386SX™ MICROPROCESSOR

- **Full 32-Bit Internal Architecture**
  - **— 8-, 16-, 32-Bit Data Types**
  - **— 8 General Purpose 32-Bit Registers**
- **Runs Intel386™ Software in a Cost Effective 16-Bit Hardware Environment**
  - **— Runs Same Applications and O.S.'s as the 386™ Processor**
  - **— Object Code Compatible with 8086, 186, 286, and 386 Processors**
  - **— Runs MS-DOS*, OS/2* and UNIX****
- **Very High Performance 16-Bit Data Bus**
  - **— 16 MHz Clock**
  - **— Two-Clock Bus Cycles**
  - **— 16 Megabytes/Sec Bus Bandwidth**
  - **— Address Pipelining Allows Use of Slower/Cheaper Memories**
- **Integrated Memory Management Unit**
  - **— Virtual Memory Support**
  - **— Optional On-Chip Paging**
  - **— 4 Levels of Hardware Enforced Protection**
  - **— MMU Fully Compatible with Those of the 286 and 386 CPUs**
- **Virtual 8086 Mode Allows Execution of 8086 Software in a Protected and Paged System**
- **Large Uniform Address Space**
  - **— 16 Megabyte Physical**
  - **— 64 Terabyte Virtual**
  - **— 4 Gigabyte Maximum Segment Size**
- **High Speed Numerics Support with the 80387SX Coprocessor**
- **On-Chip Debugging Support Including Breakpoint Registers**
- **Complete System Development Support**
  - **— Software: C, PL/M, Assembler**
  - **— Debuggers: PMON-386, ICE™-386SX**
  - **— Extensive Third-Party Support: C, Pascal, FORTRAN, BASIC, Ada*** on VAX, UNIX**, MS-DOS*, and Other Hosts**
- **High Speed CHMOS III Technology**
- **100-Pin Plastic Quad Flatpack Package**
  (See Packaging Outlines and Dimensions #231369)

INTRODUCTION

The 386SX™ microprocessor is a 32-bit CPU with a 16-bit external data bus and a 24-bit external address bus. The 386SX CPU brings the high-performance software of the Intel386™ architecture to midrange systems. It provides the performance benefits of a 32-bit programming architecture with the cost savings associated with 16-bit hardware systems.

The 386SX microprocessor is 100% object code compatible with the 386, 286 and 8086 microprocessors. System manufacturers can provide 386 CPU based systems optimized for performance and 386SX CPU based systems optimized for cost, both sharing the same operating systems and application software. Systems based on the 386SX CPU can access the world's largest existing microcomputer software base, including the growing 32-bit software base. Only the Intel386 architecture can run UNIX, OS/2 and MS-DOS.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 386SX processor is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The integrated memory management unit (MMU) includes an address translation cache, advanced multi-tasking hardware, and a four-level hardware-enforced protection mechanism to support operating systems. The virtual machine capability of the 386SX CPU allows simultaneous execution of applications from multiple operating systems such as MS-DOS and UNIX.

The 386SX processor offers on-chip testability and debugging features. Four breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses for powerful debugging of even ROM-based systems. Other testability features include self-test, tri-state of output buffers, and direct access to the page translation cache.

*MS-DOS and OS/2 are trademarks of Microsoft Corporation.
**UNIX is a trademark of AT&T.
***Ada is a trademark of the Deptartment of Defense.

1.0 PIN DESCRIPTION

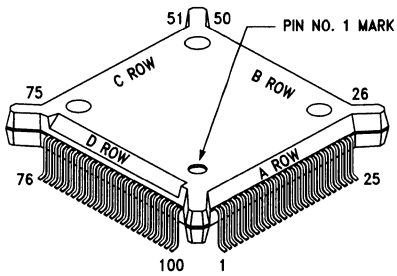


Figure 1.1. 80386SX Pin out Top View

Table 1.1. Pin Assignments

A Row		B Row		C Row		D Row	
Pin	**Label**	**Pin**	**Label**	**Pin**	**Label**	**Pin**	**Label**
1	D_0	26	LOCK#	51	A_2	76	A_{21}
2	V_{SS}	27	N/C	52	A_3	77	V_{SS}
3	HLDA	28	N/C	53	A_4	78	V_{SS}
4	HOLD	29	N/C	54	A_5	79	A_{22}
5	V_{SS}	30	N/C	55	A_6	80	A_{23}
6	NA#	31	N/C	56	A_7	81	D_{15}
7	READY#	32	V_{CC}	57	V_{CC}	82	D_{14}
8	V_{CC}	33	RESET	58	A_8	83	D_{13}
9	V_{CC}	34	BUSY#	59	A_9	84	V_{CC}
10	V_{CC}	35	V_{SS}	60	A_{10}	85	V_{SS}
11	V_{SS}	36	ERROR#	61	A_{11}	86	D_{12}
12	V_{SS}	37	PEREQ	62	A_{12}	87	D_{11}
13	V_{SS}	38	NMI	63	V_{SS}	88	D_{10}
14	V_{SS}	39	V_{CC}	64	A_{13}	89	D_9
15	CLK2	40	INTR	65	A_{14}	90	D_8
16	ADS#	41	V_{SS}	66	A_{15}	91	V_{CC}
17	BLE#	42	V_{CC}	67	V_{SS}	92	D_7
18	A_1	43	N/C	68	V_{SS}	93	D_6
19	BHE#	44	N/C	69	V_{CC}	94	D_5
20	N/C	45	N/C	70	A_{16}	95	D_4
21	V_{CC}	46	N/C	71	V_{CC}	96	D_3
22	V_{SS}	47	N/C	72	A_{17}	97	V_{CC}
23	M/IO#	48	V_{CC}	73	A_{18}	98	V_{SS}
24	D/C#	49	V_{SS}	74	A_{19}	99	D_2
25	W/R#	50	V_{SS}	75	A_{20}	100	D_1

1.0 PIN DESCRIPTION (Continued)

The following are the 80386SX pin descriptions. The following definitions are used in the pin descriptions:

The named signal is active LOW.
I Input signal.
O Output signal.
I/O Input and Output signal.
- No electrical connection.

Symbol	Type	Pin	Name and Function
CLK2	I	15	**CLK2** provides the fundamental timing for the 80386SX. For additional information see **Clock** (page 39).
RESET	I	33	**RESET** suspends any operation in progress and places the 80386SX in a known reset state. See **Interrupt Signals** (page 43) for additional information.
$D_{15}-D_0$	I/O	81-83,86-90, 92-96,99-100,1	**Data Bus** inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See **Data Bus** (page 39) for additional information.
$A_{23}-A_1$	O	80-79,76-72,70, 66-64,62-58, 56-51,18	**Address Bus** outputs physical memory or port I/O addresses. See **Address Bus** (page 40) for additional information.
W/R#	O	25	**Write/Read** is a bus cycle definition pin that distinguishes write cycles from read cycles. See **Bus Cycle Definition Signals** (page 40) for additional information.
D/C#	O	24	**Data/Control** is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and code fetch. See **Bus Cycle Definition Signals** (page 40) for additional information.
M/IO#	O	23	**Memory/IO** is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See **Bus Cycle Definition Signals** (page 40) for additional information.
LOCK#	O	26	**Bus Lock** is a bus cycle definition pin that indicates that other system bus masters are not to gain control of the system bus while it is active. See **Bus Cycle Definition Signals** (page 40) for additional information.
ADS#	O	16	**Address Status** indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and $A_{23}-A_1$ are being driven at the 80386SX pins. See **Bus Control Signals** (page 41) for additional information.
NA#	I	6	**Next Address** is used to request address pipelining. See **Bus Control Signals** (page 41) for additional information.
READY#	I	7	**Bus Ready** terminates the bus cycle. See **Bus Control Signals** (page 41) for additional information.
BHE#, BLE#	O	19,17	**Byte Enables** indicate which data bytes of the data bus take part in a bus cycle. See **Address Bus** (page 40) for additional information.

1.0 PIN DESCRIPTION (Continued)

Symbol	Type	Pin	Name and Function
HOLD	I	4	**Bus Hold Request** input allows another bus master to request control of the local bus. See **Bus Arbitration Signals** (page 41) for additional information.
HLDA	O	3	**Bus Hold Acknowledge** output indicates that the 80386SX has surrendered control of its local bus to another bus master. See **Bus Arbitration Signals** (page 41) for additional information.
INTR	I	40	**Interrupt Request** is a maskable input that signals the 80386SX to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** (page 43) for additional information.
NMI	I	38	**Non-Maskable Interrupt Request** is a non-maskable input that signals the 80386SX to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** (page 43) for additional information.
BUSY#	I	34	**Busy** signals a busy condition from a processor extension. See **Coprocessor Interface Signals** (page 42) for additional information.
ERROR#	I	36	**Error** signals an error condition from a processor extension. See **Coprocessor Interface Signals** (page 42) for additional information.
PEREQ	I	37	**Processor Extension Request** indicates that the processor has data to be transferred by the 80386SX. See **Coprocessor Interface Signals** (page 42) for additional information.
N/C	-	20, 27-31, 43-47	**No Connects** should always be left unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 80386SX.
V_{CC}	I	8-10,21,32,39 42,48,57,69, 71,84,91,97	**System Power** provides the +5V nominal DC supply input.
V_{SS}	I	2,5,11-14,22 35,41,49-50, 63,67-68, 77-78,85,98	**System Ground** provides the 0V connection from which all inputs and outputs are measured.

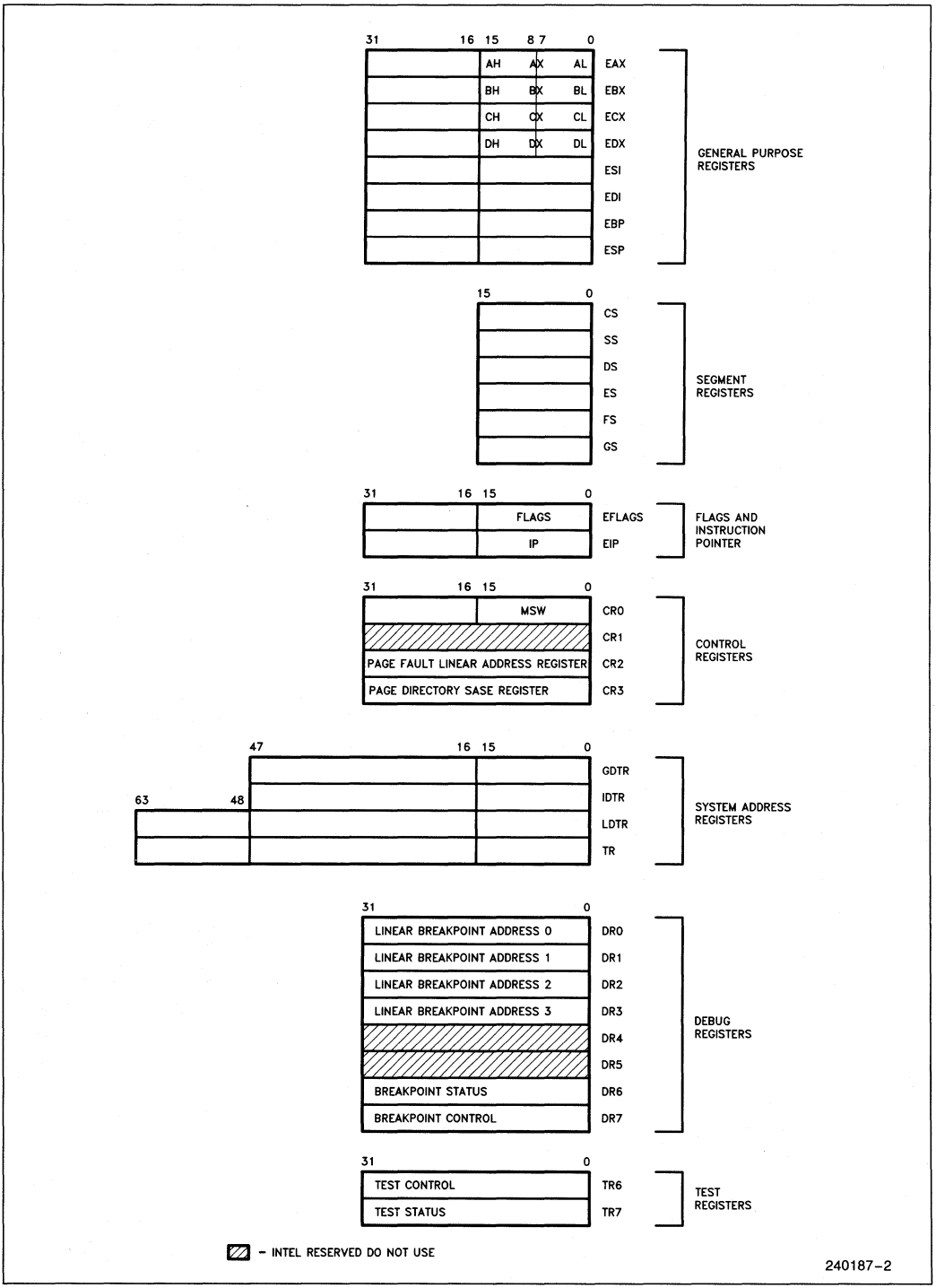


Figure 2.1. 80386SX Base Architecture Registers

2.0 BASE ARCHITECTURE

The 80386SX consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and the instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes 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.

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

The 80386SX has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the 80386SX operates as a very fast 8086, but with 32-bit extensions if desired. Real Mode is required primarily to set up the processor for Protected Mode operation.

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 80386SX operating system by use of paging.

Finally, to facilitate high performance system hardware designs, the 80386SX bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 80386SX has thirty-four registers as shown in Figure 2-1. These registers are grouped into the following seven categories:

General Purpose Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX, and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: The two 32-bit special purpose registers in figure 2.1 record or control certain aspects of the 80386SX processor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Registers: The four 32-bit control register are used to control the global nature of the 80386SX. The CR0 register contains bits that set the different processor modes (Protected, Real, Paging and Coprocessor Emulation). CR2 and CR3 registers are used in the paging operation.

System Address Registers: These four special registers reference the tables or segments supported by the 80286/80386SX/80386 protection model. These tables or segments are:

GDTR (Global Descriptor Table Register),
IDTR (Interrupt Descriptor Table Register),
LDTR (Local Descriptor Table Register),
TR (Task State Segment Register).

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.10 **Debugging Support**.

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Lookaside Buffer portion of the 80386SX. Their use is discussed in **Testability**.

EFLAGS REGISTER

The flag register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 80386SX. The lower 16 bits (bits 0–15) of EFLAGS contain the 16- bit flag register named FLAGS. This is the default flag register used when executing 8086, 80286, or real mode code. The functions of the flag bits are given in Table 2.1.

