FLT}}$ has an internal pull-up resistor, and if it is not used it should be unconnected.

Bus Activity During and Following Reset

RESET is the highest priority input signal capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage; or idle states or bus hold acknowledge states discontinued so that the RESET state is established.

RESET should remain asserted for at least 15-CLK2 periods to ensure it is recognized throughout the Am386DX/DXL microprocessor, and at least 80-CLK2 periods if Am386DX/DXL device self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

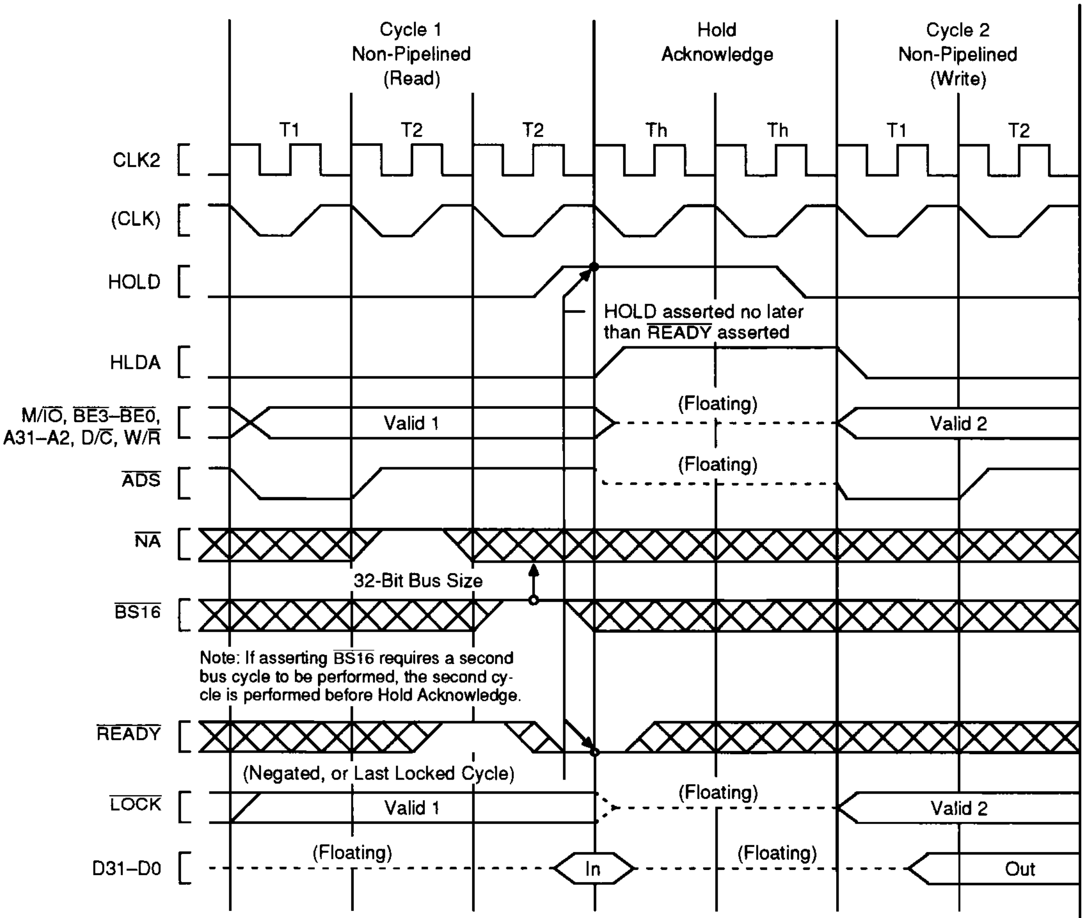
The additional RESET pulse width is required to clear additional state prior to valid self-test.

Provided the RESET falling edge meets setup and hold times, t25 and t26, the internal processor clock phase is defined at that time, as illustrated by Figure 67.

An Am386DX/DXL microprocessor self-test may be requested at the time RESET is negated by having the $\overline{\text{BUSY}}$ input at a Low level, as shown in Figure 67. The self-test requires (2^{20}) + approximately 60-CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the Am386DX/DXL device attempts to proceed with the reset sequence afterward.

After the RESET falling edge (and after the self-test if it was requested) the Am386DX/DXL microprocessor performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

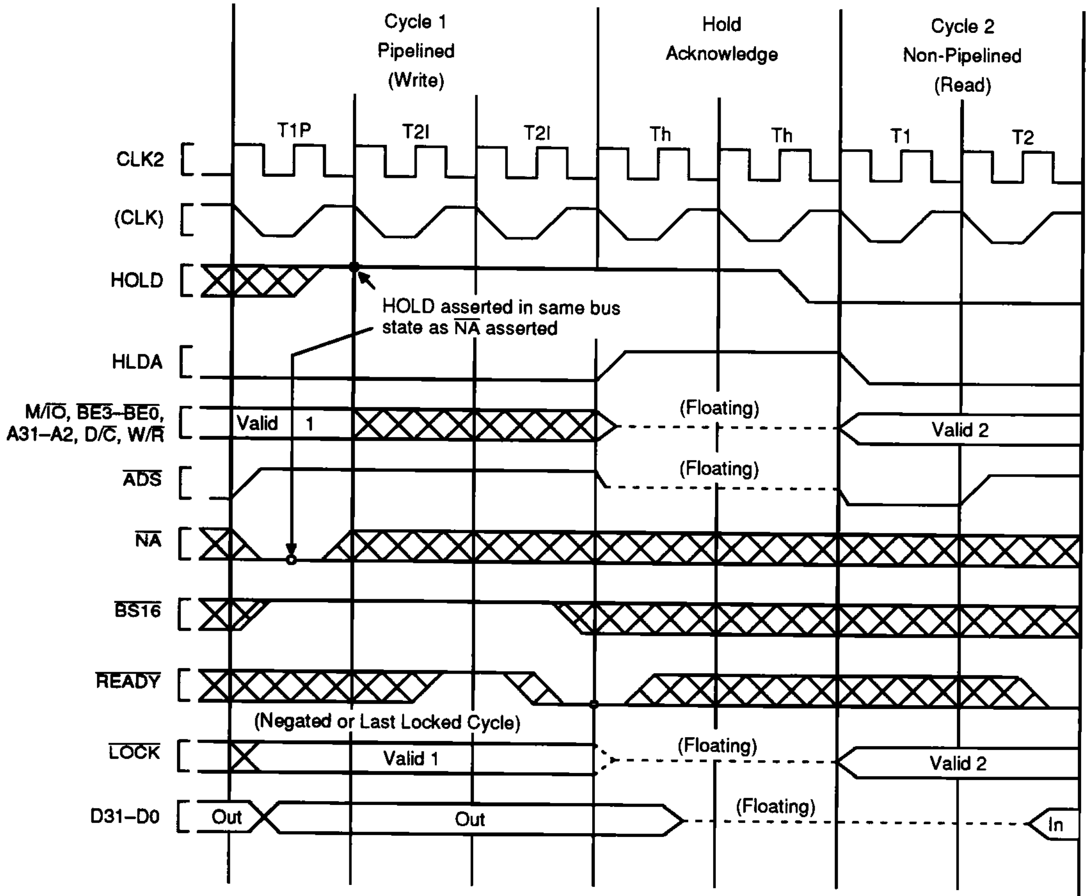
The Am386DX/DXL microprocessor samples its $\overline{\text{ERROR}}$ input some time after the falling edge of RESET and before executing the first ESC instruction. During this sampling period $\overline{\text{BUSY}}$ must be High. If $\overline{\text{ERROR}}$ was sampled active, the Am386DX/DXL device employs the 32-bit protocol of a 387DX math coprocessor. Even though this protocol was selected, it is still necessary to use a software recognition test to determine the presence or identity of the coprocessor and to assure compatibility with future processors.



Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t23 and t24) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

15021B–068

Figure 65. Requesting Hold from Active Bus ($\overline{\text{NA}}$ Negated)



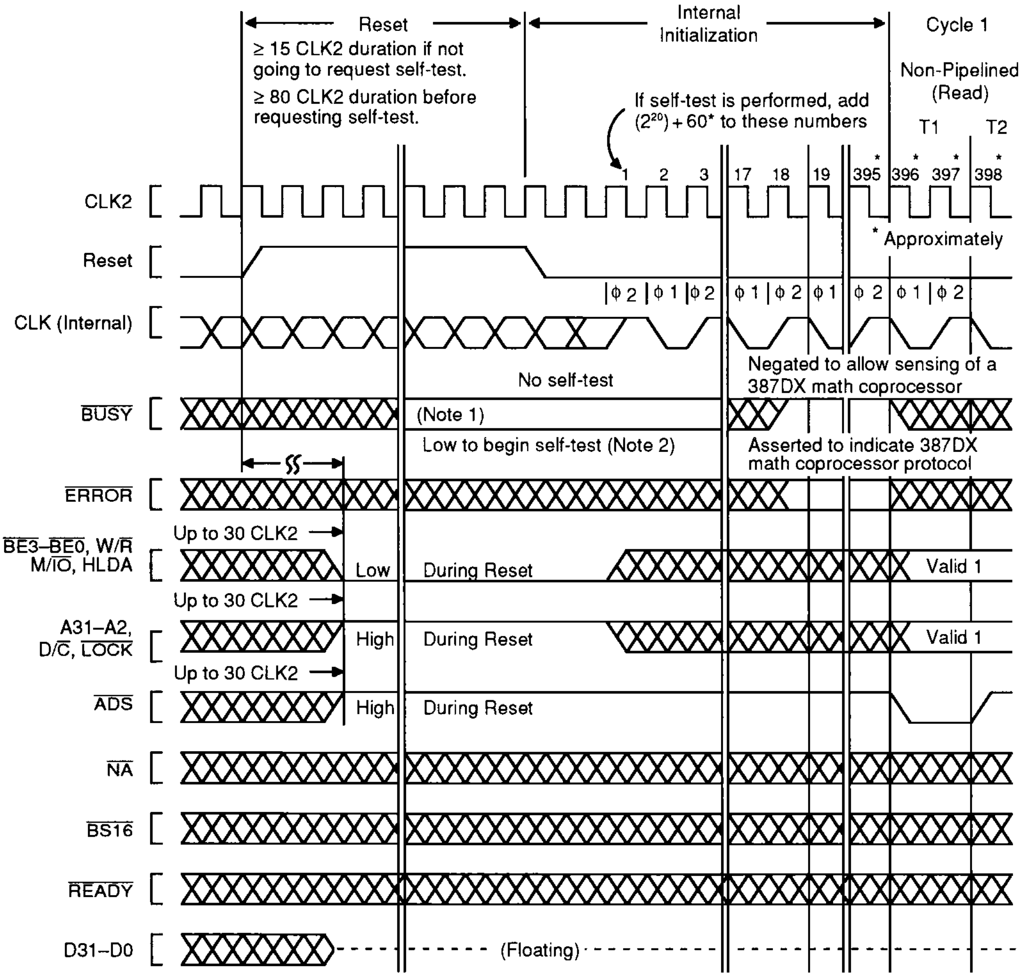
Note: HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t23 and t24) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

15021B–069

Figure 66. Requesting Hold from Active Bus (NA Asserted)

Table 21. Component and Revision Identifier History

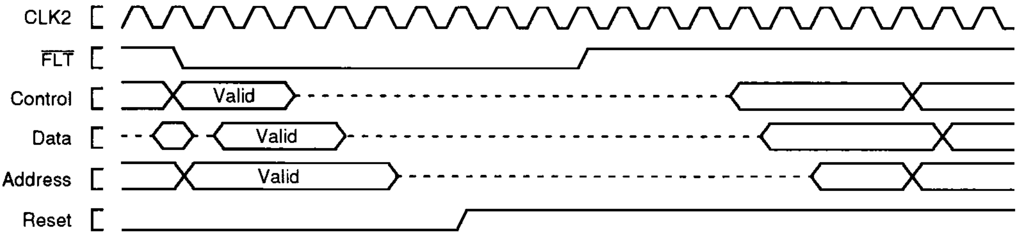
Intel i386 Stepping Name	Am386DX/DXL Microprocessor Revision	Component Identifier	Revision Identifier
D1	D	03	08



Notes: 1. BUSY should be held stable for 8-CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested, the Am386DX/DXL microprocessor outputs remain in their reset state as shown here and in Table 14.

Figure 67. Bus Activity from Reset Until First Code Fetch

15021B–070



15022B–029

Figure 68. Entering and Exiting FLT

Self-Test Signature

Upon completion of self-test, (if self-test was requested by holding $\overline{\text{BUSY}}$ Low at least eight CLK2 periods before and after the falling edge of RESET), the EAX register will contain a signature of 00000000h indicating the Am386DX/DXL CPU passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000h, applies to all Am386DX/DXL microprocessor revision levels. Any non-zero signature indicates the Am386DX/DXL CPU unit is faulty.

Component and Revision Identifiers

To assist Am386DX/DXL microprocessor users, the microprocessor after reset holds a component identifier and a revision identifier in its DX register. The upper 8 bits of DX hold 03h as identification of the Am386DX/DXL CPU component. The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier begins chronologically with a value zero and is subject to change (typically it will be incremented) with component steppings intended to have certain improvements or distinctions from previous steppings.

These features are intended to assist Am386DX/DXL microprocessor users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision nor to follow a completely uniform numerical sequence, depending on the type or intention of revision or manufacturing materials required to be changed.

Coprocessor Interfacing

The Am386DX/DXL microprocessor provides an automatic interface for a 387DX floating-point math coprocessor. A 387DX math coprocessor uses an I/O-mapped interface driven automatically by the Am386DX/DXL microprocessor and assisted by three dedicated signals: $\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$, and PEREQ.

As the Am386DX/DXL CPU begins supporting a coprocessor instruction, it tests the $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ signals to determine if the coprocessor can accept its next instruction. Thus, the $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ inputs eliminate the need for any preamble bus cycles for communication between processor and coprocessor. A 387DX math coprocessor can be given its command op-code immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the Am386DX/DXL CPU WAIT op-code (9Bh) for 387DX math coprocessor instruction synchronization (the WAIT op-code was required when 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in Am386DX/DXL microprocessor based systems, via memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol primitives. Instead, memory-mapped or I/O-mapped interfaces may use all applicable Am386DX/DXL microprocessor instructions for high-speed coprocessor communication. The $\overline{\text{BUSY}}$ and $\overline{\text{ERROR}}$ inputs of the Am386DX/DXL CPU may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the Am386DX/DXL CPU WAIT op-code (9Bh). The WAIT instruction will wait until the $\overline{\text{BUSY}}$ input is negated (interruptable by an NMI or enable INTR input), but generates an Exception 16 fault if the $\overline{\text{ERROR}}$ pin is in the asserted state when the $\overline{\text{BUSY}}$ goes (or is) negated. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the Am386DX/DXL microprocessor on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the Am386DX/DXL microprocessor IOPL (I/O Privilege Level) mechanism.

A 387DX math coprocessor interface is I/O mapped as shown in Table 22. Note that a 387DX math coprocessor interface addresses are beyond the 0hFFFFh range for programmed I/O. When the Am386DX/DXL CPU supports a 387DX math coprocessor, the Am386DX/DXL microprocessor automatically generates bus cycles to the coprocessor interface addresses.

Table 22. Math Coprocessor Port Addresses

Address in Am386DX/DXL CPU I/O Space	387DX Coprocessor Register
800000F8h	Opcode Register (32-bit port)
800000FCh	Operand Register (32-bit port)

To correctly map a 387DX math coprocessor registers to the appropriate I/O addresses, connect a 387DX math coprocessor $\overline{\text{CMD0}}$ pin directly to the A2 output of the Am386DX/DXL microprocessor.

Software Testing for Coprocessor Presence

When software is used to test for coprocessor (387DX) presence, it should use only the following coprocessor op-codes: FINIT, FNINIT, FSTCW mem, FSTSW mem, FSTSW AX. To use other coprocessor op-codes when a coprocessor is known to be not present, first set EM = 1 in Am386DX/DXL microprocessor CR0.

ABSOLUTE MAXIMUM RATINGS

Storage Temperature . . . . –65°C to +150°C
Ambient Temperature Under Bias . . –65°C to +125°C
Supply Voltage with Respect
to V_{SS} . . . . –0.5 V to +7 V
Voltage on Other Pins . . . . –0.5 V to V_{CC} +0.5 V

Stresses above those listed under ABSOLUTE MAXIMUM RATINGS may cause permanent device failure. Functionality at or above these limits is not implied. Exposure to Absolute Maximum Ratings for extended periods may affect device reliability.

DC CHARACTERISTICS over COMMERCIAL operating ranges

V_{CC} = 5 V ±5%; T_{CASE} = 0°C to +85°C (PGA)
V_{CC} = 5 V±10%; T_{CASE} = 0°C to +100°C (PQFP—20, 25, and 33 MHz)
V_{CC} = 5 V±5%; T_{CASE} = 0°C to +100°C (PQFP—40 MHz)

Symbol	Parameter Description	Notes	Min	Max	Unit
V_{IL}	Input Low Voltage	(Note 1)	–0.3	0.8	V
V_{IH}	Input High Voltage		2.0	V_{CC} + 0.3	V
V_{ILC}	CLK2 Input Low Voltage	(Note 1)	–0.3	0.8	V
V_{IHC}	CLK2 Input High Voltage		2.7	V_{CC} +0.3	V
V_{OL}	Output Low Voltage				
	I_{OL} = 4 mA: A31–A2, D31–D0	(Note 6)		0.45	V
	I_{OL} = 5 mA: $\overline{BE3}$–$\overline{BE0}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA			0.45	V
V_{OH}	Output High Voltage				
	I_{OH} = 1 mA: A31–A2, D31–D0	(Note 6)	2.4		V
	I_{OH} = 0.9 mA: $\overline{BE3}$–$\overline{BE0}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA		2.4		V
I_{LI}	Input Leakage Current (All pins except $\overline{BS16}$, PEREQ, $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$)	$0\ V \le V_{IN} \le V_{CC}$		±15	µA
I_{IH}	Input Leakage Current (PEREQ Pin)	V_{IH} = 2.4 V (Note 2)		200	µA
I_{IL}	Input Leakage Current ($\overline{BS16}$, $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$)	V_{IL} = 0.45 (Note 3)		–400	µA
I_{LO}	Output Leakage Current	$0.45\ V \le V_{OUT} \le V_{CC}$		±15	µA
I_{CC}	Supply Current	(Note 7) V_{CC} = 5.0 V		V_{CC} = 5.5 V	
	CLK2 = 40 MHz: with –20	I_{CC} Typ = 130		155	mA
	CLK2 = 50 MHz: with –25	I_{CC} Typ = 160		190	mA
	CLK2 = 66 MHz: with –33	I_{CC} Typ = 210		245	mA
	CLK2 = 80 MHz: with –40	I_{CC} Typ = 330		400	mA
I_{CCSB}	Standby Current (Am386DXL microprocessor)	I_{CCSB} Typ = 20 µA (Note 5)		150	µA
C_{IN}	Input or I/O Capacitance	F_C = 1 MHz (Note 4)		10	pF
C_{OUT}	Output Capacitance	F_C = 1 MHz (Note 4)		12	pF
C_{CLK}	CLK2 Capacitance	F_C = 1 MHz (Note 4)		20	pF

Notes: 1. The Min value, –0.3, is not 100% tested.
2. PEREQ input has an internal pulldown resistor.
3. $\overline{BS16}$, $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$ inputs each have an internal pullup resistor.
4. Not 100% tested.
5. Measurement taken with inputs at rails, outputs unloaded, $\overline{BS16}$, $\overline{BUSY}$, $\overline{FLT}$, and $\overline{ERROR}$ at V_{CC} voltage level, PEREQ at Gnd.
6. Outputs are CMOS and will pull to rail if load is not resistive.
7. Inputs at rails (V_{CC} or V_{SS}).

SWITCHING CHARACTERISTICS over COMMERCIAL operating range — 40 MHz

V_{CC} = 5 V ±5%; T_{CASE} = 0°C to +85°C (PGA)
V_{CC} = 5 V ±5%; T_{CASE} = 0°C to +100°C (PQFP)

No.	Parameter Description	Notes	Ref Figure	Min	Max	Unit
	Oper. Frequency: Am386DX CPU Am386DXL CPU	Half CLK2 freq. Half CLK2 freq.		2 0	40 40	MHz MHz
1	CLK2 Period: Am386DX CPU Am386DXL CPU		71 71	12.5 12.5	250	ns ns
2	CLK2 High Time	at V_{IHC}	71	4		ns
3	CLK2 Low Time	at 0.8 V	71	5		ns
4	CLK2 Fall Time	2.7 V–0.8 V (Note 3)	71		4	ns
5	CLK2 Rise Time	0.8 V–2.7 V (Note 3)	71		4	ns
6	A31–A2 Valid Delay	C_L = 50 pF	70, 73, 81	4	13	ns
7	A31–A2 Float Delay	(Note 1)	81	4	20	ns
8	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Valid Delay	C_L = 50 pF	70, 73, 81	4	13	ns
9	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Float Delay	(Note 1)	81	4	20	ns
10	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$ Valid Delay	C_L = 50 pF	70, 73, 81	4	13	ns
10a	$\overline{\text{ADS}}$ Valid Delay	C_L = 50 pF	70, 73, 81	4	13	ns
11	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Float Delay	(Note 1)	81	4	20	ns
12	D31–D0 Write Data Valid Delay	C_L = 50 pF (Note 4)	70, 74, 81	7	18	ns
12a	D31–D0 Write Data Hold Time	C_L = 50 pF	70, 75	2		ns
13	D31–D0 Float Delay	(Note 1)	81	4	17	ns
14	HLDA Valid Delay	C_L = 50 pF	70, 81	4	17	ns
14f	HLDA Float Delay (PQFP Only)	(Note 1)	70, 81	4	17	ns
15	$\overline{\text{NA}}$ Setup Time		72	5		ns
16	$\overline{\text{NA}}$ Hold Time		72	2		ns
17	$\overline{\text{BS16}}$ Setup Time		72	5		ns
18	$\overline{\text{BS16}}$ Hold Time		72	2		ns
19	$\overline{\text{READY}}$ Setup Time		72	7		ns
20	$\overline{\text{READY}}$ Hold Time		72	4		ns
21	D31–D0 Read Setup Time		72	4		ns
22	D31–D0 Read Hold Time		72	3		ns
23	HOLD Setup Time		72	4		ns
24	HOLD Hold Time		72	2		ns
25	RESET Setup Time		82	4		ns
26	RESET Hold Time		82	2		ns
27	NMI, INTR Setup Time	(Note 2)	72	5		ns
28	NMI, INTR Hold Time	(Note 2)	72	5		ns
29	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Setup Time	(Note 2)	72	5		ns
30	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Hold Time	(Note 2)	72	4		ns

Notes: 1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific clock period.
3. Rise and fall times are not tested.
4. Min time not 100% tested.
*PQFP package only.

SWITCHING CHARACTERISTICS over COMMERCIAL operating range — 33 MHz

$V_{CC} = 5$ V ±5%; $T_{CASE} = 0°C$ to +85°C (PGA)
$V_{CC} = 5$ V ±10%; $T_{CASE} = 0°C$ to +100°C (PQFP)

No.	Parameter Description	Notes	Ref Figure	Min	Max	Unit
	Oper. Frequency: Am386DX CPU Am386DXL CPU	Half CLK2 freq. Half CLK2 freq.		2 0	33.3 33.3	MHz MHz
1	CLK2 Period: Am386DX CPU Am386DXL CPU		71 71	15 15	250	ns ns
2	CLK2 High Time	at V_{IHC}	71	4		ns
3	CLK2 Low Time	at 0.8 V	71	5		ns
4	CLK2 Fall Time	2.7 V–0.8 V (Note 3)	71		4	ns
5	CLK2 Rise Time	0.8 V–2.7 V (Note 3)	71		4	ns
6	A31–A2 Valid Delay	$C_L = 50$ pF	70, 73, 81	4	15	ns
7	A31–A2 Float Delay	(Note 1)	81	4	20	ns
8	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Valid Delay	$C_L = 50$ pF	70, 73, 81	4	15	ns
9	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Float Delay	(Note 1)	81	4	20	ns
10	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$ Valid Delay	$C_L = 50$ pF	70, 73, 81	4	15	ns
10a	$\overline{\text{ADS}}$ Valid Delay	$C_L = 50$ pF	70, 73, 81	4	14.5	ns
11	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Float Delay	(Note 1)	81	4	20	ns
12	D31–D0 Write Data Valid Delay	$C_L = 50$ pF (Note 4)	70, 74, 81	7	23	ns
12a	D31–D0 Write Data Hold Time	$C_L = 50$ pF	70, 75	2		ns
13	D31–D0 Float Delay	(Note 1)	81	4	17	ns
14	HLDA Valid Delay	$C_L = 50$ pF	70, 81	4	20	ns
14f	HLDA Float Delay (PQFP Only)	(Note 1)	70, 81	4	20	ns
15	$\overline{\text{NA}}$ Setup Time		72	5		ns
16	$\overline{\text{NA}}$ Hold Time		72	2		ns
17	$\overline{\text{BS16}}$ Setup Time		72	5		ns
18	$\overline{\text{BS16}}$ Hold Time		72	2		ns
19	$\overline{\text{READY}}$ Setup Time		72	7		ns
20	$\overline{\text{READY}}$ Hold Time		72	4		ns
21	D31–D0 Read Setup Time		72	5		ns
22	D31–D0 Read Hold Time		72	3		ns
23	HOLD Setup Time		72	9		ns
24	HOLD Hold Time		72	2		ns
25	RESET Setup Time		82	5		ns
26	RESET Hold Time		82	2		ns
27	NMI, INTR Setup Time	(Note 2)	72	5		ns
28	NMI, INTR Hold Time	(Note 2)	72	5		ns
29	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Setup Time	(Note 2)	72	5		ns
30	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Hold Time	(Note 2)	72	4		ns

Notes: 1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific clock period.
3. Rise and fall times are not tested.
4. Min time not 100% tested.
*PQFP package only.