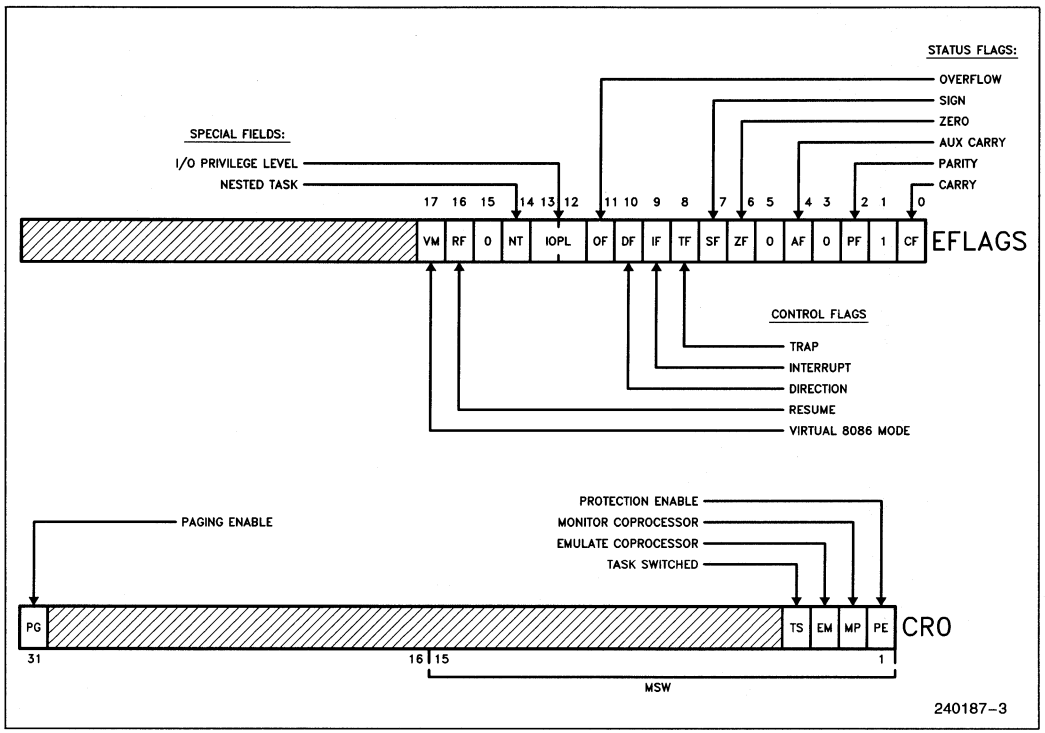


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	Auxiliary Carry Flag—Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	Zero Flag—Set if result is zero; cleared otherwise.
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.

Table 2.1. Flag Definitions (Continued)

Bit Position	Name	Function
10	DF	Direction Flag—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	Overflow Flag—Set if result is a too large positive number or a too small negative number (excluding sign- bit) to fit in destination operand; cleared otherwise.
12,13	IOPL	I/O Privilege Level—Indicates the maximum CPL permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map while executing in protected mode. For virtual 86 mode it indicates the maximum CPL allowing alteration of the IF bit.
14	NT	Nested Task—Indicates that the execution of the current task is nested within another task.
16	RF	Resume Flag—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction.
17	VM	Virtual 8086 Mode—If set while in protected mode, the 80386SX will switch to virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes.

CONTROL REGISTERS

The 80386SX has three control registers of 32 bits, CR0, CR2 and CR3, to hold the machine state of a global nature. These registers are shown in figures 2.1 and 2.2. The defined CR0 bits are described in table 2.2.

Table 2.2. CR0 Definitions

Bit Position	Name	Function
0	PE	Protection mode enable—places the 80386SX into 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 loading CR0, it cannot be reset by the LMSW instruction.
1	MP	Monitor coprocessor extension—allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension—causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched—indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.
31	PG	Paging enable bit—is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

2.2 Instruction Set

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 80386SX instructions are listed in Table 8.1 **80386SX Instruction Set and Clock Count Summary**.

All 80386SX instructions operate on either 0, 1, 2 or 3 operands; 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 80386SX has a 16 byte prefetch 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
Memory to Memory
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 80386SX (32 bit code), operands are 8 or 32 bits; when executing existing 8086 or 80286 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Memory on the 80386SX is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest 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 80386SX 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 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 80386SX supports both pages and segmentation 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 to the system programmer for managing the physical memory of a system.

ADDRESS SPACES

The 80386SX has three types of address spaces: **logical**, **linear**, and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT), discussed in section 2.4 **Addressing Modes**, into an effective address. This effective address along with the selector is known as the logical address. Since each task on the 80386SX has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes (with paging enabled) this gives a total of 2^{46} bits, or 64 terabytes, of **logical** address space per task. The programmer sees the logical 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 is truncated into a 24-bit **physical** address. The **physical address** is what appears on the address pins.

The primary differences between Real Mode and Protected Mode are how the segmentation unit performs the translation of the **logical** address into the **linear** address, size of the address space, and paging capability. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the effective address to form the **linear** address. This **linear** address is limited to 1 megabyte. In addition, real mode has no paging capability.

Protected Mode will see one of two different address spaces, depending on whether or not paging is enabled. Every selector has a **logical base** address associated with it that can be up to 32 bits in length. This 32-bit **logical base** address is added to the effective address to form a final 32-bit **linear**

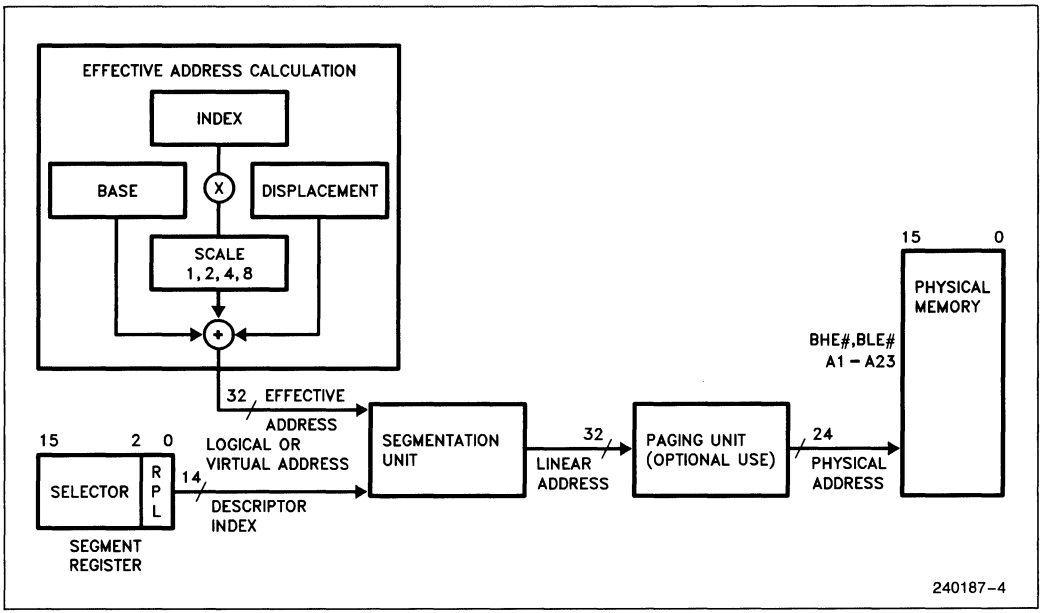


Figure 2.3. Address Translation

address. If paging is disabled this final **linear** address reflects physical memory and is truncated so that only the lower 24 bits of this address are used to address the 16 megabyte memory address space. If paging is enabled this final **linear** address reflects a 32-bit address that is translated through the paging unit to form a 16-megabyte physical address. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 80386SX, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bits).

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. The segment register is automatically chosen according to the rules of Table 2.3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register, stack references use the SS register and instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.3. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in chapter 4 **PROTECTED MODE ARCHITECTURE**.

2.4 Addressing Modes

The 80386SX provides a total of 8 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:

Table 2.3. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHA instructons	SS	None
Source of POP, POPA instructions	SS	None
Destination of STOS, MOVE, REP STOS, and REP MOVS instructions	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,DS,ES,FS,GS
[ESP]	SS	CS,DS,ES,FS,GS

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

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

32-BIT MEMORY ADDRESSING MODES

The remaining 6 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 summing any combination of the following three address elements (see figure 2.3):

DISPLACEMENT: an 8, 16 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. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures.

Combinations of these 3 components make up the 6 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.4, the effective address (EA) of an operand is calculated according to the following formula:

$$EA = Base_{Register} + (Index_{Register} * scaling) + Displacement$$

1. **Direct Mode:** The operand's offset is contained as part of the instruction as an 8, 16 or 32-bit displacement.
2. **Register Indirect Mode:** A BASE register contains the address of the operand.
3. **Based Mode:** A BASE register's contents are added to a DISPLACEMENT to form the operand's offset.
4. **Scaled Index Mode:** An INDEX register's contents are multiplied by a SCALING factor, and the result is added to a DISPLACEMENT to form the operand's offset.

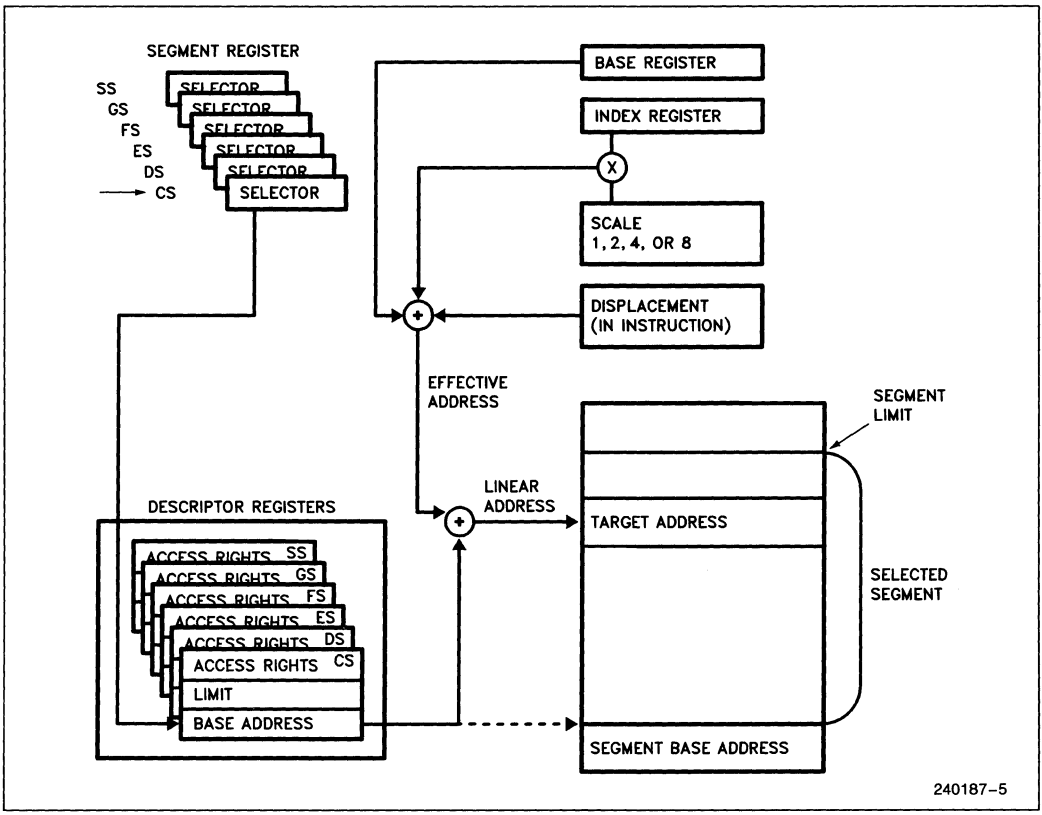


Figure 2.4. Addressing Mode Calculations

5. **Based Scaled Index Mode:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register to obtain the operand's offset.
6. **Based Scaled Index Mode with Displacement:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

DIFFERENCES BETWEEN 16 AND 32 BIT ADDRESSES

In order to provide software compatibility with the 8086 and the 80286, the 80386SX 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 a 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 80386SX is able to execute either 16 or 32-bit instructions. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by assemblers.

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 64K bytes to be accessed in Real Mode. A memory address which exceeds 0FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 80386SX addressing modes.

When executing 32-bit code, the 80386SX 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 286 model. Table 2.4 illustrates the differences.

Table 2.4. BASE and INDEX Registers for 16- and 32-Bit Addresses

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

2.5 Data Types

The 80386SX supports all of the data types commonly used in high level languages:

Bit: A single bit quantity.

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

Bit String: A set of contiguous bits; on the 80386SX, bit strings can be up to 4 gigabits 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.

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

Long 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 gigabytes

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 80386SX is coupled with its numerics coprocessor, the 80387SX, 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 the 80387SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 80386SX and the 80387SX.

2.6 I/O Space

The 80386SX has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 80386SX also supports memory-mapped peripherals. The I/O space consists of 64K bytes which can be divided into 64K 8- bit ports or 32K 16-bit ports, or any combination of ports which add up to no more than 64K bytes. The 64K I/O address space refers to physical addresses rather than linear addresses since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed by the IN and OUT instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven LOW. I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

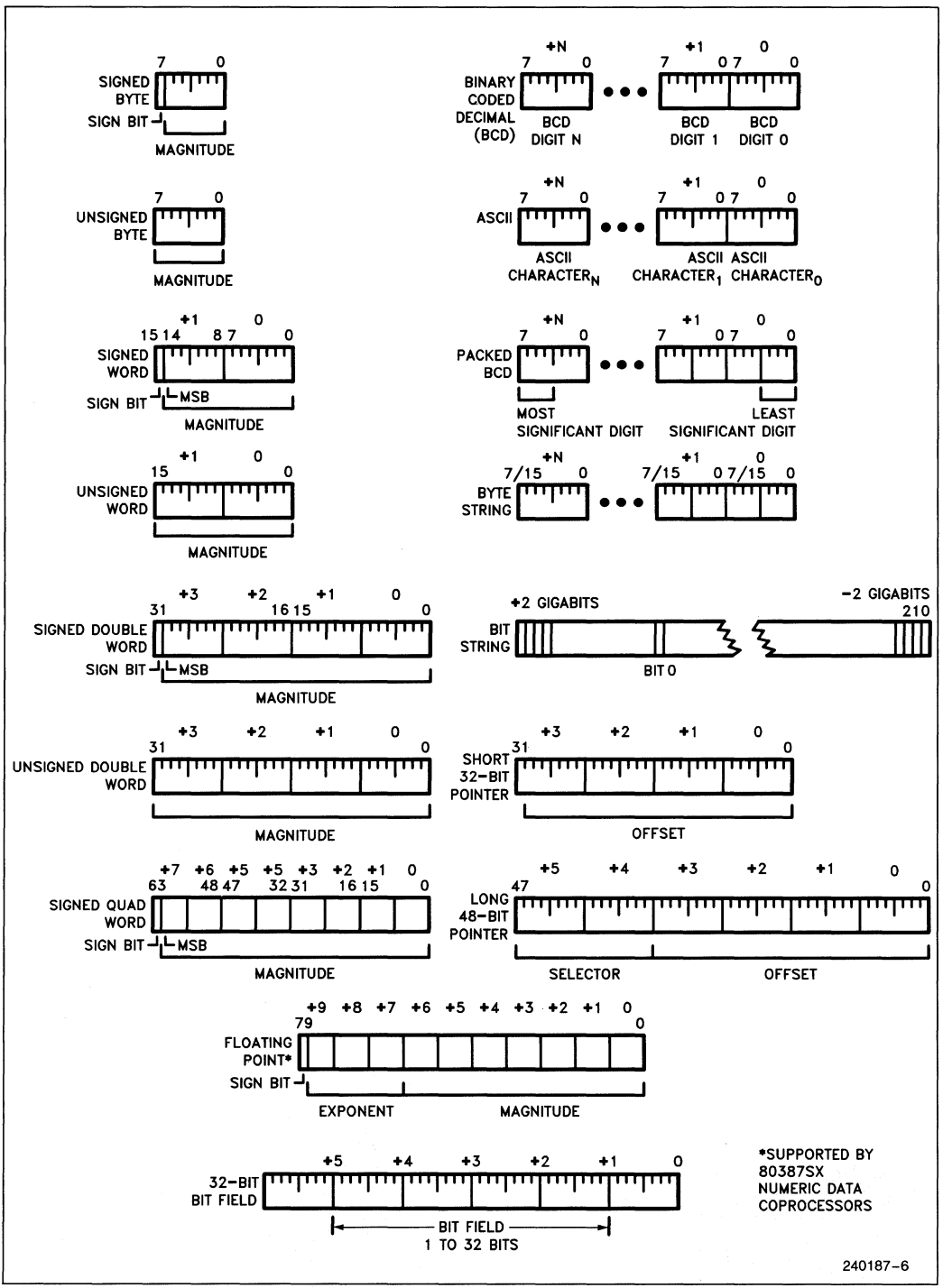


Figure 2.5. 80386SX Supported Data Types

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any illegal instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any instruction that can generate an exception		ABORT
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
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–32			
Two Byte Interrupt	0–255	INT n	NO	TRAP

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

2.7 Interrupts and Exceptions

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

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

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. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined.