SWITCHING CHARACTERISTICS over COMMERCIAL operating range — 25 MHz

$V_{CC} = 5$ V ±5%; $T_{CASE} = 0$°C to +85°C (PGA)
$V_{CC} = 5$ V ±10%; $T_{CASE} = 0$°C to +100°C (PQFP)

No.	Parameter Description	Notes	Ref Figure	Min	Max	Unit
	Oper. Frequency: Am386DX CPU Am386DXL CPU	Half CLK2 freq. Half CLK2 freq.		2 0	25 25	MHz MHz
1	CLK2 Period: Am386DX CPU Am386DXL CPU		71 71	20 20	250	ns ns
2	CLK2 High Time	at V_{IHC}	71	4		ns
3	CLK2 Low Time	at 0.8 V	71	5		ns
4	CLK2 Fall Time	2.7 V–0.8 V (Note 3)	71		7	ns
5	CLK2 Rise Time	0.8 V–2.7 V (Note 3)	71		7	ns
6	A31–A2 Valid Delay	$C_L = 50$ pF	70, 73, 81	4	17	ns
7	A31–A2 Float Delay	(Note 1)	81	4	30	ns
8	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Valid Delay	$C_L = 50$ pF	70, 73, 81	4	17	ns
9	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Float Delay	(Note 1)	81	4	30	ns
10	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Valid Delay	$C_L = 50$ pF	70, 73, 81	4	17	ns
11	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Float Delay	(Note 1)	81	4	30	ns
12	D31–D0 Write Data Valid Delay	$C_L = 50$ pF	70, 74, 81	7	23	ns
12a	D31–D0 Write Data Hold Time	$C_L = 50$ pF	70, 75	2		ns
13	D31–D0 Float Delay	(Note 1)	81	4	22	ns
14	HLDA Valid Delay	$C_L = 50$ pF	70, 81	4	22	ns
14f	HLDA Float Delay (PQFP Only)	(Note 1)	70, 81	4	30	ns
15	$\overline{\text{NA}}$ Setup Time		72	5		ns
16	$\overline{\text{NA}}$ Hold Time		72	3		ns
17	$\overline{\text{BS16}}$ Setup Time		72	5		ns
18	$\overline{\text{BS16}}$ Hold Time		72	3		ns
19	$\overline{\text{READY}}$ Setup Time		72	9		ns
20	$\overline{\text{READY}}$ Hold Time		72	4		ns
21	D31–D0 Read Setup Time		72	7		ns
22	D31–D0 Read Hold Time		72	5		ns
23	HOLD Setup Time		72	9		ns
24	HOLD Hold Time		72	3		ns
25	RESET Setup Time		82	8		ns
26	RESET Hold Time		82	3		ns
27	NMI, INTR Setup Time	(Note 2)	72	6		ns
28	NMI, INTR Hold Time	(Note 2)	72	6		ns
29	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Setup Time	(Note 2)	72	6		ns
30	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Hold Time	(Note 2)	72	5		ns

Notes: 1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific clock period.
3. Rise and fall times are not tested.
*PQFP package only.

SWITCHING CHARACTERISTICS over COMMERCIAL operating range—20 MHz

V_{CC} = 5 V ±5%; T_{CASE} = 0°C to +85°C (PGA)
V_{CC} = 5 V ±10%; T_{CASE} = 0°C to +100°C (PQFP)

No.	Parameter Description	Notes	Ref Figure	Min	Max	Unit
	Oper. Frequency: Am386DX CPU Am386DXL CPU	Half CLK2 freq. Half CLK2 freq.		2 0	20 20	MHz MHz
1	CLK2 Period: Am386DX CPU Am386DXL CPU		71 71	25 25	250	ns ns
2	CLK2 High Time	at V_{IHC}	71	6		ns
3	CLK2 Low Time	at 0.8 V	71	6		ns
4	CLK2 Fall Time	2.7 V–0.8 V (Note 3)	71		8	ns
5	CLK2 Rise Time	0.8 V–2.7 V (Note 3)	71		8	ns
6	A31–A2 Valid Delay	C_L = 120 pF	70, 73, 81	4	30	ns
7	A31–A2 Float Delay	(Note 1)	81	4	32	ns
8	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Valid Delay	C_L = 75 pF	70, 73, 81	4	30	ns
9	$\overline{\text{BE3}}$–$\overline{\text{BE0}}$, $\overline{\text{LOCK}}$ Float Delay	(Note 1)	81	4	32	ns
10	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Valid Delay	C_L = 75 pF	70, 73, 81	4	28	ns
11	W/$\overline{\text{R}}$, M/$\overline{\text{IO}}$, D/$\overline{\text{C}}$, $\overline{\text{ADS}}$ Float Delay	(Note 1)	81	4	30	ns
12	D31–D0 Write Data Valid Delay	C_L = 120 pF	70, 74, 81	4	38	ns
13	D31–D0 Float Delay	(Note 1)	81	4	27	ns
14	HLDA Valid Delay	C_L = 75 pF	70, 81	6	28	ns
14f	HLDA Float Delay (PQFP Only)	(Note 1)	70, 81	4	30	ns
15	$\overline{\text{NA}}$ Setup Time		72	9		ns
16	$\overline{\text{NA}}$ Hold Time		72	14		ns
17	$\overline{\text{BS16}}$ Setup Time		72	13		ns
18	$\overline{\text{BS16}}$ Hold Time		72	21		ns
19	$\overline{\text{READY}}$ Setup Time		72	12		ns
20	$\overline{\text{READY}}$ Hold Time		72	4		ns
21	D31–D0 Read Setup Time		72	11		ns
22	D31–D0 Read Hold Time		72	6		ns
23	HOLD Setup Time		72	17		ns
24	HOLD Hold Time		72	5		ns
25	RESET Setup Time		82	12		ns
26	RESET Hold Time		82	4		ns
27	NMI, INTR Setup Time	(Note 2)	72	16		ns
28	NMI, INTR Hold Time	(Note 2)	72	16		ns
29	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Setup Time	(Note 2)	72	14		ns
30	PEREQ, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, $\overline{\text{FLT}}$* Hold Time	(Note 2)	72	5		ns

Notes: 1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.

2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific clock period.

3. Rise and fall times are not tested.

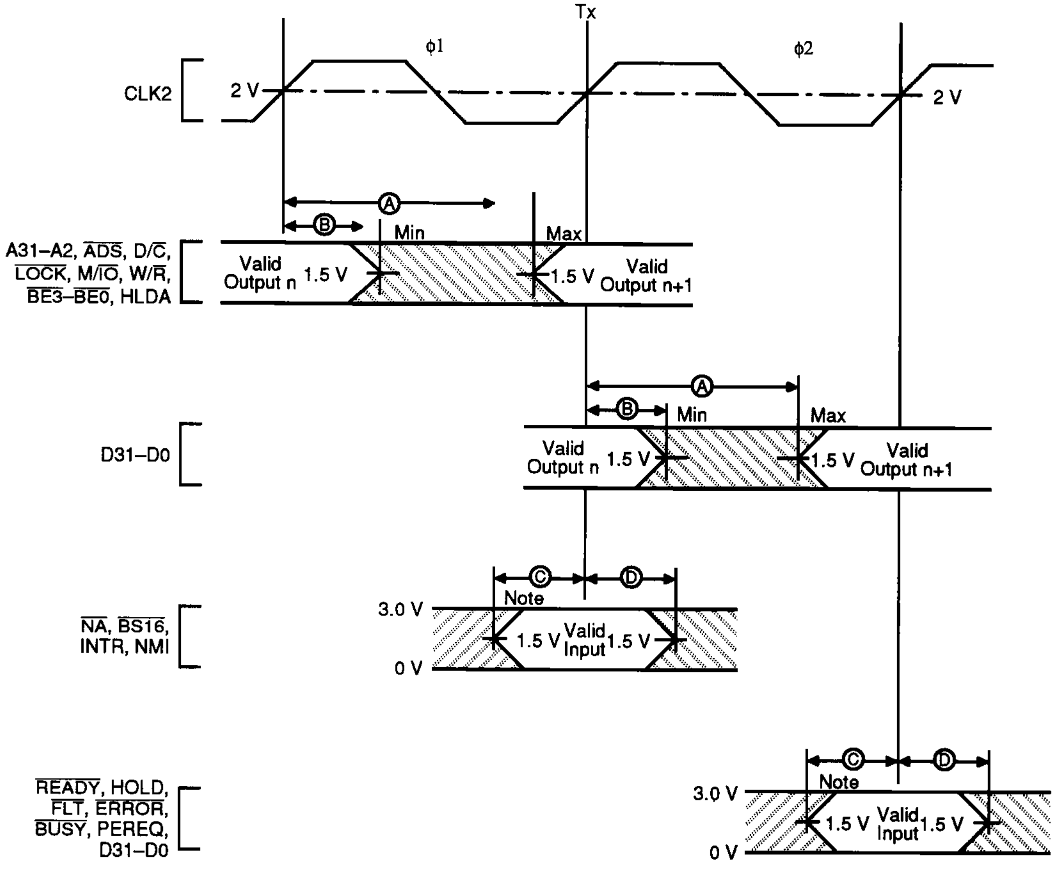
*PQFP package only.

SWITCHING WAVEFORMS

The switching characteristics consist of output delays, input setup requirements, and input hold requirements. All characteristics are relative to the CLK2 rising edge crossing the 2.0 V level.

Switching characteristic measurement is defined by Figure 69. Inputs must be driven to the voltage levels indicated by this diagram. Am386DX/DXL CPU output delays are specified with minimum and maximum limits measured as shown. The minimum Am386DX/DXL microprocessor delay times are hold times provided to external circuitry. Am386DX/DXL microprocessor input setup and hold time are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct Am386DX/DXL microprocessor operation.

Outputs $\overline{\text{ADS}}$, W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$, $\overline{\text{LOCK}}$, $\overline{\text{BE3}}$–$\overline{\text{BE0}}$, A31–A2, and HLDA only change at the beginning of phase one. D31–D0 (write cycles) only change at the beginning of phase two. The $\overline{\text{READY}}$, HOLD, $\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$, PEREQ, $\overline{\text{FLT}}$, and D31–D0 (read cycles) inputs are sampled at the beginning of phase one. The $\overline{\text{NA}}$, $\overline{\text{BS16}}$, INTR, and NMI inputs are sampled at the beginning of phase two.

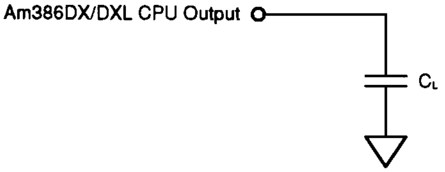


Legend: A—Maximum Output Delay Spec
B—Minimum Output Delay Spec
C—Minimum Input Setup Spec
D—Minimum Input Hold Spec

Note: Input waveforms have tr ≤ 2.0 ns from 0.8 V to 2.0 V.

15021B–071

Figure 69. Drive Levels and Measurement Points



C_L includes all parasitic capacitances.