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 to the instruction causing the exception and will include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 80386SX and shows where the return address points to.

The 80386SX 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, the vectors are 4-byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs, the following actions happen. First, the current program address and Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 80386SX which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 80386SX 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 to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled HIGH and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an 'interrupt window' between memory moves which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled HIGH it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 80386SX will not service any further NMI request or INT 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 80386SX is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the n^{th} vector in the interrupt table.

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

A final type of software interrupt is the single step interrupt. It is discussed in **Single Step Trap** (page 20).

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 80386SX invokes the NMI service routine first. If maskable interrupts are still enabled after the NMI service routine has been invoked, then the 80386SX will invoke the appropriate interrupt service routine.

As the 80386SX executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 80386SX fully supports restarting all instructions after Faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.6), the 80386SX invokes the appropriate exception service routine. The 80386SX is in a state that permits restart of the instruction, for all cases but those given in Table 2.7. Note that all such cases will be avoided by a properly designed operating system.

Table 2.6. Sequence of Exception Checking

Consider the case of the 80386SX 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 external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only; or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e. not at IOPL or at CPL = 0).
7. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
8. If ESCape opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
9. If WAIT opcode or ESCape opcode for numeric coprocessor, check ERROR# input signal (exception 16 if 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:
Segmentation exceptions are generated before paging exceptions.

Table 2.7. Conditions Preventing Instruction Restart

1. An instruction causes a task switch to a task whose Task State Segment is **partially** 'not present' (An entirely 'not present' TSS is restartable). Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4K bytes or less).
2. A coprocessor operand wraps around the top of a 64K-byte segment or a 4G-byte 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 **at a page boundary** any segments containing coprocessor operands if the segments are approximately 64K-200 bytes 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.

Table 2.8. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	Note 1
Machine Status Word (CR0)	uuuuuuu0H	Note 2
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	Note 3
Data Segment (DS)	0000H	Note 4
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	Note 4
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX register	0000H	Note 5
EDX register	component and stepping ID	Note 6
All other registers	undefined	Note 7

NOTES:
1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. CR0: All of the defined fields in CRO are 0.
3. The Code Segment Register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
4. The Data and Extra Segment Registers (DS, ES) will have their Base Address set to 000000000H and Limit set to 0FFFFH.
5. If self-test is selected, the EAX register should contain a 0 value. If a value of 0 is not found then the self-test has detected a flaw in the part.
6. EDX register always holds component and stepping identifier.
7. All undefined bits are Intel Reserved and should not be used.

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).

One other cause of generating a Double Fault is the 80386SX detecting any other exception when it is attempting to invoke the Page Fault (exception 14) service routine (for example, if a Page Fault is detected when the 80386SX attempts to invoke the Page Fault service routine). Of course, in any functional system, not only in 80386SX-based systems, the entire page fault service routine must remain 'present' in memory.

2.8 Reset and Initialization

When the processor is initialized or Reset the registers have the values shown in Table 2.8. The 80386SX will then start executing instructions near the top of physical memory, at location 0FFFFF0H. When the first Intersegment Jump or Call is executed, address lines A_{20}–A_{23} will drop LOW for CS-relative memory cycles, and the 80386SX will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a shadow ROM at the top of physical memory to initialize the system and take care of Resets.

RESET forces the 80386SX 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 80386SX will start executing instructions at the top of physical memory.

2.9 Testability

The 80386SX, like the 80386, offers testability features which include a self-test and direct access to the page translation cache.

SELF-TEST

The 80386SX has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 80386SX can be tested during self-test.

Self-Test is initiated on the 80386SX when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is LOW. The self-test takes about 2^{20} clocks, or approximately 33 milliseconds with a 16 MHz 80386SX. 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 are zero. If the results of the EAX are not zero then the self-test has detected a flaw in the part.

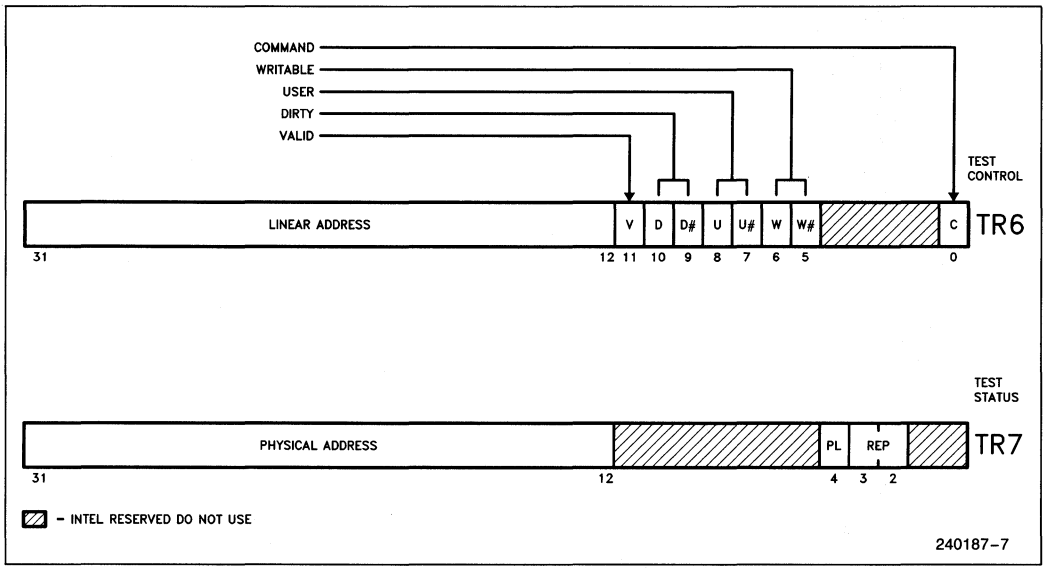


Figure 2.6. Test Registers

TLB TESTING

The 80386SX also provides a mechanism for testing the Translation Lookaside Buffer (TLB) if desired. This particular mechanism may not be continued in the same way in future processors.

There are two TLB testing operations: 1) writing entries into the TLB, and, 2) performing TLB lookups. Two Test Registers, shown in Figure 2.6, are provided for the purpose of testing. TR6 is the "test command register", and TR7 is the "test data register". For a more detailed explanation of testing the TLB, see the 80386 Programmer's Reference Manual.

2.10 Debugging Support

The 80386SX provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

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

BREAKPOINT INSTRUCTION

A single-byte software interrupt (Int 3) breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed.

SINGLE-STEP TRAP

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1.

DEBUG REGISTERS

The Debug Registers are an advanced debugging feature of the 80386SX. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 80386SX contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.7 shows the breakpoint status and control registers.

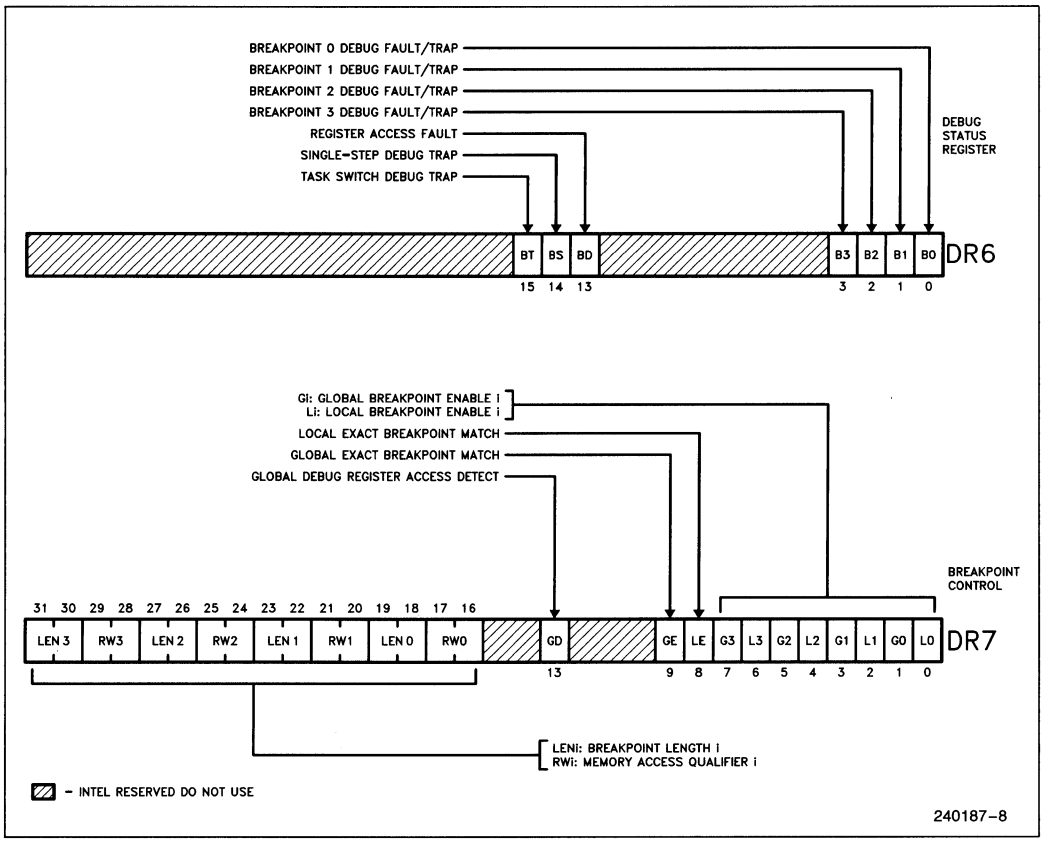


Figure 2.7. Debug Registers

3.0 REAL MODE ARCHITECTURE

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 80386SX. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286.

The default operand size in Real Mode is 16 bits, as in 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 80386SX in Real Mode is 64K bytes so 32-bit addresses must have a value less then 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode operation.

3.1 Memory Addressing

In Real Mode the linear addresses are the same as physical addresses (paging is not allowed). Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a 20-bit physical address or a 1 megabyte address space. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 80386SX will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment.

Table 3.1. 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 with offset = 0FFFFH. an attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = 0FFFFH	Before Instruction

3.2 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: the 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 0FFFFF0H through 0FFFFFFH are reserved for system initialization.

3.3 Interrupts

Many of the exceptions discussed in section 2.7 are not applicable to Real Mode operation; in particular, exceptions 10, 11 and 14 do not occur in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.4 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 80386SX 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:

1. An interrupt or an exception occurs (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table.
2. A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

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 000FH) and the stack has enough room to contain the vector and flag information (i.e. SP is greater that 0005H). Otherwise, shutdown can only be exited by a processor reset.

3.5 LOCK operation

The LOCK prefix on the 80386SX, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 80386SX in Protected Mode and Virtual 8086 Mode. The LOCK prefix is not supported during repeat string instructions.

The only instruction forms where the LOCK prefix is legal on the 80386SX are shown in Table 3.2.

Table 3.2. Legal Instructions for the LOCK Prefix

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

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above.

The LOCK prefix is not IOPL-sensitive on the 80386SX. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed in Table 3.2.

4.0 PROTECTED MODE ARCHITECTURE

The complete capabilities of the 80386SX are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes (2^{46} bytes)). In addition, Protected Mode allows the 80386SX to run all of the existing 80386 (using only 16 megabytes of physical memory), 80286 and 8086 software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions specially optimized for supporting multitasking operating systems. The base architecture of the 80386SX remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's viewpoint is the increased address space and a different addressing mechanism.

4.1 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 a 24-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 24-bit physical address.

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

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 80386SX, as such paging operates beneath segmentation. The page mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 80386SX addressing mechanism with paging enabled.

4.2 Segmentation

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

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.

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

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

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

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and the DPL. 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

The descriptor tables define all of the segments which are used in a 80386SX system. There are three types of tables on the 80386SX which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays and can vary in size from 8 bytes to 64K bytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address and the 16-bit limit of each table.

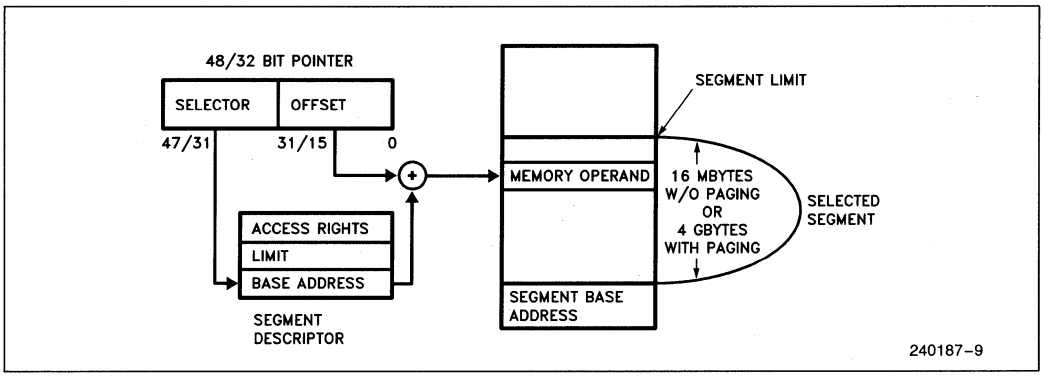


Figure 4.1. Protected Mode Addressing

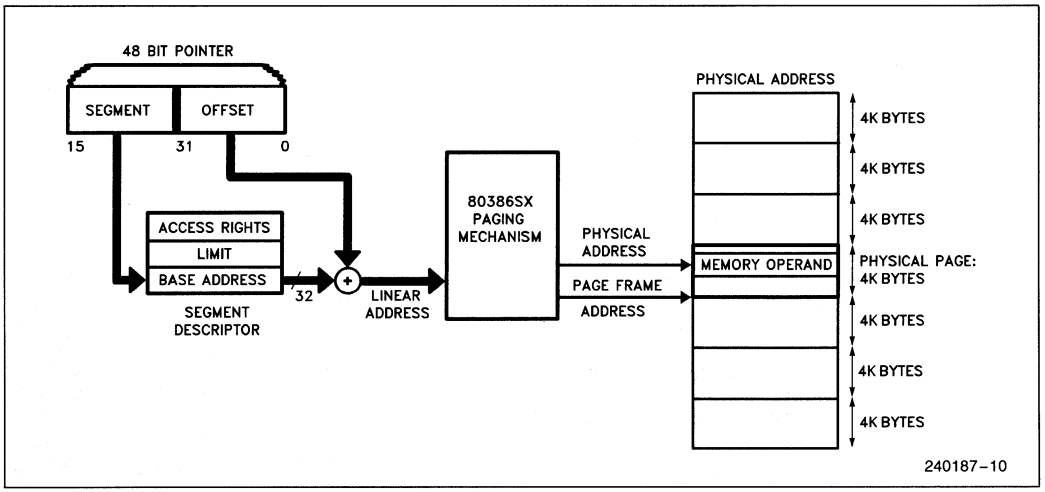


Figure 4.2. Paging and Segmentation

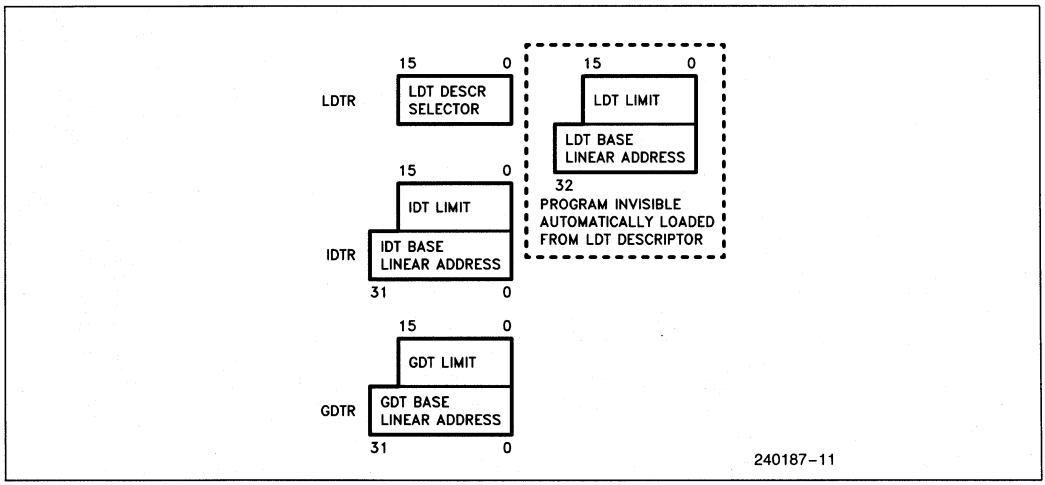


Figure 4.3. Descriptor Table Registers

Each of the tables has a register associated with it: GDTR, LDTR, and IDTR; see Figure 2.1. The LGDT, LLDT, and LIDT instructions load the base and limit of the Global, Local, and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These are privileged instructions.