15021B–072

Figure 70. AC Test Load

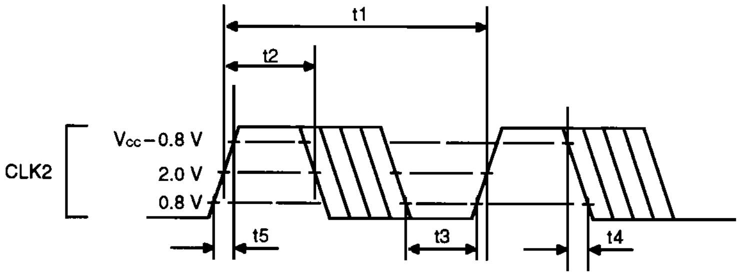


Figure 71. CLK2 Timing

15021B–073

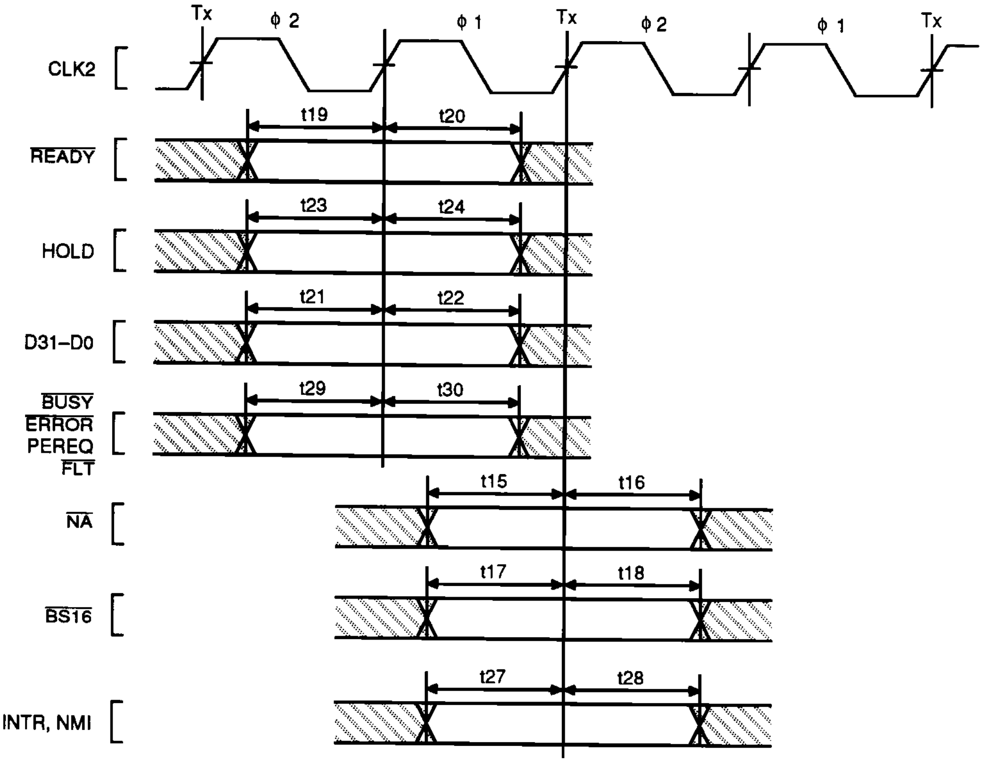


Figure 72. Input Setup and Hold Timing

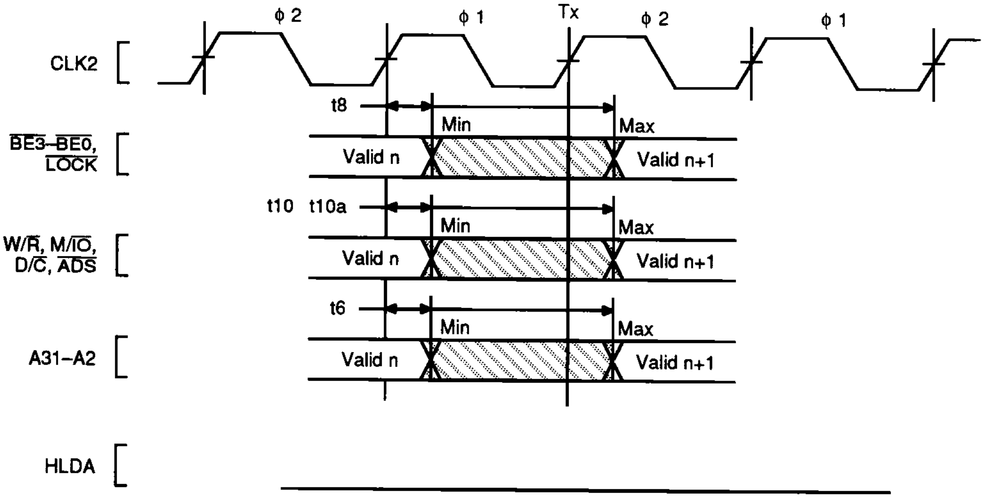
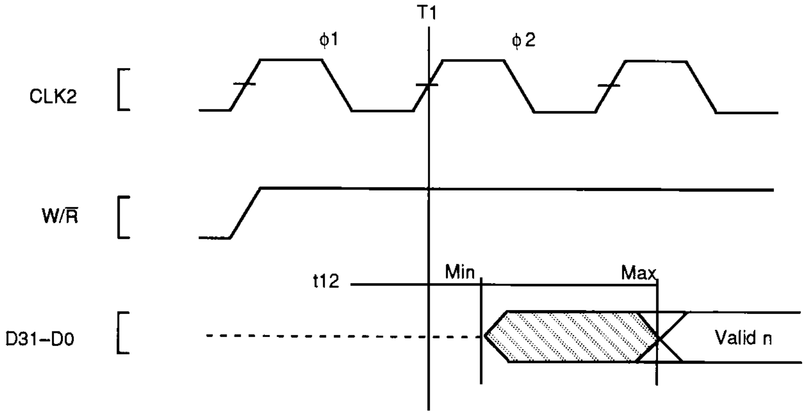
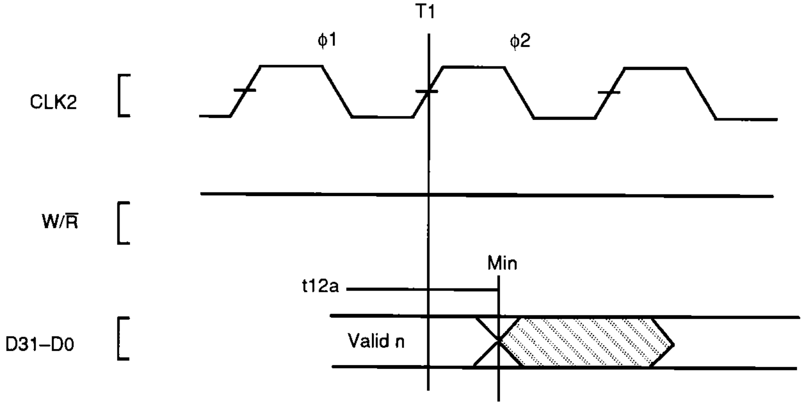


Figure 73. Output Valid Delay Timing



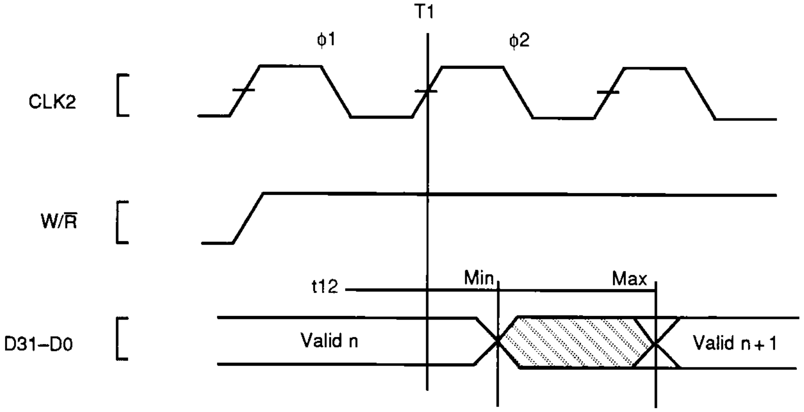
15021B–076

Figure 74. Write Data Valid Delay Timing (25, 33, and 40 MHz)



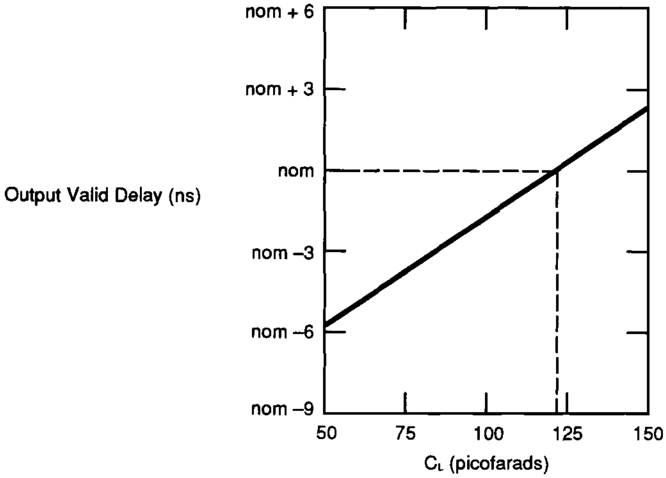
15021B–077

Figure 75. Write Data Hold Timing (25, 33, 40 MHz)



15021B–078

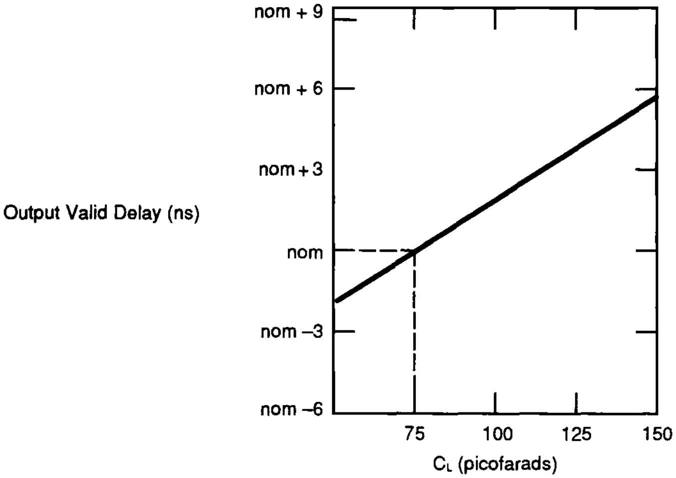
Figure 76. Write Data Valid Delay Timing (20 MHz)



Note: This graph will not be linear outside of the CL range shown.

15021B–079

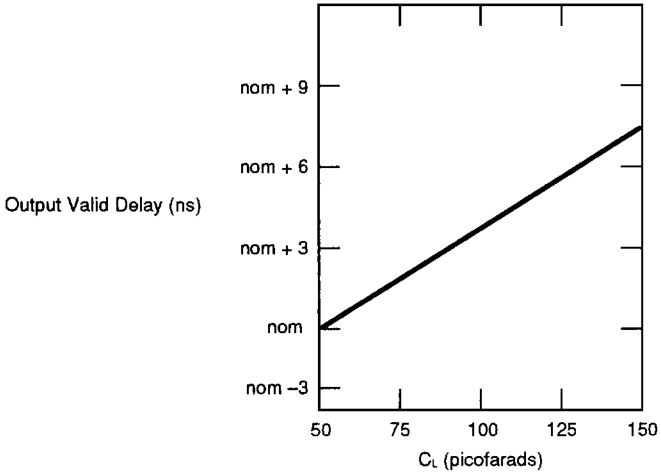
Figure 77. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (CL=120 pF)



Note: This graph will not be linear outside of the CL range shown.

15021B–080

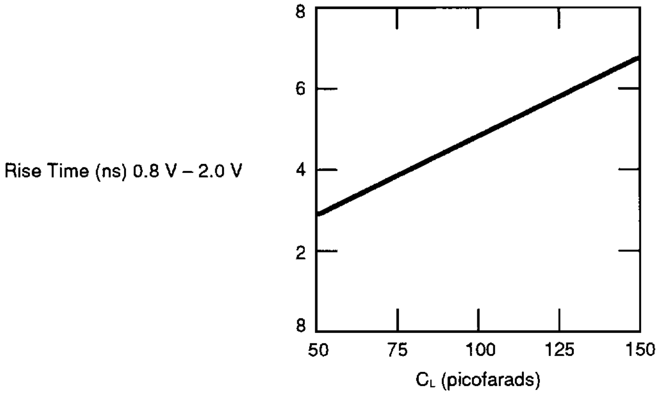
Figure 78. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (CL=75 pF)



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

15021B–081

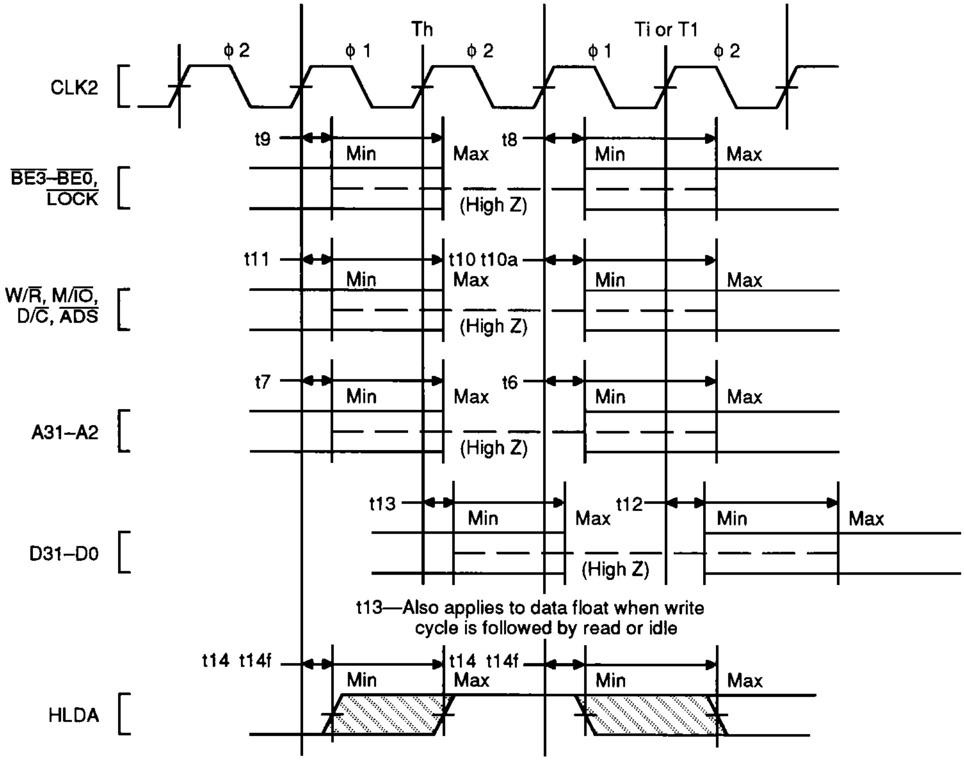
Figure 79. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L = 50 pF)



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

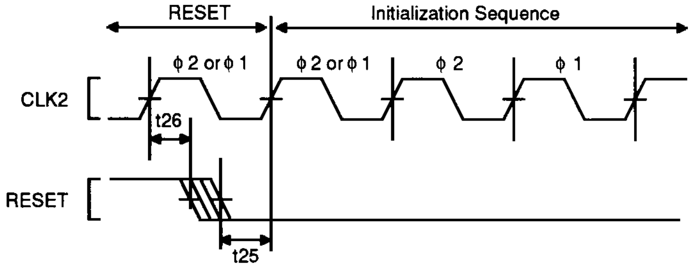
15021B–082

Figure 80. Typical Output Rise Time Versus Load Capacitance at Maximum Operating Temperature



15021B–083

Figure 81. Output Float Delay and HLDA Valid Delay Timing



The second internal processor phase following RESET High-to-Low transition (provided t25 and t26 are met) is ϕ2.

15021B–084

Figure 82. RESET Setup and Hold Timing and Internal Phase

INSTRUCTION SET

This section describes the Am386DX/DXL microprocessor instruction set. A table lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within Am386DX/DXL CPU instructions.

Am386DX/DXL Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 23, by the processor clock period (e.g., 50 ns for a 20-MHz, 40 ns for a 25-MHz, 30 ns for a 33-MHz, and 25 ns for a 40-MHz Am386DX/DXL microprocessor).

For more detailed information on the encodings of instructions refer to Section Instruction Encodings. Section Instruction Encodings explains the general structure of instruction encodings and defines exactly the encodings of all fields contained within the instruction.

Instruction Clock Count Assumptions

1. The instruction has been prefetched and decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No Exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling, and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. n = number of times repeated.
3. m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component; the entire immediate data (if any) counts as one component; and each of the other bytes of the instruction and prefix(es) each count as one component.

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary

Instruction	Format			Clock Count Virtual 8086 Mode	Clock Count		Comments	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
GENERAL DATA TRANSFER								
MOV = Move:								
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2/2	2/2	b	h
Register/Memory to Register	1 0 0 0 1 0 1 w	mod reg r/m			2/4	2/4	b	h
Immediate to Register/Memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	immediate data		2/2	2/2	b	h
Immediate to Register (short form)	1 0 1 1 w reg	immediate data			2	2		
Memory to Accumulator (short form)	1 0 1 0 0 0 0 w	full displacement			4	4	b	h
Accumulator to Memory (short form)	1 0 1 0 0 0 1 w	full displacement			2	2	b	h
Register/Memory to Segment Register	1 0 0 0 1 1 1 0	mod sreg3 r/m			2/5	18, 19	b	h, i, j
Segment Register to Register/Memory	1 0 0 0 1 1 0 0	mod reg r/m			2/2	2/2	b	h
MOVSX = Move with Sign Extension								
Register from Register/Memory	0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 w	mod reg r/m		3/6	3/6	b	h
MOVZX = Move with Zero Extension								
Register from Register/Memory	0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 w	mod reg r/m		3/6	3/6	b	h
PUSH = Push:								
Register/Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			5	5	b	h
Register (short form)	0 1 0 1 0 reg				2	2	b	h
Segment Register (ES,CS,SS, or DS)	0 0 0 sreg 2 1 1 0				2	2	b	h
Segment Register (FS or GS)	0 0 0 0 1 1 1 1	1 0 sreg 3 0 0 0			2	2	b	h
Immediate	0 1 1 0 1 0 s 0	immediate data			2	2	b	h
PUSHA = Push All	0 1 1 0 0 0 0 0				18	18	b	h
POP = Pop								
Register/Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			5	5	b	h
Register (short form)	0 1 0 1 1 reg				4	4	b	h
Segment Register (ES, SS, or DS)	0 0 0 sreg 2 1 1 1				7	21	b	h, i, j
Segment Register (FS or GS)	0 0 0 0 1 1 1 1	1 0 sreg 3 0 0 1			7	21	b	h, i, j
POPA = Pop All	0 1 1 0 0 0 0 1				24	24	b	h
XCHG = Exchange								
Register/Memory with Register	1 0 0 0 0 1 1 w	mod reg r/m			3/5	3/5	b, f	f, h
Register with Accumulator (short form)	1 0 0 1 0 reg				3	3		
IN = Input from:								
Fixed Port	1 1 1 0 0 1 0 w	port number		◊26	12	6*/26**		m
Variable Port	1 1 1 0 1 1 0 w			◊27	13	7*/27**		m
OUT = Output to:								
Fixed Port	1 1 1 0 0 1 1 w	port number		◊24	10	4*/24**		m
Variable Port	1 1 1 0 1 1 1 w			◊25	11	5*/25**		m
LEA = Load EA to Register	1 0 0 0 1 1 0 1	mod reg r/m			2	2		

* If CPL ≤ IOPL ** If CPL > IOPL

◊ Clock count shown applies if I/O permission allows I/O to the port in Virtual 8086 Mode. If I/O bit map denies permission, Exception 13 fault occurs; refer to clock counts for INT 3 instruction.

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
SEGMENT CONTROL							
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m		7	22	b	h, i, j
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m		7	22	b	h, i, j
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	7	25	b	h, i, j
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	7	25	b	h, i, j
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	7	22	b	h, i, j
FLAG CONTROL							
CLC = Clear Carry Flag	1 1 1 1 1 0 0 0			2	2		
CLD = Clear Direction Flag	1 1 1 1 1 1 0 0			2	2		
CLI = Clear Interrupt Enable Flag	1 1 1 1 1 0 1 0			8	8		m
CLTS = Clear Task Switched Flag	0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 0		6	6	c	i
CMC = Complement Carry Flag	1 1 1 1 0 1 0 1			2	2		
LAHF = Load AH into Flag	1 0 0 1 1 1 1 1			2	2		
POPF = Pop Flag	1 0 0 1 1 1 0 1			5	5	b	h,n
PUSHF = Push Flag	1 0 0 1 1 1 0 0			4	4	b	h
SAHF = Store AH into Flag	1 0 0 1 1 1 1 0			3	3		
STC = Set Carry Flag	1 1 1 1 1 0 0 1			2	2		
STD = Set Direction Flag	1 1 1 1 1 1 0 1			2	2		
STI = Set Interrupt Enable Flag	1 1 1 1 1 0 1 1			8	8		m
ARITHMETIC							
ADD=Add							
Register to Register	0 0 0 0 0 0 d w	mod reg r/m		2	2		
Register to Memory	0 0 0 0 0 0 0 w	mod reg r/m		7	7	b	h
Memory to Register	0 0 0 0 0 0 1 w	mod reg r/m		6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 0 0 0 1 0 w	immediate data		2	2		
ADC = Add with carry							
Register to Register	0 0 0 1 0 0 d w	mod reg r/m		2	2		
Register to Memory	0 0 0 1 0 0 0 w	mod reg r/m		7	7	b	h
Memory to Register	0 0 0 1 0 0 1 w	mod reg r/m		6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 0 1 0 1 0 w	immediate data		2	2		
INC =Increment							
Register/Memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m		2/6	2/6	b	h
Register (short form)	0 1 0 0 0 reg			2	2		
SUB =Subtract							
Register from Register	0 0 1 0 1 0 d w	mod reg r/m		2	2		
Register from Memory	0 0 1 0 1 0 0 w	mod reg r/m		7	7	b	h
Memory from Register	0 0 1 0 1 0 1 w	mod reg r/m		6	6	b	h

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (continued)							
Immediate from Register/Memory	0 0 1 0 0 1 1 w	mod 1 0 1 r/m	immediate data	2/7	2/7	b	h
Immediate from Accumulator (short form)	0 0 0 1 1 1 0 w	immediate data		2	2		
SBB = Subtract with Borrow							
Register from Register	0 0 0 1 1 0 d w	mod reg r/m		2	2		
Register from Memory	0 0 0 1 1 0 0 w	mod reg r/m		7	7	b	h
Memory from Register	0 0 0 1 1 0 1 w	mod reg r/m		6	6	b	h
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	immediate data	2/7	2/7	b	h
Immediate from Accumulator	0 0 0 1 1 1 0 w	immediate data		2	2		
DEC = Decrement							
Register/Memory	1 1 1 1 1 1 1 w	reg 0 0 1 r/m		2/6	2/6	b	h
Register (short form)	0 1 0 0 1 reg			2	2		
CMP = Compare							
Register with Register	0 0 1 1 1 0 d w	mod reg r/m		2	2		
Memory with Register	0 0 1 1 1 0 0 w	mod reg r/m		5	5	b	h
Register with Memory	0 0 1 1 1 0 1 w	mod reg r/m		6	6	b	h
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	immediate data	2/5	2/5	b	h
Immediate with Accumulator(short form)	0 0 1 1 1 1 0 w	immediate data		2	2		
NEG = Change Sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m		2/6	2/6	b	h
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1			4	4		
DAA = Decimal Adjust for Add	0 0 1 1 1 1 1 1			4	4		
AAS = ASCII Adjust for Subtract	0 0 1 0 0 1 1 1			4	4		
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1			4	4		
MUL = Multiply (Unsigned)							
Accumulator with Register/Memory	1 1 1 1 0 1 1 w	mod 1 0 0 r/m					
Multiplier -Byte				12-17/15-20	12-17/15-20	b,d	d,h
-Word				12-25/15-28	12-25/15-28	b,d	d,h
-Doubleword				12-41/15-44	12-41/15-44	b,d	d,h
IMUL = Integer Multiply (signed)							
Accumulator with Register/Memory	1 1 1 1 0 1 1 w	mod 1 0 1 r/m					
Multiplier -Byte				12-17/15-20	12-17/15-20	b,d	d,h
-Word				12-25/15-28	12-25/15-28	b,d	d,h
-Doubleword				12-41/15-44	12-41/15-44	b,d	d,h
Register with Register/Memory	0 0 0 0 1 1 1 1	1 0 1 0 1 1 1 1	mod reg r/m				
Multiplier -Byte				12-17/15-20	12-17/15-20	b,d	d,h
-Word				12-25/15-28	12-25/15-28	b,d	d,h
-Doubleword				12-41/15-44	12-41/15-44	b,d	d,h
Register/Memory with Immediate to Register	0 1 1 0 1 0 s 1	mod reg r/m	immediate data				
-Word				13-26/14-27	13-26/14-27	b,d	d,h
-Doubleword				13-42/14-43	13-42/14-43	b,d	d,h

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format				Clock Count		Comments	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (continued)								
DIV = Divide (Unsigned)								
Accumulator by Register/Memory	1 1 1 1 0 1 1 w	mod 1 1 0 r/m						
Divisor -Byte					14/17	14/17	b,e	e, h
-Word					22/25	22/25	b,e	e, h
-Doubleword					38/41	38/41	b,e	e, h
IDIV = Integer Divide (Signed)								
Accumulator by Register/Memory	1 1 1 1 0 1 1 2	mod 1 1 1 r/m						
Divisor -Byte					19/22	19/22	b,e	e, h
-Word					27/30	27/30	b,e	e, h
-Doubleword					43/46	43/46	b,e	e, h
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			19	19		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			17	17		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0				3	3		
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1				2	2		
LOGIC								
Shift/Rotate Instructions								
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)								
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			3/7	3/7	b	h
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			3/7	3/7	b	h
Register Memory by Immediate Count	1 1 0 0 0 0 0 w	mod TTT r/m	immediate 8-bit data		3/7	3/7	b	h
Through Carry (RCL and RCR)								
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			9/10	9/10	b	h
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			9/10	9/10	b	h
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w	mod TTT r/m	immediate 8-bit data		9/10	9/10	b	h
	TTT	**Instruction**						
	0 0 0	ROL						
	0 0 1	ROR						
	0 1 0	RCL						
	0 1 1	RCR						
	1 0 0	SHL/SAL						
	1 0 1	SHR						
	1 1 1	SAR						
SHLD = Shift Left Double								
Register/Memory by Immediate	0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 0	mod reg r/m	immediate 8-bit data	3/7	3/7		
Register/Memory by CL	0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 1	mod reg r/m		3/7	3/7		
SHRD = Shift Right Double								
Register/Memory by Immediate	0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 0	mod reg r/m	immediate 8-bit data	3/7	3/7		
Register/Memory by CL	0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 1	mod reg r/m		3/7	3/7		
AND = And								
Register to Register	0 0 1 0 0 0 d w	mod reg r/m			2	2		
Register to Memory	0 0 1 0 0 0 0 w	mod reg r/m			7	7	b	h
Memory to Register	0 0 1 0 0 0 1 w	mod reg r/m			6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	immediate data		2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 1 0 0 1 0 w	immediate data			2	2		
TEST = And Function to Flags, no Result								
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m			2/5	2/5	b	h
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	immediate data		2/5	2/5	b	h
Immediate Data and Accumulator (short form)	1 0 1 0 1 0 0 w	immediate data			2	2		