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every 80386SX system contains a GDT.

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.

Local Descriptor Table

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

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see figure 2.1).

Interrupt Descriptor Table

The third table needed for 80386SX systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of the up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.4 shows the general format of a descriptor. All segments on the 80386SX have three attribute fields in common: the P bit, the DPL bit, and the S bit. The P (Present) Bit is 1 if the

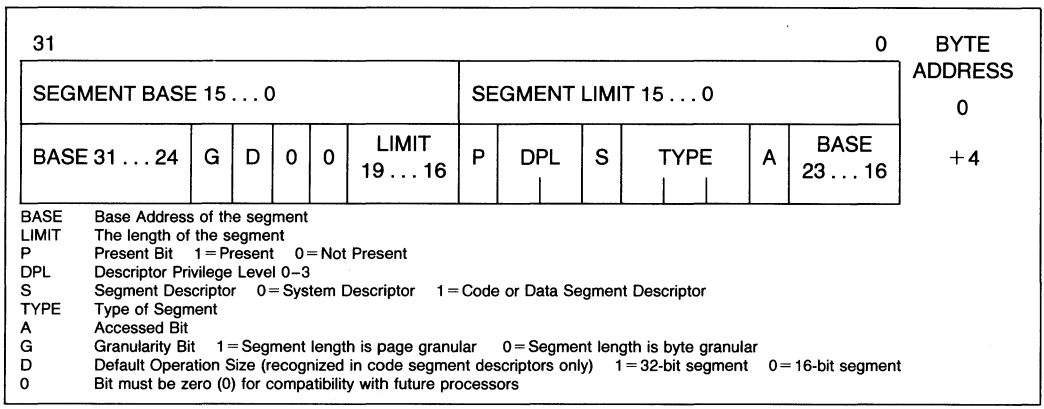


Figure 4.4. Segment Descriptors

segment is loaded in physical memory. If P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level, DPL, is a two bit field which specifies the protection level, 0–3, associated with a segment.

The 80386SX has two main categories of segments: system segments and non-system segments (for code and data). The segment bit, S, 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.

Code and Data Descriptors (S=1)

Figure 4.5 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Right Byte are interpreted.

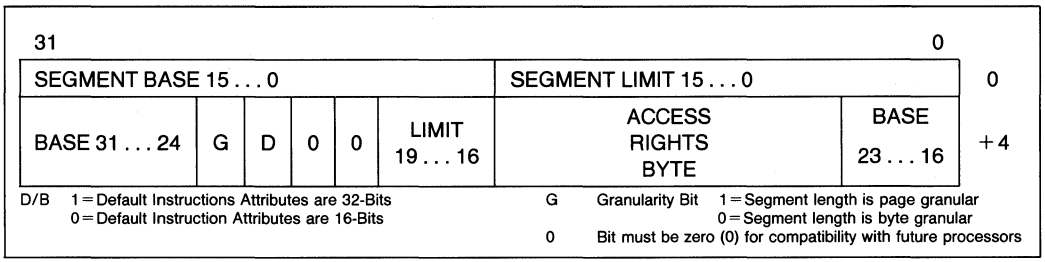
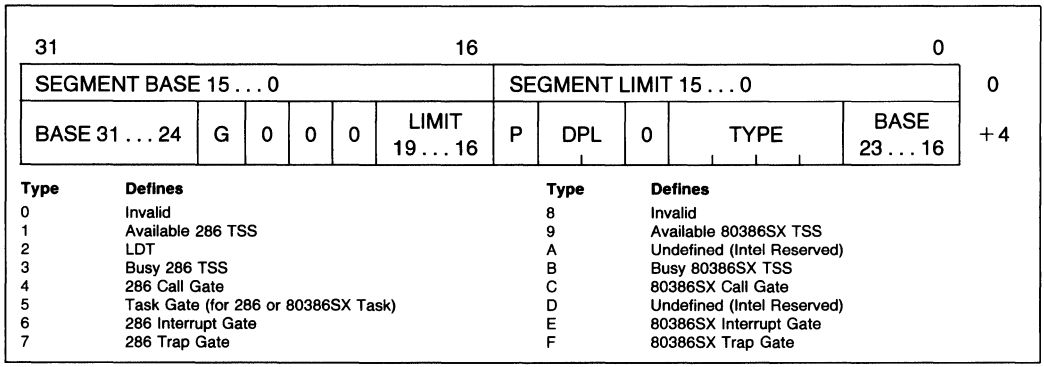


Figure 4.5. Code and Data Descriptors

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

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



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 286 TSS	9	Available 80386SX TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 286 TSS	B	Busy 80386SX TSS
4	286 Call Gate	C	80386SX Call Gate
5	Task Gate (for 286 or 80386SX Task)	D	Undefined (Intel Reserved)
6	286 Interrupt Gate	E	80386SX Interrupt Gate
7	286 Trap Gate	F	80386SX Trap Gate

Figure 4.6. System Descriptors

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is byte-granular or page-granular.

System Descriptor Formats (S = 0)

System segments describe information about operating system tables, tasks, and gates. Figure 4.6 shows the general format of system segment descriptors, and the various types of system segments. 80386SX system descriptors (which are the same as 386 system descriptors) contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

Differences Between 80386SX and 286 Descriptors

In order to provide operating system compatibility between the 80286 and the 80386SX, the 80386SX supports all of the 80286 segment descriptors. The 80286 system segment descriptors contain a 24-bit base address and 16-bit limit, while the 80386SX system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit. The word count field specifies the number of 16-bit quantities to copy for 286 call gates and 32-bit quantities for 80386SX call gates.

Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.7. The TI bit selects either 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 which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

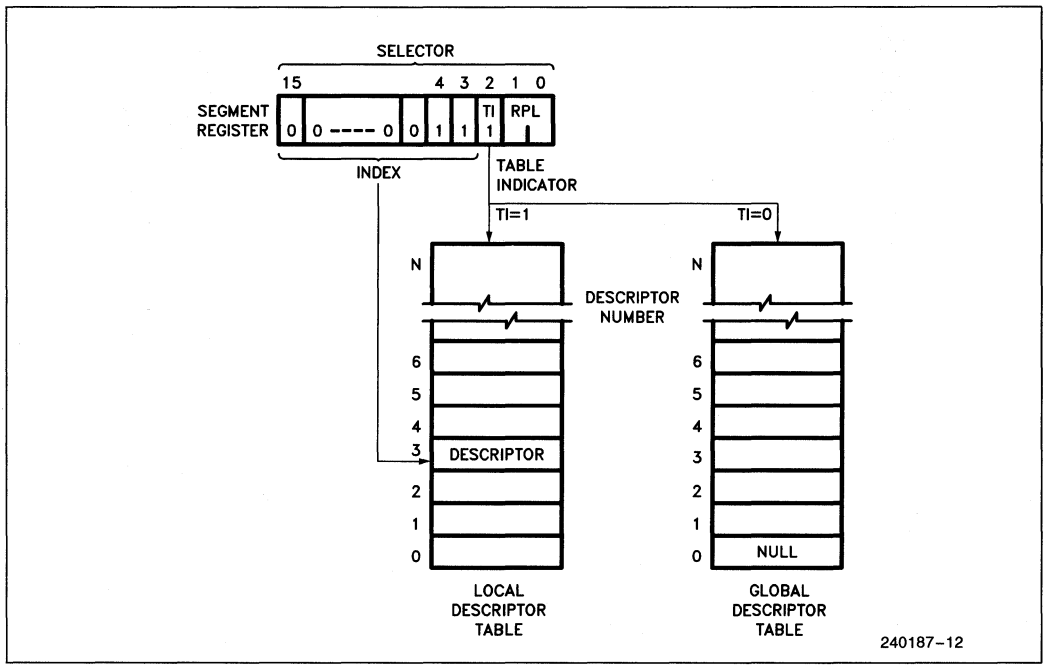


Figure 4.7. Example Descriptor Selection

4.3 Protection

The 80386SX has four levels of protection which are optimized to support a multi-tasking operating system and 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. The 80386SX also offers an additional type of protection on a page basis when paging is enabled.

The four-level hierarchical privilege system is an extension of the user/supervisor privilege mode commonly used by minicomputers. The user/supervisor mode is fully supported by the 80386SX paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged level.

RULES OF PRIVILEGE

The 80386SX 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

At any point in time, a task on the 80386SX always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task 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 (numerically larger) level of a task's CPL and a selector's RPL. 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.

Table 4.2. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt instruction Exception External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt instruction, Exception, External Interrupt	Task Gate	IDT

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

I/O Privilege

The I/O privilege level (IOPL) lets the operating system code executing at CPL = 0 define the least privileged level at which I/O instructions can be used. An exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged then the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI, LOCK prefix.

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 a data segment register (DS, ES, FS, GS) the 80386SX makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

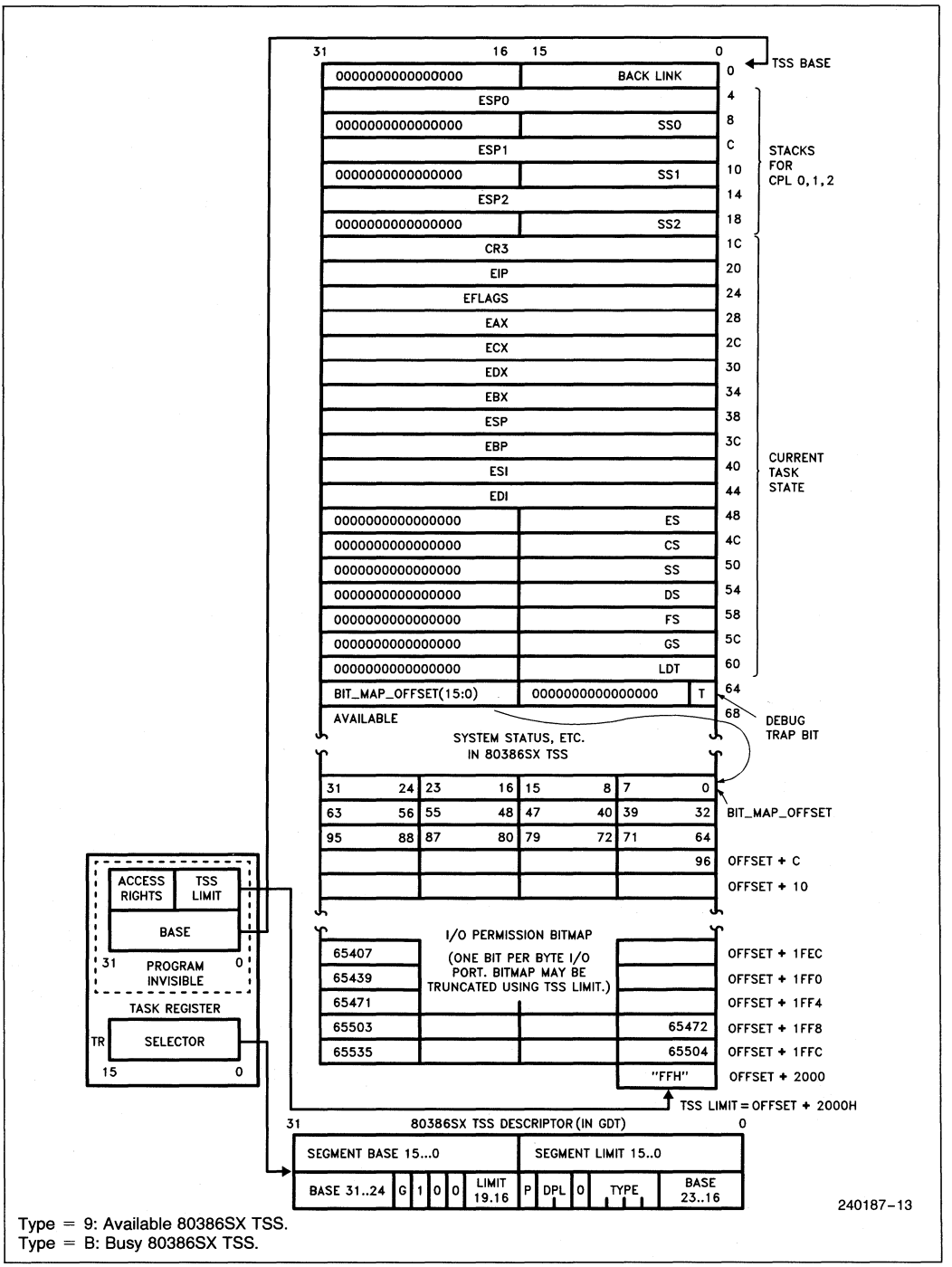
Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an exception 13 (general protection fault) is generated.

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

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13.



Type = 9: Available 80386SX TSS.
Type = B: Busy 80386SX TSS.

Figure 4.8. 80386SX TSS and TSS Registers

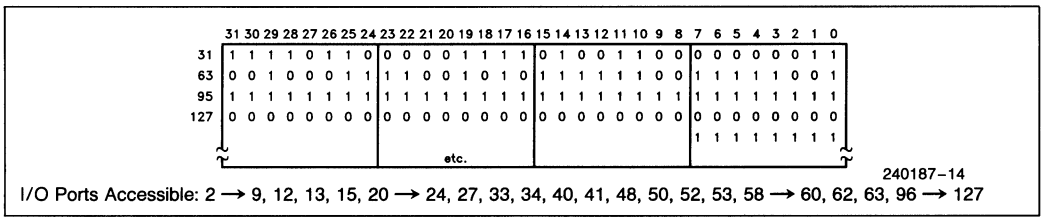


Figure 4.9. Sample I/O Permission Bit Map

CALL GATES

Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The 80386SX directly supports this operation by providing a task switch instruction in hardware. The 80386SX 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. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.8) containing the entire 80386SX execution state. A task gate descriptor contains a TSS selector. The 80386SX supports both 286 and 80386SX-type TSSs. The limit of a 80386SX TSS must be greater than 64H (2BH for a 286 TSS), and can be as large as 16 megabytes. 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, or 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 80386SX 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 TSS descriptor are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The currently executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT = 0 the IRET instruction performs the regular return. If 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 80386SX task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. A 286 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 VM (Virtual Mode) bit 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 by a task switch.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CR0 register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80386SX switches task, it sets the TS bit. The 80386SX 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.

The T bit in the 80386SX 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 80386SX 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 the GDT must contain descriptors for the initial code and data segments.

Protected Mode is enabled by loading CR0 with PE bit set. This can be accomplished by using the **MOV CR0, R/M** instruction. After enabling Protected Mode, the next instruction should execute an inter-segment 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 is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor.

4.4 Paging

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

PAGE ORGANIZATION

The 80386SX 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 80386SX: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 80386SX paging mechanism are the same size, namely 4K bytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.10 shows how the paging mechanism works.