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format				Clock Count		Comments	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
LOGIC (continued)								
OR = Or								
Register to Register	0 0 0 0 1 0 d w	mod reg r/m			2	2		
Register to Memory	0 0 0 0 1 0 0 w	mod reg r/m			7	7	b	h
Memory to Register	0 0 0 0 1 0 1 w	mod reg r/m			6	6	b	h
Immediate and Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	immediate data		2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 0 0 1 1 0 w	immediate data			2	2		
XOR = Exclusive or								
Register to Register	0 0 1 1 0 0 d w	mod reg r/m			2	2		
Register to Memory	0 0 1 1 0 0 0 w	mod reg r/m			7	7	b	h
Memory to Register	0 0 1 1 0 0 1 w	mod reg r/m			6	6	b	h
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	immediate data		2/7	2/7	b	h
Immediate to Accumulator (short form)	0 0 1 1 0 1 0 w	immediate data			2	2		
NOT = Invert Register/Memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			2/6	2/6	b	h
STRING MANIPULATION				Clock Count Virtual 8086 Mode				
CMPS = Compare Byte/Word	1 0 1 0 0 1 1 w				10	10	b	h
INS = Input Byte/Wd from DX Port	0 1 1 0 1 1 0 w			◊29	15	9*/29**	b	h, m
LODS = Load Byte/Wd to AL/AX	1 0 1 0 1 1 0 w				5	5	b	h
MOVS = Move Byte/Word	1 0 1 0 0 1 0 w				8	8	b	h
OUTS = Output Byte/Wd to DX Port	0 1 1 0 1 1 1 w			◊28	14	8*/28**	b	h, m
SCAS = Scan Byte/Word	1 0 1 0 1 1 1 w				8	8	b	h
STOS = Store Byte/Word from AL/AX/EX	1 0 1 0 1 0 1 w				5	5	b	h
XLAT = Translate String	1 1 0 1 0 1 1 1				5	5		h
REPEATED STRING MANIPULATION Repeated by 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			5+9n	5+9n	b	h
REPNE CMPS = Compare String (Find Match)	1 1 1 1 0 0 1 0	1 0 1 0 0 1 1 w		Clock Count Virtual 8086 Mode	5+9n	5+9n	b	h
REP INS = Input String	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w		◊28 + 6n	14 + 6n	8 + 6n*/ 28 + 6n**	b	h, m
REP LODS = Load String	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			5 + 6n	5 + 6n	b	h
REP MOVS = Move String	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			8 + 4n	8 + 4n	b	h
REP OUTS = Output String	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w		◊26 + 5n	12 + 5n	6 + 5n*/ 26 + 5n**	b	h, m
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			5 + 8n	5 + 8n	b	h
REPNE SCAS = Store String (Find AL/AX/EAX)	1 1 1 1 0 0 1 0	1 0 1 0 1 1 1 w			5 + 8n	5 + 9n	b	h
REP STOS = Store String	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			5+5n	5+5n	b	h
BIT MANIPULATION								
BSF = Scan Bit Forward	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 0	mod reg r/m		11 + 3n	11 + 3n	b	h
BSR = Scan Bit Reverse	0 0 0 0 1 1 1 1	1 0 1 1 1 1 0 1	mod reg r/m		9 + 3n	9 + 3n	b	h

* If CPL ≤ IOPL   ** If CPL > IOPL

◊ Clock count shown applies if I/O permission allows I/O to the port in Virtual 8086 Mode. If I/O bit map denies permission, Exception 13 fault occurs; refer to clock counts for INT 3 instruction.

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format				Clock Count		Comments	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
BIT MANIPULATION (continued)								
BT = Test Bit								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 0 r/m	immediate 8-bit data	3/6	3/6	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 0 0 0 1 1	mod reg r/m		3/12	3/12	b	h
BTC = Test Bit and Complement								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 1 1 r/m	immediate 8-bit data	6/8	6/8	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 1	mod reg r/m		6/13	6/13	b	h
BTR = Test Bit and Reset								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 1 0 r/m	immediate 8-bit data	6/8	6/8	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 1	mod reg r/m		6/13	6/13	b	h
BTS = Test Bit and Set								
Register/Memory, Immediate	0 0 0 0 1 1 1 1	1 0 1 1 1 0 1 0	mod 1 0 1 r/m	immediate 8-bit data	6/8	6/8	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1	1 0 1 0 1 0 1 1	mod reg r/m		6/13	6/13	b	h
CONTROL TRANSFER								
CALL = Call								
Direct Within Segment	1 1 1 0 1 0 0 0	full displacement			7 + m	7 + m	b	r
Register/Memory Indirect Within Segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			7 + m 10 + m	7 + m 10 + m	b	h, r
Direct Intersegment	1 0 0 1 1 0 1 0	unsigned full offset, selector			17 + m	34 + m	b	j, k, r
Protected Mode Only (Direct Intersegment)								
Via Call Gate to Same Privilege Level						52 + m		h, j, k, r
Via Call Gate to Different Privilege Level, (No Parameters)						86 + m		h, j, k, r
Via Call Gate to Different Privilege Level, (x Parameters)						94+4x+m		h, j, k, r
From 80286 Task to 80286 TSS						273		h, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS						298		h, j, k, r
From 80286 Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)						218		h, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS						273		h, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS						300		h, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)						218		h, j, k, r
Indirect Intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m			22 + m	38 + m	b	h, j, k, r
Protected Mode Only (Indirect Intersegment)								
Via Call Gate to Same Privilege Level						56 + m		h, j, k, r
Via Call Gate to Different Privilege Level (No Parameters)						90 + m		h, j, k, r
Via Call Gate to Different Privilege Level (x Parameters)						98+4x+m		h, j, k, r
From 80286 Task to 80286 TSS						278		h, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS						303		h, j, k, r
From 80286 Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)						222		h, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS						278		h, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS						305		h, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)						222		h, j, k, r
JMP = Unconditional Jump								
Short	1 1 1 0 1 0 1 1	8-bit displacement			7 + m	7 + m		r
Direct within Segment	1 1 1 0 1 0 0 1	full displacement			7 + m	7 + m		r
Register/Memory Indirect within Segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m			7 + m 10 + m	7 + m 10 + m	b	h,r
Direct Intersegment	1 1 1 0 1 0 1 0	unsigned full offset, selector			12 + m	27 + m		j, k, r

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONTROL TRANSFER (continued)							
Protected Mode Only (Direct Intersegment)							
Via Call Gate to Same Privilege Level					45 + m		h, j, k, r
From 80286 Task to 80286 TSS					274		h, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS					301		h, j, k, r
From 80286 Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)					219		h, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS					270		h, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS					303		h, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)					221		h, j, k, r
Indirect Intersegment	11111111	mod 101 r/m		17 + m	31 + m	b	h, j, k, r
Protected Mode Only (Indirect Intersegment)							
Via Call Gate to Same Privilege Level					49 + m		h, j, k, r
From 80286 Task to 80286 TSS					279		h, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS					306		h, j, k, r
From 80286 Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)					223		h, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS					275		h, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS					308		h, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Task (Am386DX/DXL CPU TSS)					225		h, j, k, r
RET = Return from CALL							
Within Segment	11000011			10 + m	10 + m	b	g, h, r
Within Seg. Adding Immediate to SP	11000010	16-bit displacement		10 + m	10 + m	b	g, h, r
Intersegment	11001011			18 + m	32 + m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010	16-bit displacement		18 + m	32 + m	b	g, h, j, k, r
Protected Mode Only (RET) to Different Privilege Level							
Intersegment					69		h, j, k, r
Intersegment Adding Immediate to SP					69		h, j, k, r
CONDITIONAL JUMPS (Note: Times are Jump "Taken or Not Taken")							
JO = Jump on Overflow							
8-bit Displacement	01110000	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000000	full displacement	7+m or 3	7+m or 3		r
JNO = Jump on Not Overflow							
8-bit Displacement	01110001	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000001	full displacement	7+m or 3	7+m or 3		r
JB/JNAE = Jump on Below/Not Above or Equal							
8-bit Displacement	01110010	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000010	full displacement	7+m or 3	7+m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal							
8-bit Displacement	01110011	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000011	full displacement	7+m or 3	7+m or 3		r
JE/JZ = Jump on Equal/ Zero							
8-bit Displacement	01110100	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000100	full displacement	7+m or 3	7+m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero							
8-bit Displacement	01110101	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000101	full displacement	7+m or 3	7+m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above							
8-bit Displacement	01110110	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	00001111	10000110	full displacement	7+m or 3	7+m or 3		r

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONDITIONAL JUMPS (continued)							
JNBE/JA = Jump on Not Below or Equal/Above							
8-bit Displacement	0 1 1 1 0 1 1 1	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 1	full displacement	7+m or 3	7+m or 3		r
JS = Jump on Sign							
8-bit Displacement	0 1 1 1 1 0 0 0	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 0	full displacement	7+m or 3	7+m or 3		r
JNS = Jump on Not Sign							
8-bit Displacement	0 1 1 1 1 0 0 1	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 1	full displacement	7+m or 3	7+m or 3		r
JP/JPE = Jump on Parity/Parity Even							
8-bit Displacement	0 1 1 1 1 0 1 0	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 0 1 0	full displacement	7+m or 3	7+m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd							
8-bit Displacement	0 1 1 1 1 0 1 1	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 0 1 1	full displacement	7+m or 3	7+m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal							
8-bit Displacement	0 1 1 1 1 1 0 0	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 1 0 0	full displacement	7+m or 3	7+m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal							
8-bit Displacement	0 1 1 1 1 1 0 1	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 1 0 1	full displacement	7+m or 3	7+m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater							
8-bit Displacement	0 1 1 1 1 1 1 0	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 1 1 0	full displacement	7+m or 3	7+m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater							
8-bit Displacement	0 1 1 1 1 1 1 1	8-bit displacement		7+m or 3	7+m or 3		r
Full Displacement	0 0 0 0 1 1 1 1	1 0 0 0 1 1 1 1	full displacement	7+m or 3	7+m or 3		r
JCXZ = Jump on CX Zero *	1 1 1 0 0 0 1 1	8-bit displacement		9+m or 5	9+m or 5		r
JECXZ = Jump on ECX Zero *	1 1 1 0 0 0 1 1	8-bit displacement		9+m or 5	9+m or 5		r
LOOP = Loop CX Times	1 1 1 0 0 0 1 0	8-bit displacement		11 +m	11 +m		r
LOOPZ/LOOPE = Loop with Zero/Equal	1 1 1 0 0 0 0 1	8-bit displacement		11 +m	11 +m		r
LOOPNZ/LOOPNE = Loop while Not Zero	1 1 1 0 0 0 0 0	8-bit displacement		11 +m	11 +m		r
CONDITIONAL BYTE SET (Note: Times are Register/Memory)							
SETO = Set Byte on Overflow							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 0 0 0	mod 0 0 0 r/m	4/5	4/5		h
SETNO = Set Byte on Not Overflow							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 0 0 1	mod 0 0 0 r/m	4/5	4/5		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 0 1 0	mod 0 0 0 r/m	4/5	4/5		h

* Address Size Prefix Differentiates JCXZ from JECXZ.

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONDITIONAL BYTE SET (continued)							
SETNB = Set Byte on Not Below/Above or Equal							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 0 1 1	mod 0 0 0 r/m	4/5	4/5		h
SETE/SETZ = Set Byte on Equal/Zero							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 1 0 0	mod 0 0 0 r/m	4/5	4/5		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 1 0 1	mod 0 0 0 r/m	4/5	4/5		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 1 1 0	mod 0 0 0 r/m	4/5	4/5		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 0 1 1 1	mod 0 0 0 r/m	4/5	4/5		h
SETS = Set Byte on Sign							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 0 0 0	mod 0 0 0 r/m	4/5	4/5		h
SETNS = Set Byte on Not Sign							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 0 0 1	mod 0 0 0 r/m	4/5	4/5		h
SETP/SETPE = Set Byte on Parity/Parity Even							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 0 1 0	mod 0 0 0 r/m	4/5	4/5		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 0 1 1	mod 0 0 0 r/m	4/5	4/5		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 1 0 0	mod 0 0 0 r/m	4/5	4/5		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal							
To Register/Memory	0 0 0 0 1 1 1 1	0 1 1 1 1 1 0 1	mod 0 0 0 r/m	4/5	4/5		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 0	mod 0 0 0 r/m	4/5	4/5		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater							
To Register/Memory	0 0 0 0 1 1 1 1	1 0 0 1 1 1 1 1	mod 0 0 0 r/m	4/5	4/5		h
ENTER = Enter Procedure	1 1 0 0 1 0 0 0	16-bit displacement, 8-bit level					
L = 0				10	10	b	h
L = 1				12	12	b	h
L > 1				15+4(n–1)	15+4(n–1)	b	h
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1			4	4	b	h
INTERRUPT INSTRUCTIONS							
INT = Interrupt:							
Type Specified	1 1 0 0 1 1 0 1	type		37		b	
Type 3	1 1 0 0 1 1 0 0			33		b	
INTO = Interrupt 4 if Overflow Flag Set	1 1 0 0 1 1 1 0						
If OF = 1				35		b, e	
If OF = 0				3	3	b, e	

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format	Clock Count		Comments	
		Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS (continued)					
Bound = Interrupt 5 if Detect Value Out of Range	0 1 1 0 0 0 1 0 | mod reg r/m				
If Out of Range		44		b, e	e, g, h, j, k, r
If in Range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			282		g, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS via Task Gate			309		g, j, k, r
From 80286 Task to Virtual 8086 Mode via Task Gate			226		g, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS via Task Gate			284		g, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS via Task Gate			311		g, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Mode via Task Gate			228		g, j, k, r
From Virtual 8086 Mode to 80286 TSS via Task Gate			289		g, j, k, r
From Virtual 8086 Mode to Am386DX/DXL CPU TSS via Task Gate			316		g, j, k, r
From Virtual 8086 Mode to Privilege Level 0 via Trap Gate or Interrupt Gate			119		g, j, k, r
INT: Type 3					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			278		g, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS via Task Gate			305		g, j, k, r
From 80286 Task to Virtual 8086 Mode via Task Gate			222		g, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS via Task Gate			280		g, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS via Task Gate			307		g, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Mode via Task Gate			224		g, j, k, r
From Virtual 8086 Mode to 80286 TSS via Task Gate			285		g, j, k, r
From Virtual 8086 Mode to Am386DX/DXL CPU TSS via Task Gate			312		g, j, k, r
From Virtual 8086 Mode to Privilege Level 0 via Trap Gate or Interrupt Gate			119		g, j, k, r
INTO					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			280		g, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS via Task Gate			307		g, j, k, r
From 80286 Task to Virtual 8086 Mode via Task Gate			224		g, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS via Task Gate			282		g, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS via Task Gate			309		g, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Mode via Task Gate			225		g, j, k, r
From Virtual 8086 Mode to 80286 TSS via Task Gate			287		g, j, k, r
From Virtual 8086 Mode to Am386DX/DXL CPU TSS via Task Gate			314		g, j, k, r
From Virtual 8086 Mode to Privilege Level 0 via Trap Gate or Interrupt Gate			119		g, j, k, r
BOUND					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			254		g, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS via Task Gate			284		g, j, k, r
From 80286 Task to Virtual 8086 Mode via Task Gate			231		g, j, k, r
From Am386DX/DXL CPU Task to 80286 TSS via Task Gate			264		g, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS via Task Gate			294		g, j, k, r
From Am386DX/DXL CPU Task to Virtual 8086 Mode via Task Gate			243		g, j, k, r
From Virtual 8086 Mode to 80286 TSS via Task Gate			264		g, j, k, r
From Virtual 8086 Mode to Am386DX/DXL CPU TSS via Task Gate			294		g, j, k, r
From Virtual 8086 Mode to Privilege Level 0 via Trap Gate or Interrupt Gate			119		g, j, k, r
INTERRUPT RETURN					
IRET = Interrupt Return	1 1 0 0 1 1 1 1	22			g, h, j, k, r
Protected Mode Only (IRET)					
To the Same Privilege Level (within Task)			38		g, h, j, k, r
To Different Privilege Level (within Task)			82		g, h, j, k, r
From 80286 Task to 80286 TSS			232		h, j, k, r
From 80286 Task to Am386DX/DXL CPU TSS			265		h, j, k, r
From 80286 Task to Virtual 8086 Task			213		h, j, k, r
From 80286 Task to Virtual 8086 Mode (within Task)			60		
From Am386DX/DXL CPU Task to 80286 TSS			271		h, j, k, r
From Am386DX/DXL CPU Task to Am386DX/DXL CPU TSS		275		h, j, k, r	
From Am386DX/DXL CPU Task to Virtual 8086 Task		223		h, j, k, r	
From Am386DX/DXL CPU Task to Virtual 8086 Mode (within Task)			60		

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROCESSOR CONTROL							
HLT = HALT	11110100			5	5		l
MOV = Move to and From Control/Debug/Test Registers							
CR0/CR2/CR3 from register	00001111	00100010	1 1 eee reg	11/4/5	11/4/5		l
Register From CR3–0	00001111	00100000	1 1 eee reg	6	6		l
DR3–0 From Register	00001111	00100011	1 1 eee reg	22	22		l
DR7–6 From Register	00001111	00100011	1 1 eee reg	16	16		l
Register from DR7–6	00001111	00100001	1 1 eee reg	14	14		l
Register from DR3–0	00001111	00100001	1 1 eee reg	22	22		l
TR7–6 from Register	00001111	00100110	1 1 eee reg	12	12		l
Register from TR7–6	00001111	00100100	1 1 eee reg	12	12		l
NOP = No Operation	10010000			3	3		
WAIT = Wait until $\overline{\text{BUSY}}$ pin is negated	10011011			7	7		
NOP = No Operation	10010000			3	3		
PROCESSOR EXTENSION INSTRUCTIONS							
Processor Extension Escape	11011TTT	mod L L L r/m					h
	TTT and LLL bits are op-code information for coprocessor						
PREFIX BYTES							
Address Size Prefix	01100111			0	0		
LOCK = Bus Lock Prefix	11110000			0	0		m
Operand Size Prefix	01100110			0	0		
Segment Override Prefix							
CS:	00101110			0	0		
DS:	00111110			0	0		
ES:	00100110			0	0		
FS:	01100100			0	0		
GS:	01100101			0	0		
SS:	00110110			0	0		
PROTECTION CONTROL							
ARPL = Adjust Requested Privilege Level							
From Register/Memory	01100011	mod reg r/m		N/A	20/21	a	h
LAR = Load Access Rights							
From Register/Memory	00001111	00000010	mod reg r/m	N/A	15/16	a	g, h, j, p
LGDT = Load Global Descriptor							
Table Register	00001111	00000001	mod 0 1 0 r/m	11	11	b,c	h,l
LIDT = Load Interrupt Descriptor							
Table Register	00001111	00000001	mod 0 1 1 r/m	11	11	b, c	h, l
LLDT = Load Local Descriptor							
Table Register to Register/Memory	00001111	00000000	mod 0 1 0 r/m	N/A	20/24	a	g, h, j, l
LMSW = Load Machine Status Word							
From Register/Memory	00001111	00000001	mod 1 1 0 r/m	11/14	11/14	b.c	h, l

Table 23. Am386DX/DXL Microprocessor Instruction Set Summary (continued)

Instruction	Format			Clock Count		Comments	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROTECTION CONTROL (continued)							
LSL = Load Segment Limit							
From Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 1	mod reg r/m				
Byte-Granular Limit				N/A	21/22	a	g, h, j, p
Page-Granular Limit				N/A	25/26	a	g, h, j, p
LTR = Load Task Register							
From Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1 r/m	N/A	23/27	a	g, h, j, l
SGDT = Store Global Descriptor							
Table Register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 0 r/m	9	9	b, c	h
SIDT = Store Interrupt Descriptor							
Table Register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 1 r/m	9	9	b, c	h
SLDT = Store Local Descriptor Table Register							
To Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 0 r/m	N/A	2/2	a	h
SMSW = Store Machine Status Word	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 0 0 r/m	2/2	2/2	b, c	h, l
STR = Store Task Register							
To Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1 r/m	N/A	2/2	a	h
VERR = Verify Read Access							
Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 0 r/m	N/A	10/11	a	g, h, j, p
VERW = Verify Write Access	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 1 r/m	N/A	15/16	a	g, h, j, p

Instruction Notes for Table 23.

Notes a through c apply to Am386DX/DXL CPU Real Address Mode only.

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in Exception 6 (Invalid op-code).

b. Exception 13 fault (General Protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS limit, FFFFH. Exception 12 (fault stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to Am386DX/DXL CPU Real Address Mode and Am386DX/DXL CPU Protected Virtual Address Mode.

d. The Am386DX/DXL CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).
Clock counts given are minimum to maximum. To calculate actual clocks, use the following formula:
Actual Clock = if $m <> 0$ then $\max([\log_2 |m|], 3) + b$ clocks: if $m = 0$ then $3 + b$ clocks
In this formula, *m* is the multiplier, and
$b = 9$ for register to register,
$b = 12$ for memory to register,
$b = 10$ for register with immediate to register,
$b = 11$ for memory with immediate to register.

e. An Exception may occur, depending on the value of the operand.

f. $\overline{\text{LOCK}}$ is automatically asserted, regardless of the presence or absence of the $\overline{\text{LOCK}}$ prefix.

g. $\overline{\text{LOCK}}$ is asserted during descriptor table accesses.

Notes h through r apply to Am386DX/DXL CPU Protected Virtual Address Mode only.

h. Exception 13 fault (General Protection Violation) will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an Exception 12 (Stack Segment Limit Violation or Not Present) occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an Exception 13 fault (General Protection Violation). The segment's descriptor must indicate present or Exception 11 (CS, DS, ES, FS, GS Not Present). If the SS register is loaded and a stack segment not present is detected, an Exception 12 (Stack Segment Limit Violation or Not Present) occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an Exception 13 (General Protection Violation) if an applicable privilege rule is violated.

l. An Exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An Exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection Exception; rather, the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an Exception 13 fault (General Protection Exception) will occur before the ESC instruction is executed. An Exception 12 fault (Stack Segment Limit Violation or Not Present) will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an Exception 13 fault (General Protection Violation) will occur.

Instruction Encoding

Overview

All instruction encodings are subsets of the general instruction format shown in Figure 83. Instructions consist of one or two primary op-code 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 op-code or op-codes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary op-code 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 instruction.

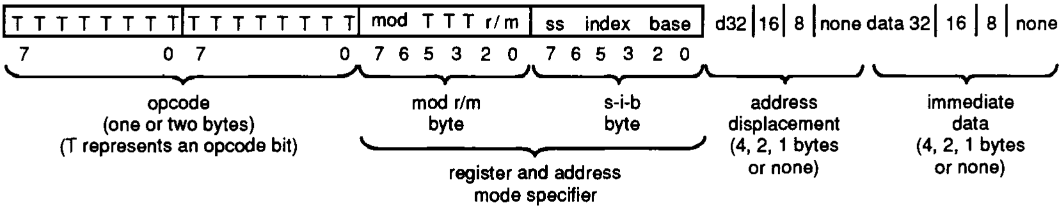
Figure 83 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the op-code bytes themselves. Table 24 is a complete list of all fields appearing in the Am386DX/DXL microprocessor instruction set. Further ahead, following Table 24, are detailed tables for each field.

32-Bit Extensions of the Instruction Set

With the Am386DX/DXL microprocessor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 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 Am386DX/DXL microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any op-code bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the op-code bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value opposite from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.



15021B–085

Figure 83. General Instruction Format

Table 24. Fields within Am386DX/DXL Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 bits)	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Condition Instructions, specifies a Condition Asserted or a Condition Negated	4

Note: Table 23 shows encoding of individual instructions.

These 32-bit extensions are available in all Am386DX/DXL microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8- and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

Encoding of 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.

Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32- or 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

Encoding of The General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary op-code bytes, or as the reg field of the mod r/m byte, or as the r/m field of the mod r/m byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
	Function of w Field	
reg	(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

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 Am386DX/DXL microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary op-code. The primary addressing byte is the mod r/m byte, and a second byte of addressing information, the scale-index-base (s-i-b) 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 83) sometimes used as an extension of the primary op-code. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the mod r/m byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the mod r/m byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following pages define all encodings of all 16- and 32-bit addressing modes.

Encoding of 16-Bit Address Mode with mod r/m Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	DS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	Register—See Below
11 001	Register—See Below
11 010	Register—See Below
11 011	Register—See Below
11 100	Register—See Below
11 101	Register—See Below
11 110	Register—See Below
11 111	Register—See Below

Register Specified by r/m During 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

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

Encoding of 32-Bit Address Mode with mod r/m byte
(No s-i-b Byte Present)

mod	r/m	Effective Address
00	000	DS:[EAX]
00	001	DS:[ECX]
00	010	DS:[EDX]
00	011	DS:[EBX]
00	100	s-i-b is present
00	101	DS:d32
00	110	DS:[ESI]
00	111	DS:[EDI]
01	000	DS:[EAX + d8]
01	001	DS:[ECX + d8]
01	010	DS:[EDX + d8]
01	011	DS:[EBX + d8]
01	100	s-i-b is present
01	101	SS:[EBP + d8]
01	110	DS:[ESI + d8]
01	111	DS:[EDI + d8]

mod	r/m	Effective Address
10	000	DS:[EAX + d32]
10	001	DS:[ECX + d32]
10	010	DS:[EDX + d32]
10	011	DS:[EBX + d32]
10	100	s-i-b is present
10	101	SS:[EBP + d32]
10	110	DS:[ESI + d32]
10	111	DS:[EDI + d32]
11	000	Register—See Below
11	001	Register—See Below
11	010	Register—See Below
11	011	Register—See Below
11	100	Register—See Below
11	101	Register—See Below
11	110	Register—See Below
11	111	Register—See Below

Register Specified by reg or r/m During 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

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

Encoding of 32-Bit Address Mode (mod r/m Byte and s-i-b present)

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

Note: Mod field in mod r/m byte; ss, index, base fields in s-i-b byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

Index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg (see note)
101	EBP
110	ESI
111	EDI

Note: When index field is 100, indicating no index register, then ss field must equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory ← Register reg Field indicates Source Operand; mod r/m or mod ss index base indicates Destination Operand.
1	Register ← Register Memory reg Field indicates Destination Operand; mod r/m or mod ss index base indicates Source Operand.