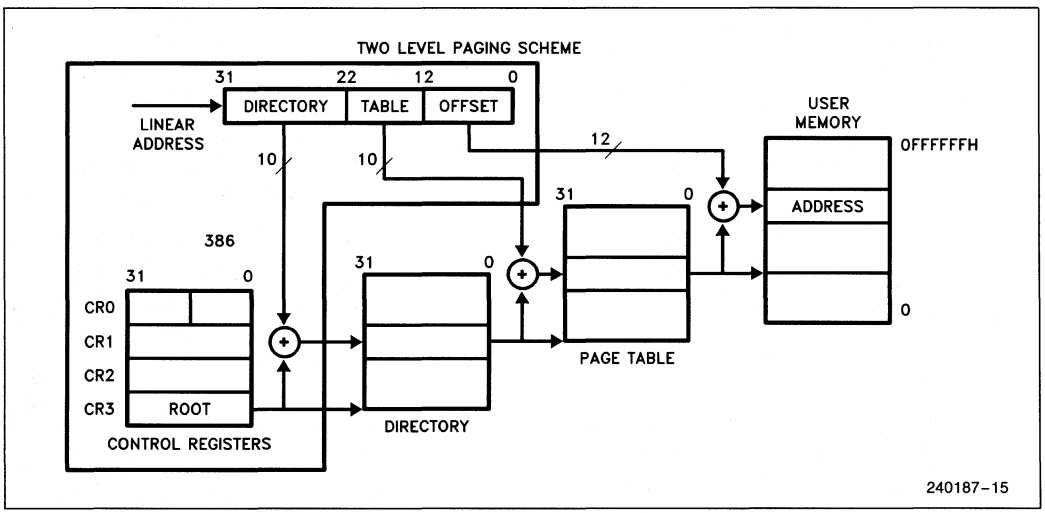


Figure 4.10. Paging Mechanism

31 — 12	11 — 9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	System Software Defineable	0	0	D	A	0	0	U — S	R — W	P

Figure 4.11. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	System Software Defineable		0	0	D	A	0	0	U — S	R — W	P

Figure 4.12. Page Table Entry (Points to Page)

Page Fault Register

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

Page Descriptor Base Register

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory (this value is truncated to a 24-bit value associated with the 80386SX's 16 megabyte physical memory limitation). The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it with a **MOV CR3, reg** instruction causes the page table entry cache to be flushed, as will a task switch through a TSS which changes the value of CR0.

Page Directory

The Page Directory is 4k bytes long and allows up to 1024 page directory entries. Each page directory entry contains information about the page table and the address of the next level of tables, the Page Tables. The contents of a Page Directory Entry are shown in figure 4.11. The upper 10 bits of the linear address ($A_{31}-A_{22}$) are used as an index to select the correct Page Directory Entry.

The page table address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the next set of tables, the page tables. The lower 12 bits of the page table address are zero so that the page table addresses appear on 4 kbyte boundaries. For an 80386 system the upper 20 bits will select one of 2^{20} page tables, but for an 80386SX system the upper 20 bits only select one of 2^{12} page tables. Again, this is because the 80386SX is limited to a 24-bit physical address and the upper 8 bits ($A_{24}-A_{31}$) are truncated when the address is output on its 24 address pins.

Page Tables

Each Page Table is 4K bytes long and allows up to 1024 Page table Entries. Each page table entry contains information about the Page Frame and its address. The contents of a Page Table Entry are shown in figure 4.12. The middle 10 bits of the linear address ($A_{21}-A_{12}$) are used as an index to select the correct Page Table Entry.

The Page Frame Address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the Page Frame. The lower 12 bits of the Page Frame Address are zero so that the Page Frame addresses appear on 4 kbyte boundaries. For an 80386 system the upper 20 bits will select one of 2^{20} Page Frames, but for an 80386SX system the upper 20 bits only select one of 2^{12} Page Frames. Again, this is because the 80386SX is limited to a 24-bit physical address space and the upper 8 bits ($A_{24}-A_{31}$) are truncated when the address is output on its 24 address pins.

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 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. 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 disk the page is stored.

The A (Accessed) bit is set by the 80386SX for both types of entries before a read or write access occurs to an address covered by the entry. The D (Dirty) bit 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 80386SX, the processor generates a Read- Modify-Write cycle which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked system software definable in Figures 4.11 and Figure 4.12 are software definable. System software writers are free to use these bits for whatever purpose they wish.

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

The 80386SX 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 second level Page Table Entry apply only to the page described by that entry. While 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 for a given page are obtained by taking the most restrictive of the U/S and R/W from the Page Directory Table Entries and using these bits to address the page.

TRANSLATION LOOKASIDE BUFFER

The 80386SX 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 80386SX keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB coupled with a 4K page size results in coverage of 128K bytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of greater than 98%. This means that the processor will only have to access the two-level page structure for less than 2% of all memory references.

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 24-bit physical address is calculated and is placed on the address bus.

If the page table entry is not in the TLB, the 80386SX 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 80386SX 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 80386SX 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. If P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault Exception 14.

The processor will also generate a Page Fault (Exception 14) if the memory reference violated the page protection attributes. CR2 will hold the linear address which caused the page fault. Since Exception 14 is classified as a fault, CS:EIP will point to the instruction causing the page-fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the Page Fault. Figure 4.13 shows the format of the Page Fault error code and the interpretation of the bits. Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.14 indicates what type of access caused the page fault.

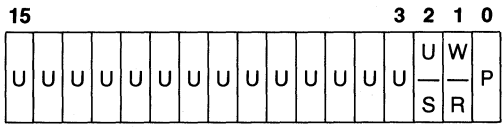


Figure 4.13. Page Fault Error Code Format

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

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

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

U = Undefined

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

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

Figure 4.14. Type of Access Causing Page Fault

OPERATING SYSTEM RESPONSIBILITIES

When the operating system enters or exits paging mode (by setting or resetting bit 31 in the CR0 register) a short JMP must be executed to flush the 80386SX's prefetch queue. This ensures that all instructions executed after the address mode change will generate correct addresses.

The 80386SX takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

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

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

4.5 Virtual 8086 Environment

The 80386SX allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 80386SX protection mechanism.

VIRTUAL 8086 ADDRESSING MECHANISM

One of the major differences between 80386SX 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 a fashion identical 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 80386SX allows the operating system to specify which programs use the 8086 address mechanism and which programs use Protected Mode addressing on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 80386SX. Like Real Mode, Virtual Mode addresses that exceed one megabyte 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 one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into as many as 256 pages. Each one of the pages can be located anywhere within the maximum 16 megabyte physical address space of the 80386SX. 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.

PROTECTION AND I/O PERMISSION BIT MAP

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

The following are privileged instructions, which may be executed only at Privilege Level 0. Attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL $\geq$ 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 and the protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

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

The instructions which are IOPL sensitive in Protected Mode are:

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

In Virtual 8086 Mode the following instructions are IOPL-sensitive:

```
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 to be virtualized to the virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 mode. Note that the INT 3, INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 Mode.

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT, and OUTS. The 80386SX has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the *I/O Permission Bit Map* in the TSS segment (see Figures 4.8 and 4.9). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the **I/O map base** field in the fixed portion of the TSS. The **I/O map base** field is 16 bits wide and contains the offset of the beginning of the I/O permission map. The upper limit of the I/O permission map is the same as the limit of the TSS segment.

In protected mode when an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether CPL ≤ IOPL. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map (in Virtual 8086 Mode, the processor consults the map without regard for the IOPL).

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at **I/O map base** + 5, bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one-bits in the map. The **I/O map base** should be at least one byte less than the TSS limit, the last byte beyond the I/O mapping information must contain all 1's.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

IMPORTANT IMPLEMENTATION NOTE: Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 80386SX's TSS segment (see Figure 4.8).

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 80386SX operating system. The 80386SX operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

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

The 80386SX operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 80386SX 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 80386SX operating system.

An 80386SX operating system can provide a Virtual 8086 Environment which is totally transparent to the application software by 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 a 32-bit IRET instruction at CPL = 0 where the stack has a 1 in the VM bit of its EFLAGS image, or a Task Switch (at any CPL) to a 80386SX task whose 80386SX TSS has a EFLAGS image containing a 1 in the VM bit position while the processor is executing in the Protected Mode. POPF does not affect the VM bit but a PUSHF always pushes a 0 in the VM bit.

The transition out of virtual 8086 mode to 80386SX protected mode occurs only on receipt of an interrupt or exception. In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 80386SX mode. As part of the interrupt processing the VM bit is cleared.

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

Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the 80386SX format (type 9 or 11 descriptor). A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 80386SX TSS. All of the programmer visible state, including the EFLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 80386SX TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 80386SX 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 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 exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 80386SX Trap Gate (Type 14), or 80386SX Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 80386SX gates must be used since 286 gates save only the low 16 bits of the EFLAGS register (the VM bit will not be saved). Also, the 16-bit IRET used to terminate the 286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 80386SX 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, TF, and IF bits.
2. Interrupt and Trap gates must perform a level switch from 3 (where the Virtual 8086 Mode program executes) to level 0 (so IRET can return).
3. Push the 8086 segment register values onto the new stack, in this order: GS, FS, DS, ES. These are pushed as 32-bit quantities. Then load these 4 registers with null selectors (0).
4. Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits), then pushing the 32-bit ESP register saved above.
5. Push the 32-bit EFLAGS register saved in step 1.
6. Push the old 8086 instruction onto the new stack by pushing the CS register (as 32-bits), 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 80386SX mode.

The transition out of V86 mode performs a level change and stack switch, in addition to changing back to protected mode. Also 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 286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prologue and epilogue code for state saving 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 cause a trap in the interrupt handler. Interrupt routines which expect or return values in the 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 80386SX IRET instruction (operand size = 32) can be used and must be executed at level 0 to change the VM bit to 1.

1. If the NT bit in the FLAGS register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed. Otherwise, continue with the following sequence:
2. Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
3. Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM = 0, this CS load is done as a protected mode segment load. If VM = 1, this will be done as an 8086 segment load.
4. Increment the ESP register by 4 to bypass the FLAGS image which was 'popped' in step 1.
5. If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP = 20], respectively, where the new value of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

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

5.0 FUNCTIONAL DATA

The 80386SX features a straightforward functional interface to the external hardware. The 80386SX has separate parallel buses for data and address. The data bus is 16-bits in width, and bi-directional. The address bus outputs 24-bit address values using 23 address lines and two byte enable signals.

The 80386SX has two selectable address bus cycles: address pipelined and non-address pipelined. The address pipelining option allows as much time as possible for data access by starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor CLK 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. 80386SX bus cycles perform data transfer in a minimum of only two clock periods. The maximum transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 80386SX can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 80386SX, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

5.1 Signal Description Overview

Ahead is a brief description of the 80386SX input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a LOW voltage. When no # is present after the signal name, the signal is asserted when at the HIGH voltage level.

Example signal: M/IO# — HIGH voltage indicates Memory selected

— LOW voltage indicates I/O selected

The signal descriptions sometimes refer to AC timing parameters, such as 't_{25} Reset Setup Time' and 't_{26} Reset Hold Time.' The values of these parameters can be found in Table 7.4.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 80386SX. 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 5.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26}.

DATA BUS (D_{15}–D_0)

These three-state bidirectional signals provide the general purpose data path between the 80386SX and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

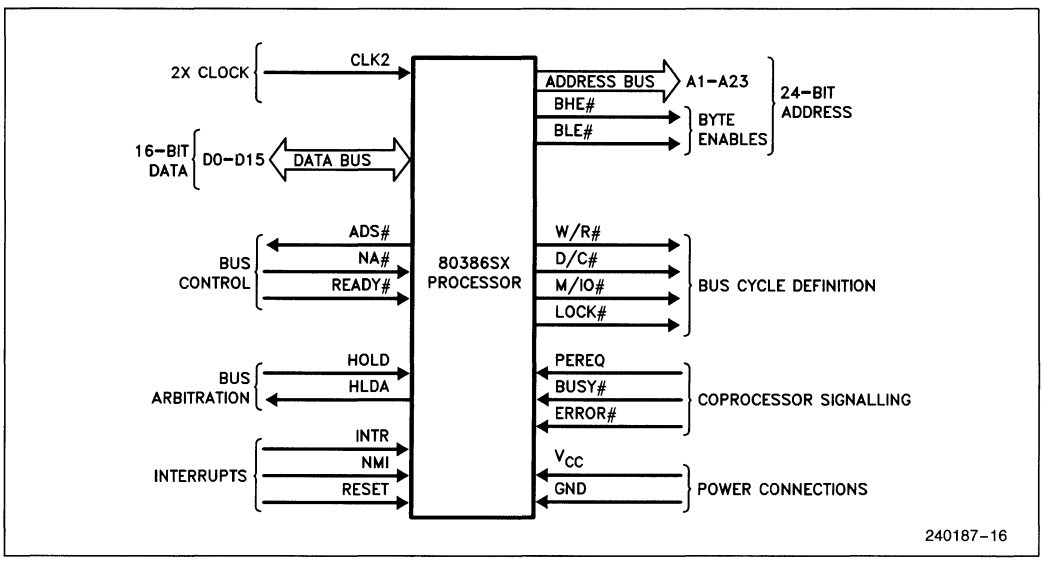


Figure 5.1. Functional Signal Groups

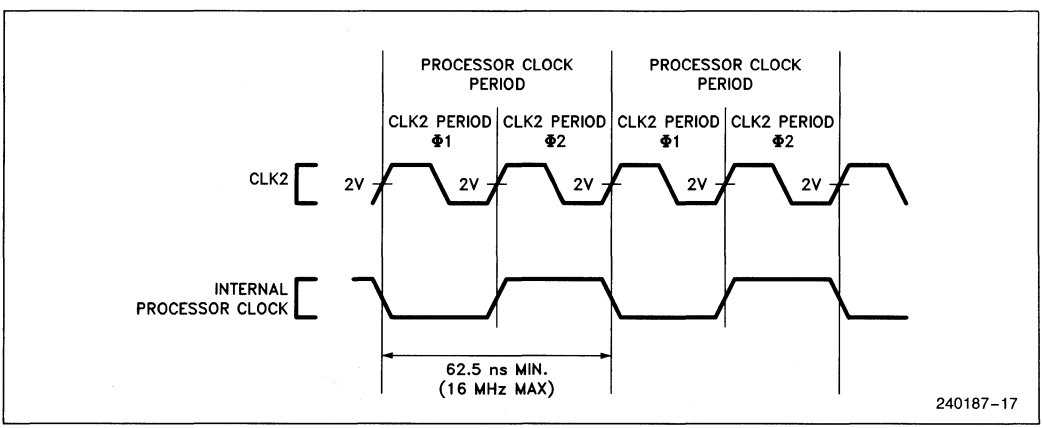


Figure 5.2. CLK2 Signal and Internal Processor Clock

ADDRESS BUS (A_{23}–A_1, BHE#, BLE#)

These three-state outputs provide physical memory addresses or I/O port addresses. A_{23}–A_{16} are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A_{22}–A_{16} are driven LOW, and A_{23} is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 80386SX for coprocessor commands is 8000F8H, the I/O address driven by the 80386SX for coprocessor data is 8000FCH for cycles to the 80387SX.

The address bus is capable of addressing 16 megabytes of physical memory space (000000H through FFFFFFH), and 64 kilobytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs, BHE# and BLE# directly indicate which bytes of the 16-bit data bus are involved with the current transfer. BHE# applies to D_{15}–D_8 and BLE# applies to D_7–D_0. If both BHE# and BLE# are asserted, then 16 bits of data are being transferred. See Table 5.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

BUS CYCLE DEFINITION SIGNALS (W/R#, D/C#, M/IO#, LOCK#)

These three-state outputs define the type of bus cycle being performed: W/R# distinguishes between write and read cycles, D/C# distinguishes between data and control cycles, M/IO# distinguishes between memory and I/O cycles, and LOCK# distinguishes between locked and unlocked bus cycles. All of these signals are active LOW and will float during bus acknowledge.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as ADS# (Address Status output) becomes active. The LOCK# is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after ADS# becomes active. Exact bus cycle definitions, as a function of W/R#, D/C#, and M/IO# are given in Table 5.2.

LOCK# indicates that other system bus masters are not to gain control of the system bus while it is active. LOCK# is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when ready is returned at the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when READY# is returned in a previous bus cycle and another is pending (ADS# is active) or the clock in which ADS# is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be locked longer than desired. The LOCK# signal may be explicitly activated by the LOCK prefix on certain instructions. LOCK# is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

Table 5.1. Byte Enable Definitions

BHE#	BLE#	Function
0	0	Word Transfer
0	1	Byte transfer on upper byte of the data bus, D_{15}–D_8
1	0	Byte transfer on lower byte of the data bus, D_7–D_0
1	1	Never occurs

Table 5.2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
0	0	0	Interrupt Acknowledge	Yes
0	0	1	does not occur	—
0	1	0	I/O Data Read	No
0	1	1	I/O Data Write	No
1	0	0	Memory Code Read	No
1	0	1	Halt: Address = 2, BHE# = 1, BLE# = 0 / Shutdown: Address = 0, BHE# = 1, BLE# = 0	No
1	1	0	Memory Data Read	Some Cycles
1	1	1	Memory Data Write	Some Cycles

BUS CONTROL SIGNALS (ADS#, READY#, NA#)

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

Address Status (ADS#)

This three-state output indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A_{23}–A_1) are being driven at the 80386SX pins. ADS# is an active LOW output. Once ADS# is driven active, valid address, byte enables, and definition signals will not change. In addition, ADS# will remain active until its associated bus cycle begins (when READY# is returned for the previous bus cycle when running pipelined bus cycles). When address pipelining is utilized, maximum throughput is achieved by initiating bus cycles when ADS# and READY# are active in the same clock cycle. ADS# will float during bus hold acknowledge. See sections **Non-Pipelined Address** (page 49) and **Pipelined Address** (page 50) for additional information on how ADS# is asserted for different bus states.

Transfer Acknowledge (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BHE# and BLE# are accepted or provided. When READY# is sampled active during a read cycle or interrupt acknowledge cycle, the 80386SX latches the input data and terminates the cycle. When READY# is sampled active during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY# must always meet setup and hold times t_{19} and t_{20} for correct operation.

Next Address Request (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BHE#, BLE#, A_{23}–A_1, W/R#, D/C# and M/IO# from the 80386SX even if the end of the current cycle is not being acknowledged on READY#. If this input is active when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. NA# is ignored in CLK cycles in which ADS# or READY# is activated. This signal is active LOW and must satisfy setup and hold times t_{15} and t_{16} for correct operation. See **Pipelined Address** (page 50) and **Read and Write Cycles** (page 47) for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

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** (page 57) for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the 80386SX requires bus mastership. When control is granted, the 80386SX floats A_{23}–A_1, BHE#, BLE#, D_{15}– D_0, LOCK#, M/IO#, D/C#, W/R# and ADS#, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 80386SX will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 80386SX floated outputs have internal pull-up resistors. See **Resistor Recommendations** (page 64) for additional information. HOLD is not recognized while RESET is active. 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 a level-sensitive, active HIGH, synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 80386SX has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 80386SX. The other output signals or bidirectional signals (D_{15}–D_0, BHE#, BLE#, A_{23}–A_1, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus master may

control them. These pins remain OFF throughout the time that HLDA remains active (see Table 5.3)). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** (page 64) for additional information.

When the HOLD signal is made inactive, the 80386SX will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 5.3. Output pin State During HOLD

Pin Value	Pin Names
1	HLDA
Float	LOCK#, M/IO#, D/C#, W/R#, ADS#, A_{23}–A_1, BHE#, BLE#, D_{15}–D_0

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.

HOLD Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK# signal (internal to the CPU) activated by the LOCK# prefix, and interrupts. The 80386SX will not honor a HOLD request until the current bus operation is complete. Table 5.4 shows the types of bus operations that can affect HOLD latency, and indicates the types of delays that these operations may introduce. When considering maximum HOLD latencies, designers must select which of these bus operations are possible, and then select the maximum latency from among them.

As indicated in Table 5.4, wait states affect HOLD latency. The 80386SX will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that READY# returns sufficiently soon.

{*** **Not Available At This Time** ***}

Table 5.4. Locked Bus Operations Affecting HOLD Latency in Systems Clocks

COPROCESSOR INTERFACE SIGNALS (PEREQ, BUSY#, ERROR#)

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 80386SX and its 80387SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 80386SX. In response, the 80386SX transfers information between the coprocessor and memory. Because the 80386SX has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is a level-sensitive active HIGH asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 K-ohms to ground so that it will not float active when left unconnected.

Coprocessor Busy (BUSY#)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 80386SX encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the BUSY# input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT, FNSTENV, FNSAVE, FNSTSW, FNSTCW and FNCLEX coprocessor instructions are allowed to execute even if BUSY# is active, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the

80386SX performs an internal self-test (see **Bus Activity During and Following Reset**, page 58). If BUSY# is sampled HIGH, no self-test is performed.

Coprocessor Error (ERROR#)

When asserted (LOW), 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 80386SX when a coprocessor instruction is encountered, and if active, the 80386SX generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 80386SX generating exception 16 even if ERROR# is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

ERROR# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

INTERRUPT SIGNALS (INTR, NMI, RESET)

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 80386SX Flag Register IF bit. When the 80386SX 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 D_7–D_0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction in the 80386SX's Execution Unit. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the instruction. If recognized, the 80386SX will begin execution of the interrupt.

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 an active HIGH, rising edge-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the instruction boundary in the 80386SX's Execution Unit.

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.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, an INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 80386SX encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 80386SX's Execution Unit except for the following cases:
   - — Repeat string instructions can be interrupted after each iteration.
   - — If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP. This allows the entire stack pointer to be loaded without interruption.
   - — If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

The longest latency occurs when the interrupt request arrives while the 80386SX is executing a long instruction such as multiplication, division, or a task-switch in the protected mode.

4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 80386SX.

RESET

This input signal suspends any operation in progress and places the 80386SX in a known reset state. The 80386SX is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5.5. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 80386SX was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26}, must be met in order to assure proper operation of the 80386SX.

Table 5.5. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
ADS#	1
$D_{15}-D_0$	Float
BHE#, BLE#	0
$A_{23}-A_1$	1
W/R#	0
D/C#	1
M/IO#	0
LOCK#	1
HLDA	0

5.2 Bus Transfer Mechanism

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

The 80386SX address signals are designed to simplify external system hardware. Higher-order address bits are provided by $A_{23}-A_1$. BHE# and BLE# provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs BHE# and BLE# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5.6.

Table 5.6. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BLE#	D_7-D_0	(byte 0 — least significant)
BHE#	$D_{15}-D_8$	(byte 1 — most significant)

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See section **5.4 Bus Functional Description**.

5.3 Memory and I/O Spaces

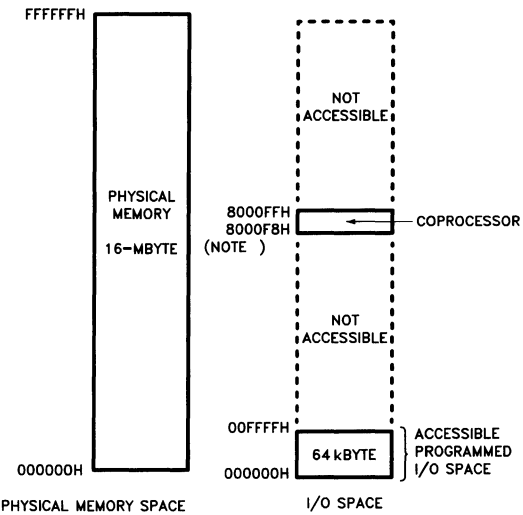
Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 5.3, physical memory addresses range from 000000H to 0FFFFFFH (16 megabytes) and I/O addresses from 000000H to 00FFFFH (64 kilobytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A_{23} and M/IO# signals.

5.4 Bus Functional Description

The 80386SX has separate, parallel buses for data and address. The data bus is 16-bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

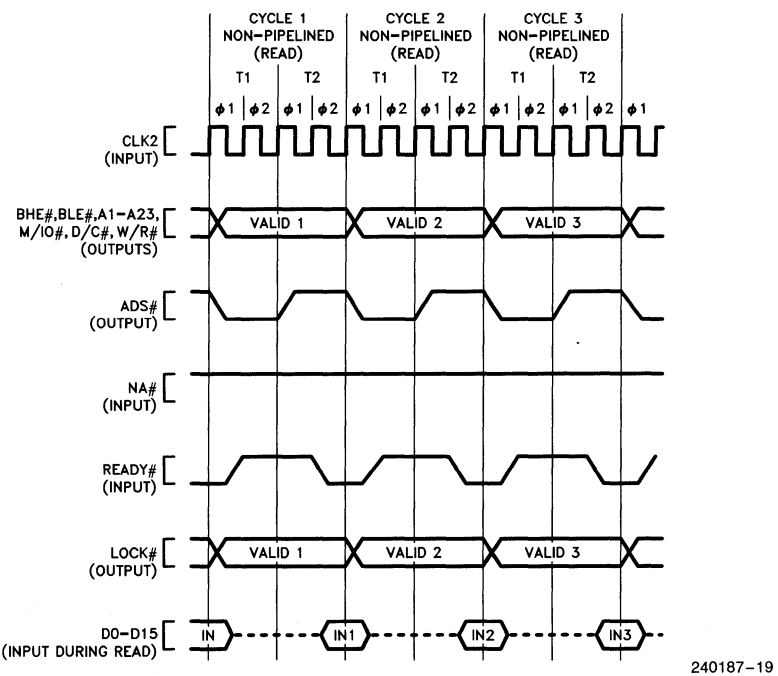
The definition of each bus cycle is given by three signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals, BHE# and BLE#, and the other address signals $A_{23}-A_1$. A status signal, ADS#, indicates when the 80386SX issues a new bus cycle definition and address.

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



NOTE:
Since A23 is HIGH during automatic communication with coprocessor, A23 HIGH and M/IO# LOW can be used to easily generate a coprocessor select signal.

Figure 5.3. Physical Memory and I/O Spaces



Fastest non-pipelined bus cycles consist of T1 and T2

Figure 5.4. Fastest Read Cycles with Non-pipelined Address Timing

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 (always locked)
8. Indicate halt, or indicate shutdown

Table 5.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** (page 40) for additional information.

When the 80386SX bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected externally. The idle state can be identified by the 80386SX 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 having been terminated. The hold acknowledge state is identified by the 80386SX 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 80386SX bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5.4. 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.

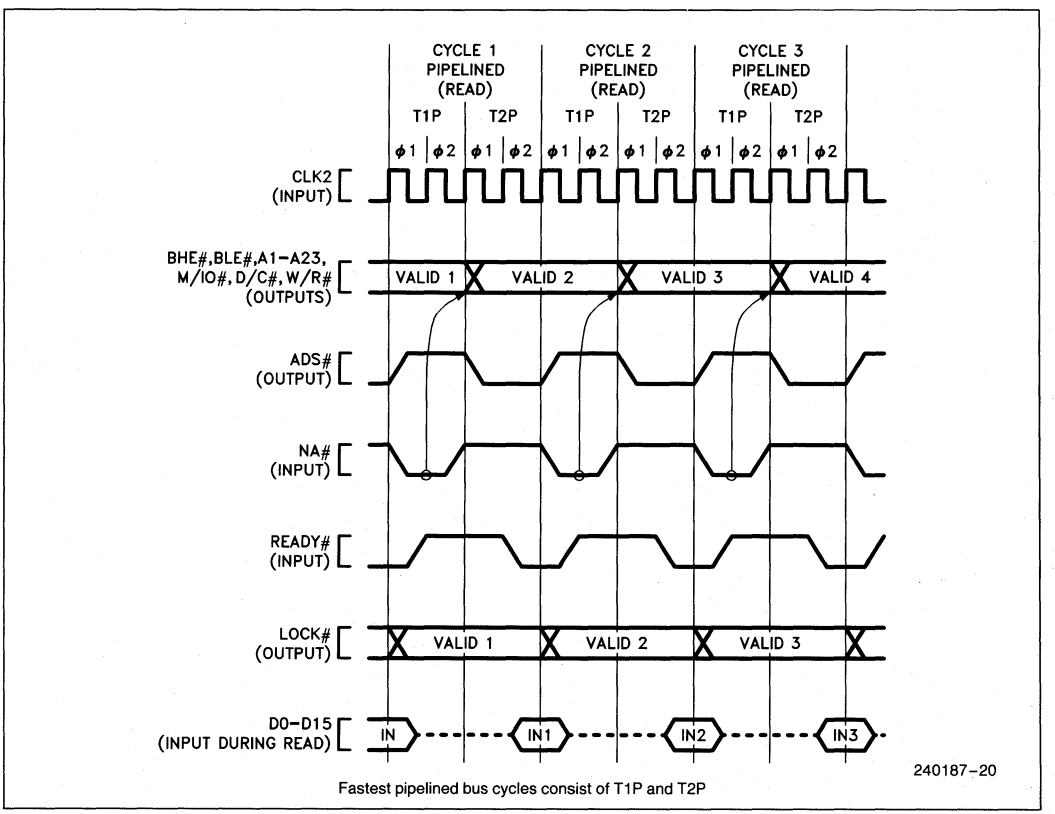


Figure 5.5. Fastest Read Cycles with Pipelined Address Timing

Every bus cycle continues until it is acknowledged by the external system hardware, using the 80386SX 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 active.

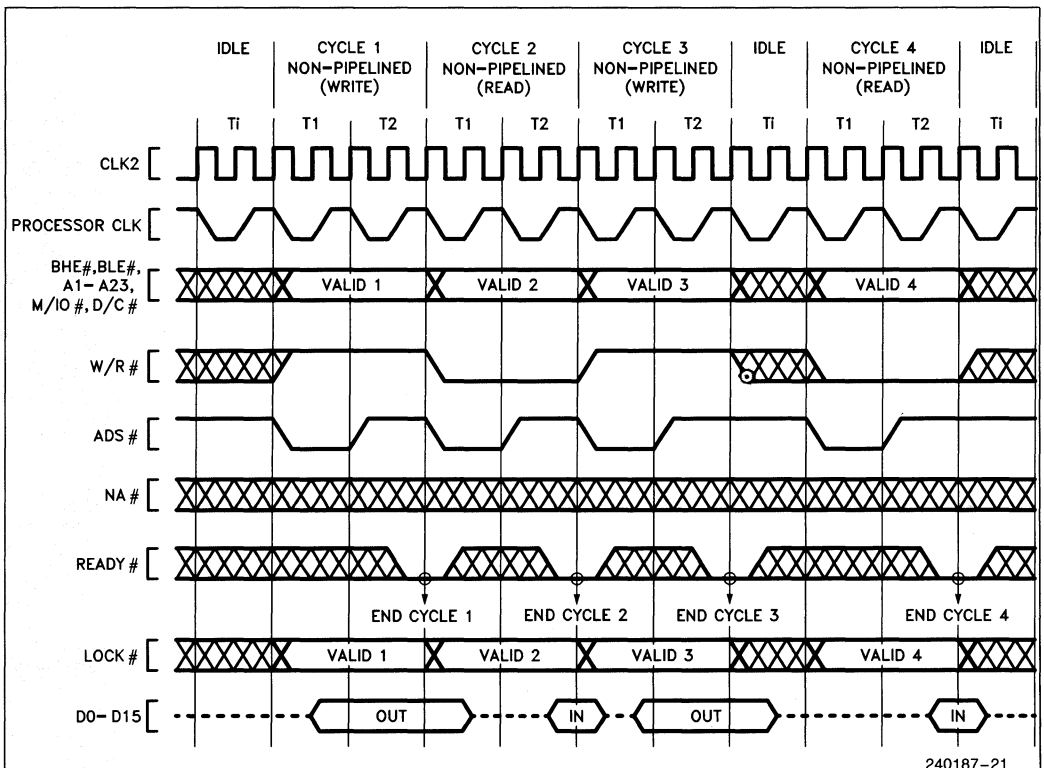
The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address (NA#) input.