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- or 32-bit destination.

s	Effect on Immediate Data 8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extended Data 8 to fill 16-Bit or 32-Bit Destination	None

Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n=0) or its negation (n=1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Not Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding.	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding.	

When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding.	

MECHANICAL DATA

Introduction

In this section, the physical packaging and its connections are described in detail.

Package Dimensions and Mounting

The initial Am386DX/DXL microprocessor package is a 132-pin ceramic pin grid array (PGA). Pins of this package are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap.

Package Thermal Specification

The Am386DX/DXL microprocessor is specified for operation when ambient temperature is within the range of 0°C–100°C. The ambient temperature may be measured in any environment, to determine whether the Am386DX/DXL microprocessor is within specified operating range.

The PGA ambient temperature should be measured at the center of the top surface opposite the pins.

ELECTRICAL DATA

Introduction

The following sections describe recommended electrical connections for the Am386DX/DXL microprocessor and its electrical specifications.

Power and Grounding

Power Connections

The Am386DX/DXL CPU is implemented in CS21S technology and has modest power requirements. However, its high clock frequency and 72 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 20 V_{CC} and 21 V_{SS} pins separately feed functional units of the Am386DX/DXL CPU.

Power and ground connections must be made to all external V_{CC} and GND pins of the Am386DX/DXL CPU. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected to the GND plane.

Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the Am386DX/DXL CPU. The Am386DX/DXL microprocessor driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the Am386DX/DXL microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

Resistor Recommendations

The $\overline{\text{ERROR}}$, $\overline{\text{FLT}}$, and $\overline{\text{BUSY}}$ inputs have resistor pull-ups of approximately 20 Kohms built into the Am386DX/DXL CPU to keep these signals negated when no 387DX math coprocessor is present in the system (or temporarily removed from its socket). The $\overline{\text{BS16}}$ input also has an internal pull-up resistor of approximately 20 Kohms, and the PEREQ input has an internal pull-down resistor of approximately 20 Kohms.

In typical designs, the external pull-up resistors are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. NC pins should always remain unconnected.

Particularly when not using interrupts or bus hold, (as when first prototyping, perhaps) prevent any chance of spurious activity by connecting these associated inputs to GND.

Pin	Signal
B7	INTR
B8	NMI
D14	HOLD

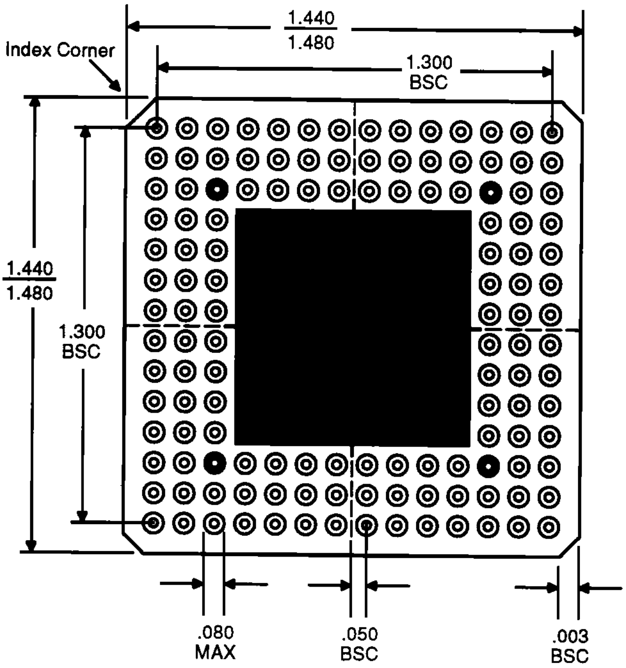
If not using address pipelining, pull-up D13 $\overline{\text{NA}}$ to V_{CC}.

If not using 16-bit size, pull-up C14 $\overline{\text{BS16}}$ to V_{CC}.

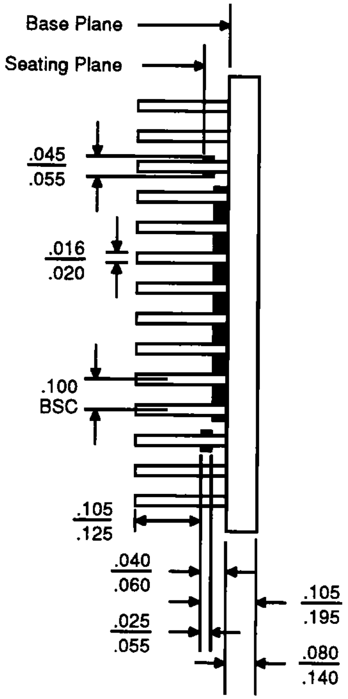
Pull-ups in the range of 20 Kohms are recommended.

PHYSICAL DIMENSIONS

CGX 132



Bottom View (Pins Facing Up)

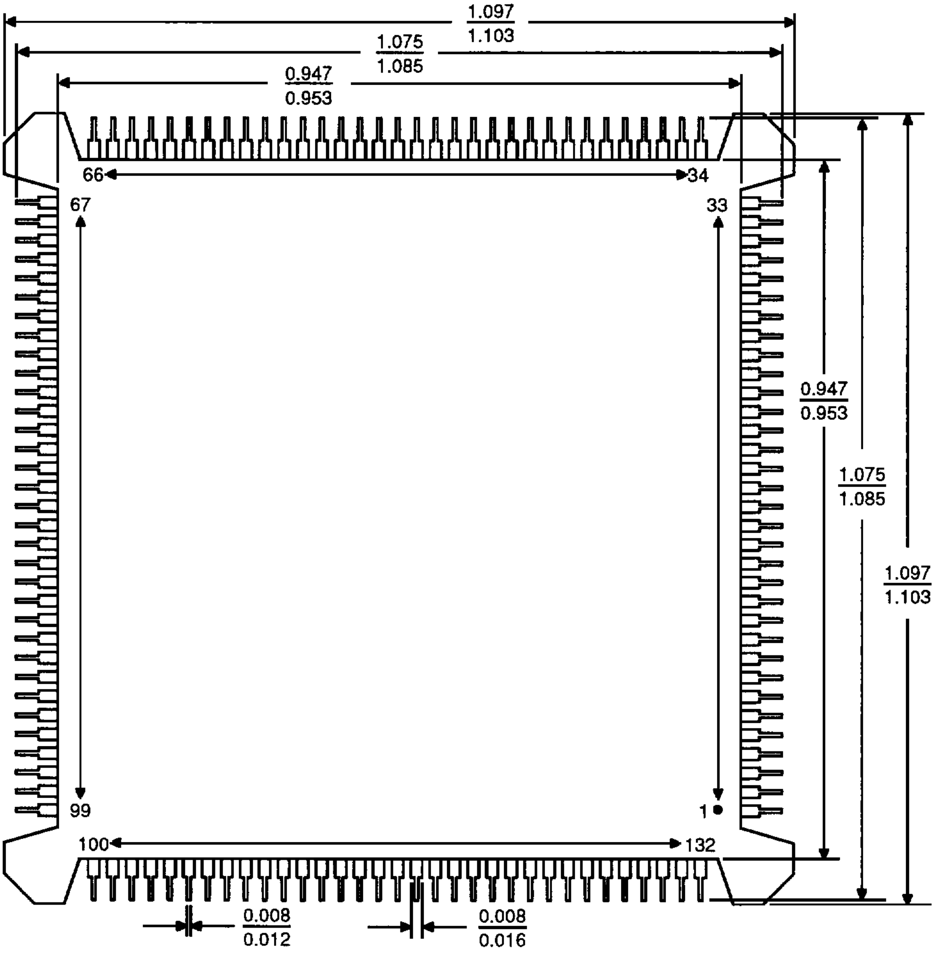


Side View

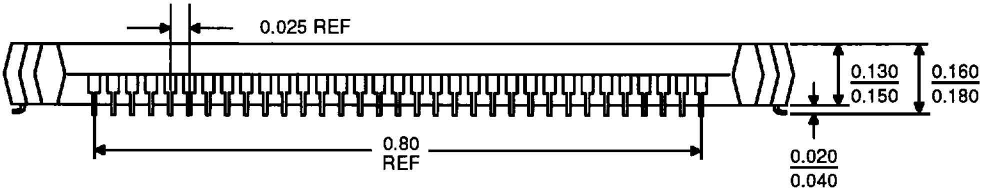
15304C
BP 36
4/30/91 CD

PHYSICAL DIMENSIONS (continued)

PQB 132—Plastic Quad Flat Pack (Trimmed and Formed)



Top View



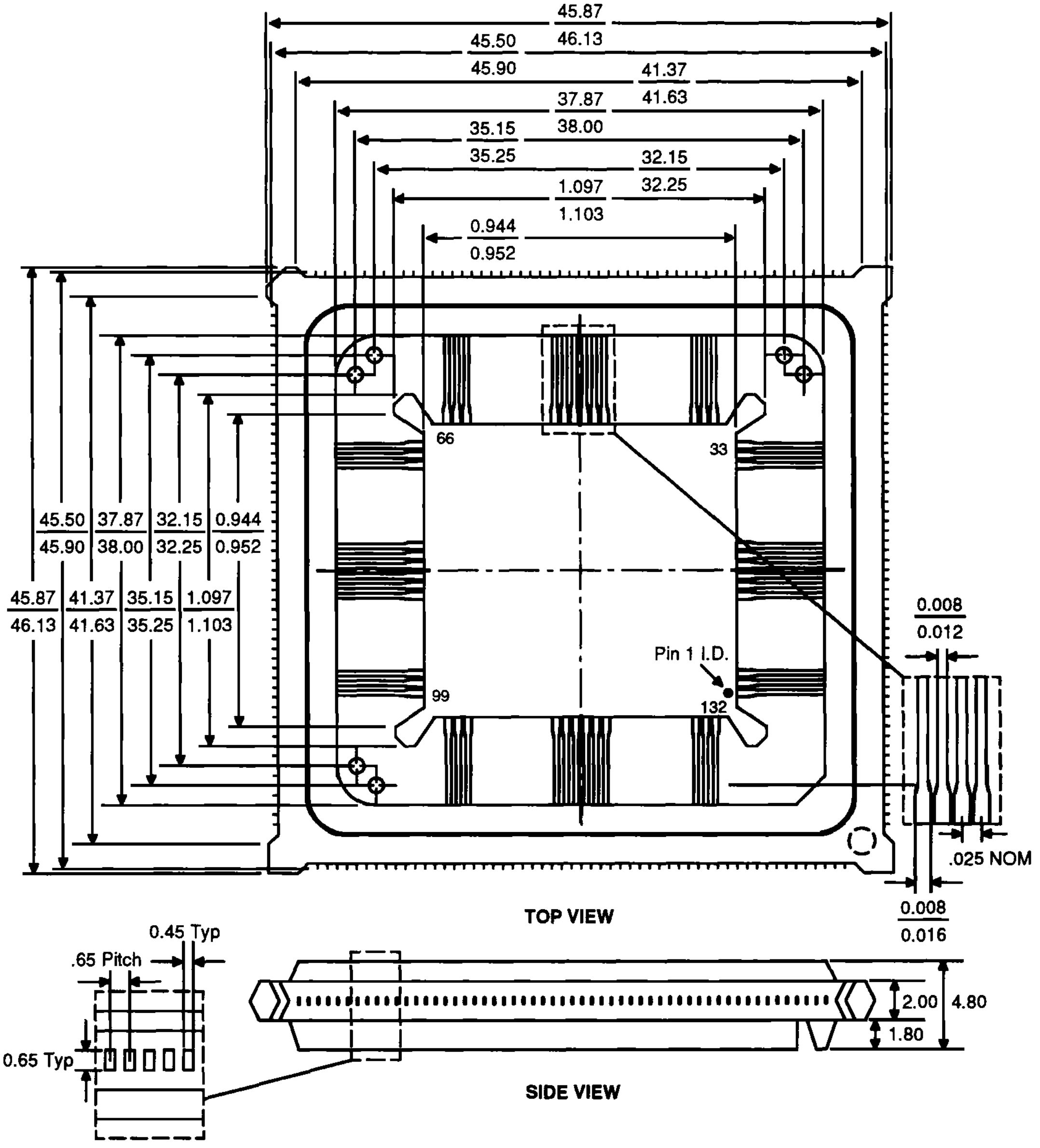
Side View

11772E
CF 43
7/28/92 SG

PHYSICAL DIMENSIONS (continued)

PQB 132—Plastic Quad Flat Pack with Molded Carrier Ring

(Outer Ring measured in millimeters)



TOP VIEW

SIDE VIEW

14826E
CB 51
7/28/92 SG