When address pipelining is selected the address (BHE#, BLE# and A_{23}–A_1) and definition (W/R#, D/C#, M/IO# and LOCK#) of the next cycle are available before the end of the current cycle. To signal their availability, the 80386SX address status output (ADS#) is asserted. Figure 5.5 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5.5 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 by one T-state time compared to that of a non-pipelined cycle.

READ AND WRITE CYCLES

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 from the processor to an external device.



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.

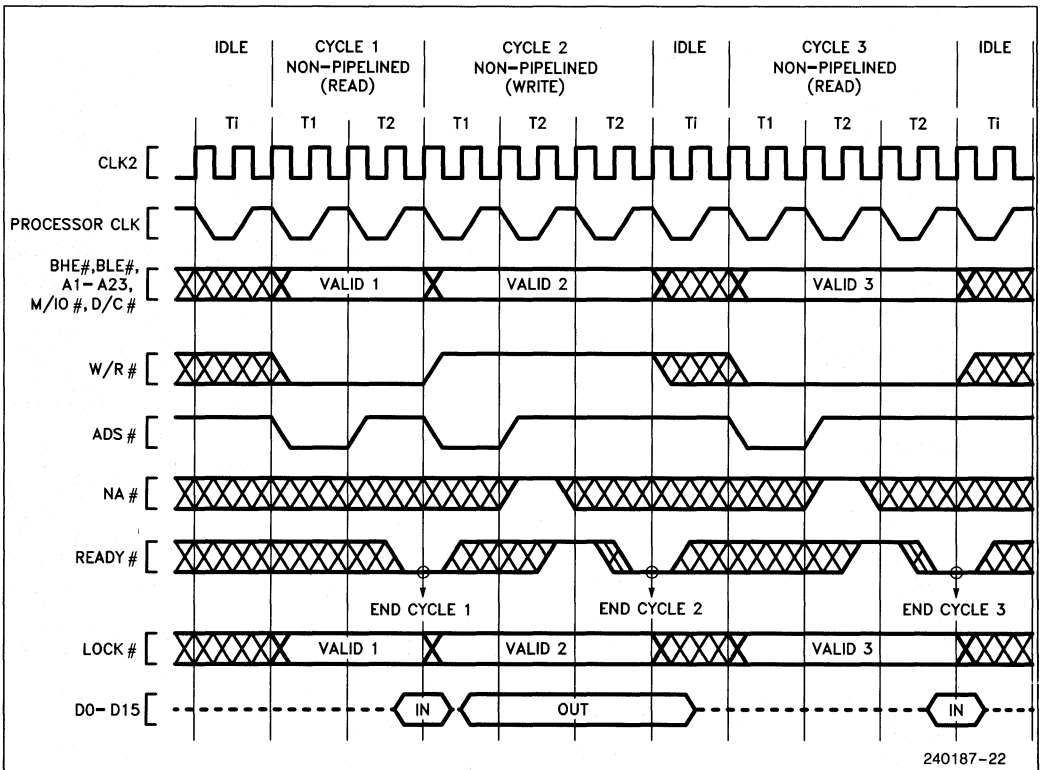
Figure 5.6. Various Bus Cycles with Non-Pipelined Address (zero wait states)

Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After an idle bus 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 80386SX 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#.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the 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, which has decoded the address and bus cycle type, asserts the READY# input at the appropriate time.

At the end of the second bus state within the bus cycle, READY# is sampled. At that time, if external hardware acknowledges the bus cycle by asserting READY#, the bus cycle terminates as shown in Figure 5.6. If READY# is negated as in Figure 5.7, the 80386SX executes another bus state (a wait state) and READY# is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by READY# asserted.

When the current cycle is acknowledged, the 80386SX terminates it. When a read cycle is acknowledged, the 80386SX latches the information present at its data pins. When a write cycle is acknowledged, the 80386SX's write data remains valid throughout phase one of the next bus state, to provide write data hold time.



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.

Figure 5.7. Various Bus Cycles with Non-Pipelined Address (various number of wait states)

Non-Pipelined Address

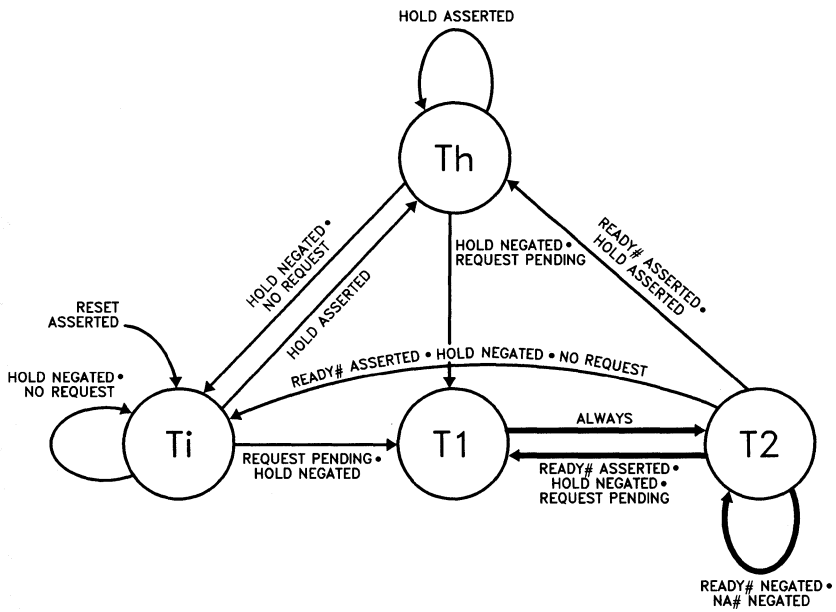
Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5.6 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5.6 shows that the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 80386SX floats its data signals to allow driving by the external device being addressed. **The 80386SX 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. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 80386SX beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

Figure 5.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. READY# is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, READY# is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5.7 Cycles 2 and 3. If NA# is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5.8. The bus transitions between four possible states, T1, T2, T_i, and T_h. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, T_i, or in the hold acknowledge state T_h.



240187-23

Bus States:
T1—first clock of a non-pipelined bus cycle (80386SX 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 (80386SX asserts HLDA).
The fastest bus cycle consists of two states T1 and T2.
Four basic bus states describe bus operation when not using pipelined address.

Figure 5.8. 80386SX Bus States (not using pipelined address)

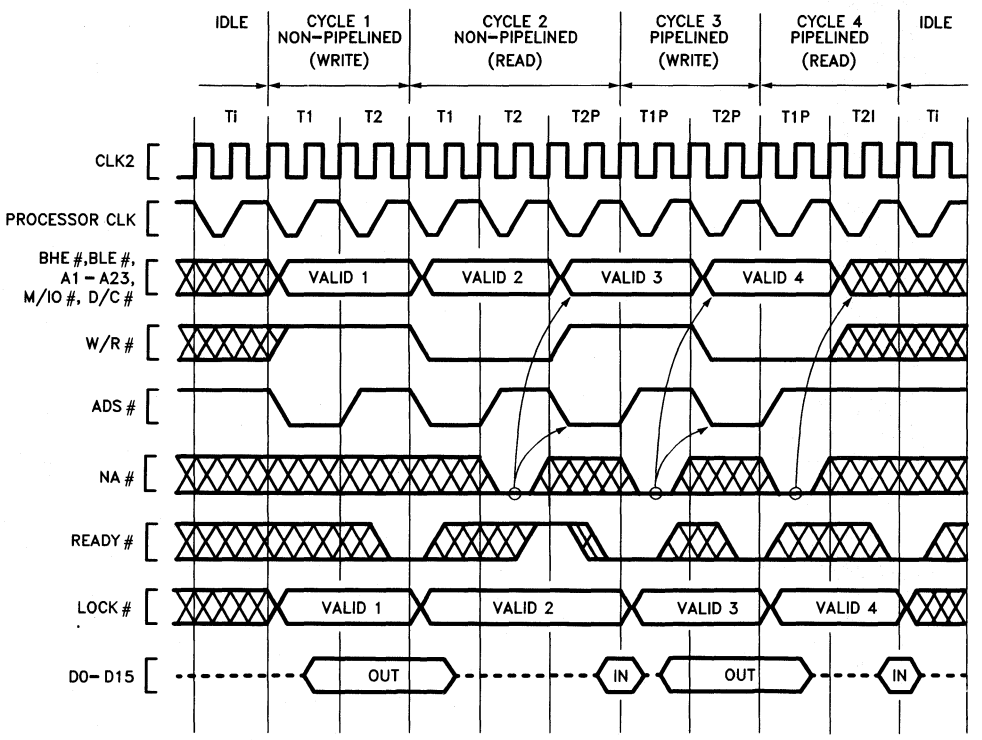
Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or T_i if there is no bus request pending, or T_h if the HOLD input is being asserted.

Use of pipelined address allows the 80386SX to enter three additional bus states not shown in Figure 5.8. Figure 5.12 on page 53 is the complete bus state diagram, including pipelined address cycles.

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 READY# asserted. ADS# is asserted by the 80386SX when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the 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 NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5.9, during which 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).



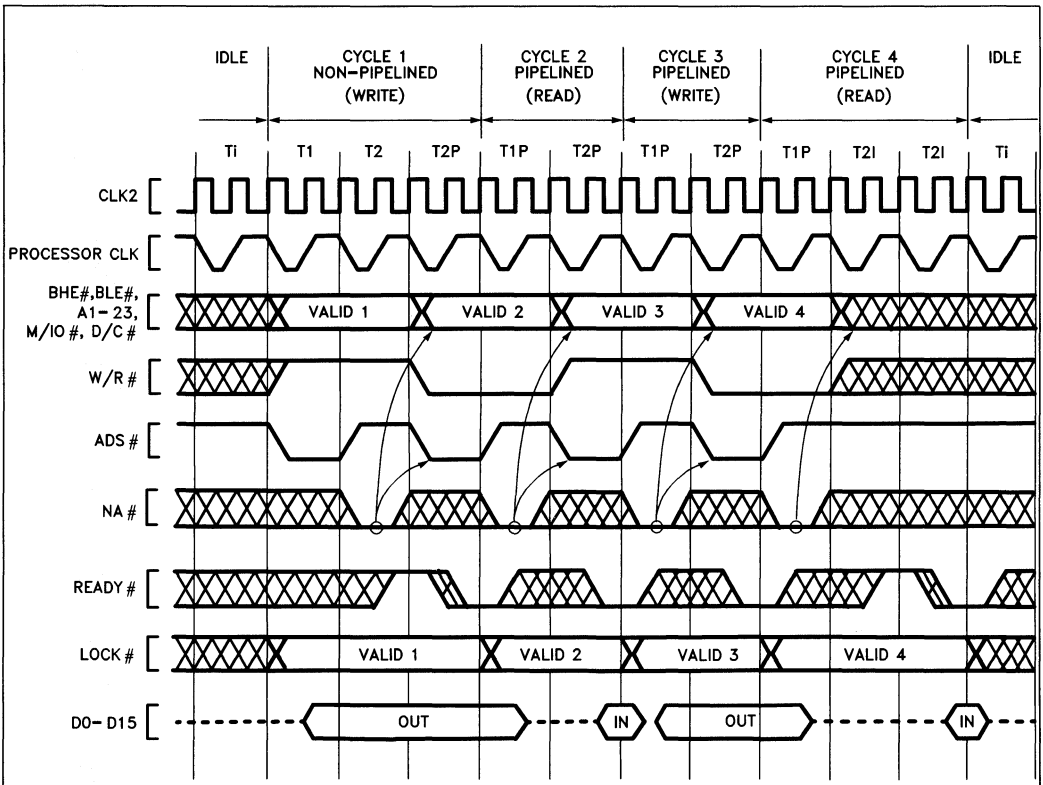
Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, NA# is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

Figure 5.9. Transitioning to Pipelined Address During Burst of Bus Cycles

If NA# is sampled active, the 80386SX is free to drive the address and bus cycle definition of the next bus cycle, and assert 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 80386SX has the following characteristics:

1. The next address may appear as early as the bus state after NA# was sampled active (see Figures 5.9 or 5.10). 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 NA# is asserted and T2I is entered instead of T2P (see Figure 5.11 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 80386SX does drive the next address. External hardware should therefore observe the ADS# output as confirmation the next address is actually being driven on the bus.
2. Any address which is validated by a pulse on the 80386SX ADS# output will remain stable on the address pins for at least two processor clock periods. The 80386SX cannot produce a new address more frequently than every two processor clock periods (see Figures 5.9, 5.10, and 5.11).
3. 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 5.11 Cycle 1).



Following any 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.

Figure 5.10. Fastest Transition to Pipelined Address Following Idle Bus State

The complete bus state transition diagram, including operation with pipelined address is given by Figure 5.12. 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.

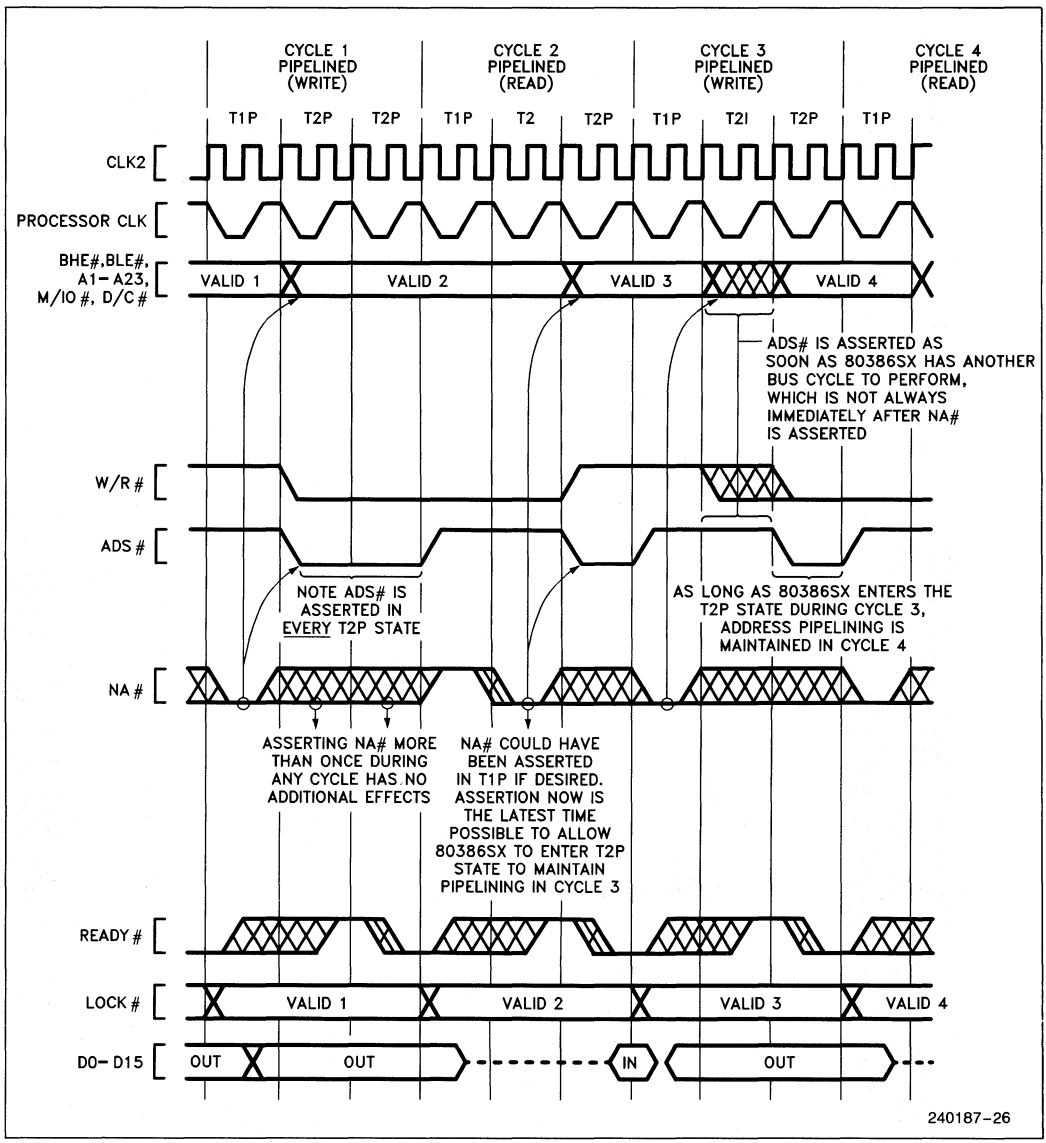
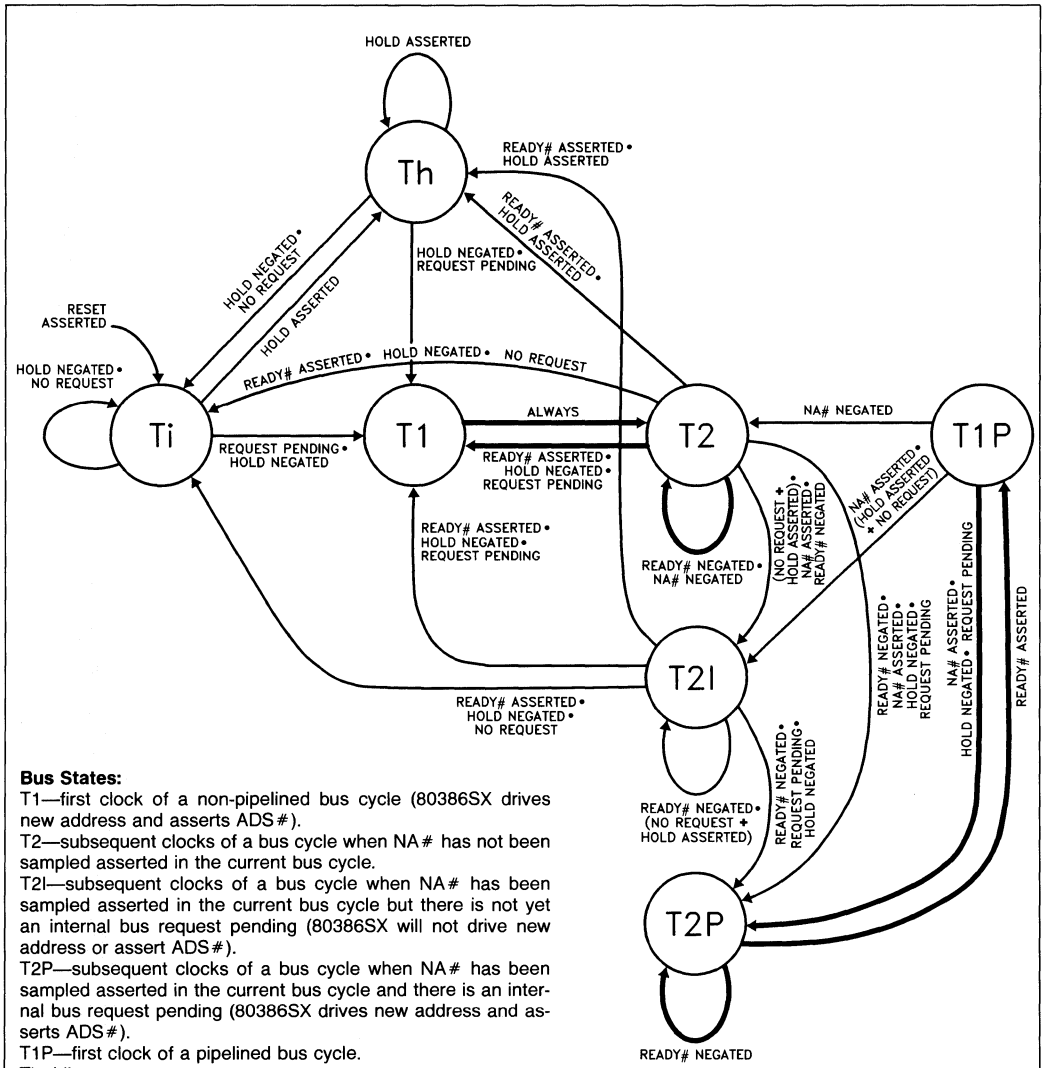


Figure 5.11. Details of Address Pipelining During Cycles with Wait States



Bus States:

T1—first clock of a non-pipelined bus cycle (80386SX 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.

T2I—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (80386SX will not drive new address or assert ADS#).

T2P—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (80386SX drives new address and asserts ADS#).

T1P—first clock of a pipelined bus cycle.

Ti—idle state.

Th—hold acknowledge state (80386SX asserts HLDA).

Asserting 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.

Figure 5.12. 80386SX Complete Bus States (including pipelined address)

Initiating and Maintaining Pipelined Address

Using the state diagram Figure 5.12, observe the transitions from an idle state, T_i, to the beginning of a pipelined bus cycle T1P. From an idle state, T_i, 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 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 bold below:

T_i, T_i, T_i, T1 - T2 - T2P, T1P - T2P,

idle	non-pipelined	pipelined
states	cycle	cycle

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

T_h, T_h, T_h, T1 - T2 - T2P, T1P - T2P,

hold acknowledge states	non-pipelined cycle	pipelined cycle

The transition to pipelined address is shown functionally by Figure 5.10 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The 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 been valid for one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. Sampling begins in T2 during Cycle 1 in Figure 5.10. Once NA# is sampled active during the current cycle, the 80386SX is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5.10 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 READY# asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 5.10 Cycle 1 and Figure 5.9 Cycle 2. Figure 5.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5.9 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 (NA# is asserted at that time), and T2P (provided the 80386SX 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 that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5.10 Cycle 1. Figure 5.10 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 NA# and detecting that the 80386SX 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 ADS#. Figures 5.9 and 5.10 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 80386SX didn't have an internal bus request prior to the acknowledgement 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 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 inactive) and NA# is sampled active in each of the bus cycles.

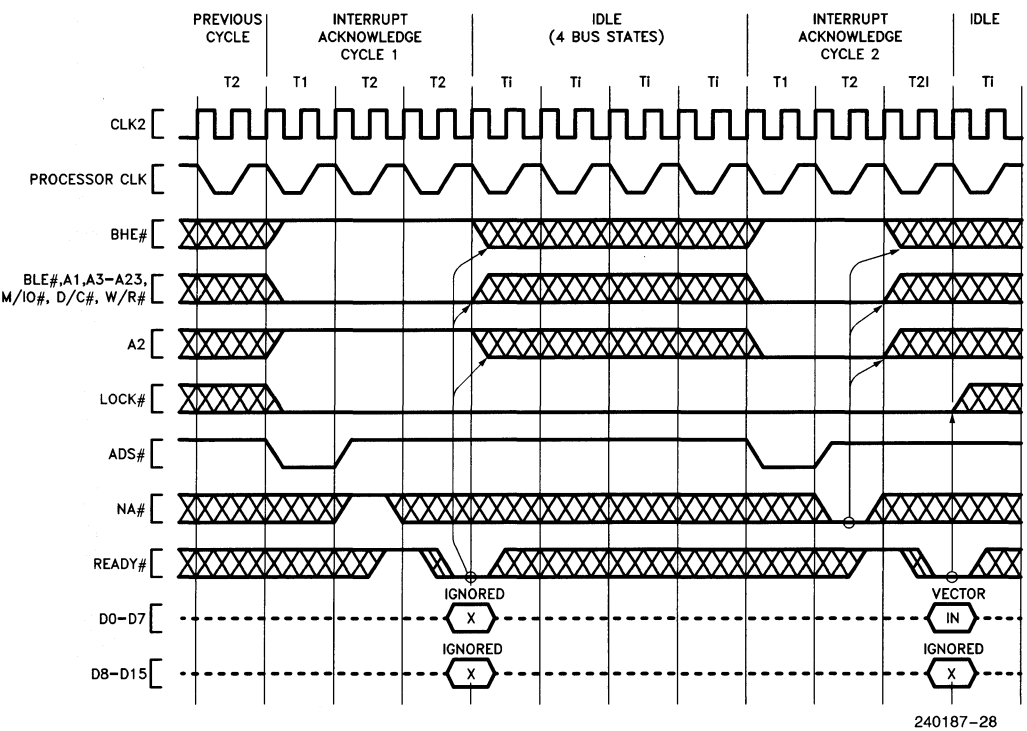
INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In response to an interrupt request on the INTR input when interrupts are enabled, the 80386SX 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 READY# sampled active.

The state of A_2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A_{23}–A_3, A_1, BLE# LOW, A_2 and BHE# HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A_{23}–A_1, BLE# LOW, and BHE# HIGH).

The 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, T_i, are inserted by the 80386SX between the two interrupt acknowledge cycles for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D_{15}–D_0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 80386SX will read an external interrupt vector from D_7–D_0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.



Interrupt Vector (0–255) is read on D0–D7 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 which is simplest for your system hardware design.

Figure 5.13. Interrupt Acknowledge Cycles

HALT INDICATION CYCLE

The 80386SX execution unit halts as a result of executing a HLT 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 on page 40, **Bus Cycle Definition Signals**, and an address of 2. The halt indication cycle must be acknowledged by READY# asserted. A halted 80386SX resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.

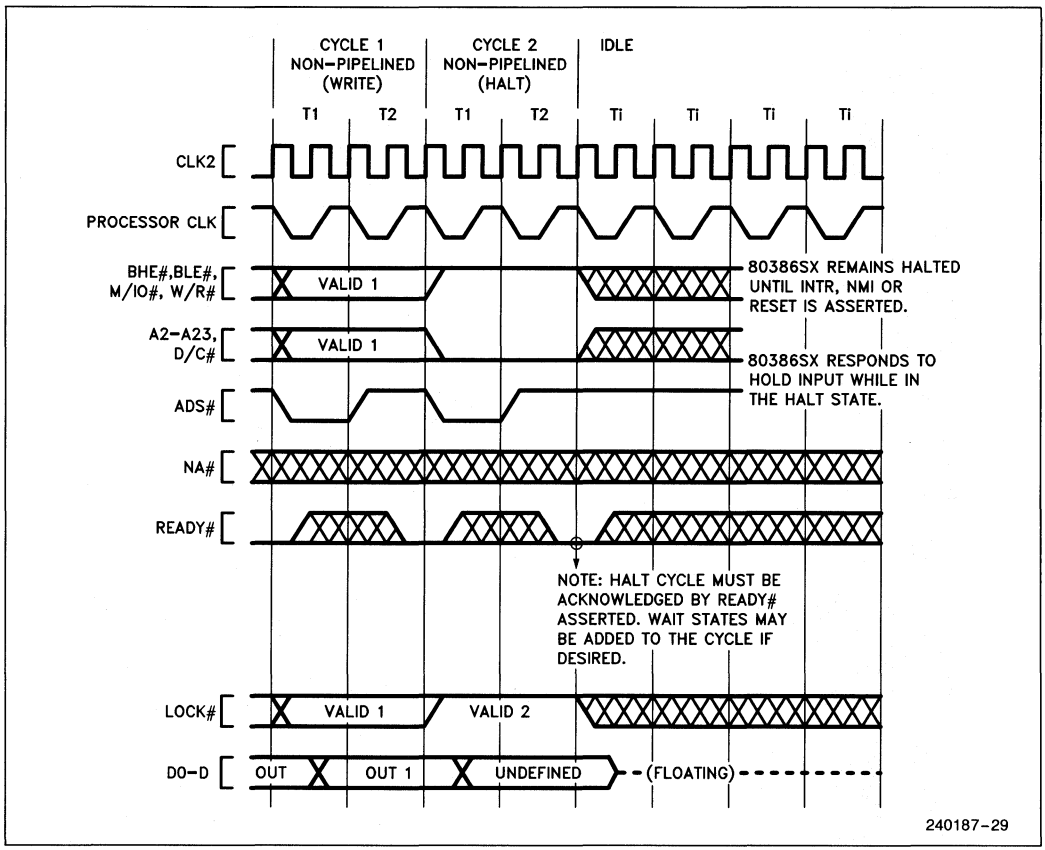


Figure 5.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 80386SX 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 Signals** (page 40) and an address of 0. The shutdown indication cycle must be acknowledged by READY# asserted. A shutdown 80386SX resumes execution when NMI or RESET is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 80386SX floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the 80386SX remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD and RESET are ignored.

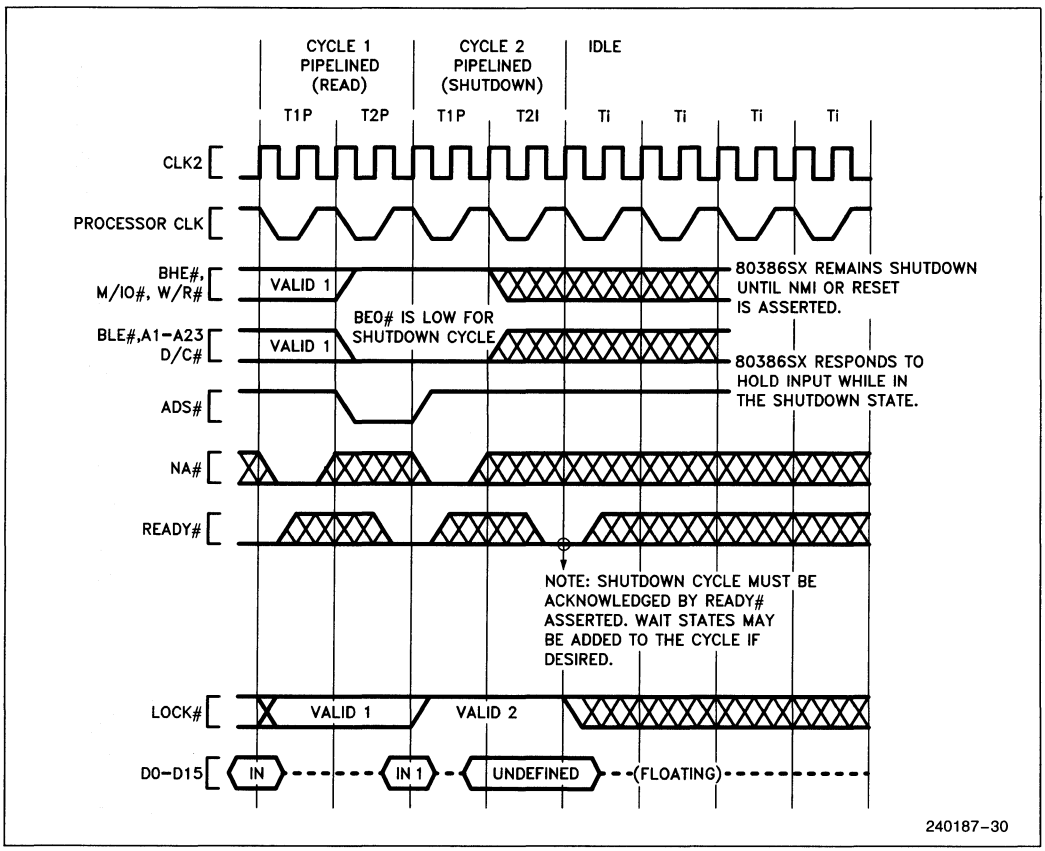


Figure 5.15. Example Shutdown Indication Cycle from Non- pipelined Cycle

T_h may be entered from a bus idle state as in Figure 5.16 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5.17 and 5.18.

T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 5.16 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 5.17 and 5.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h, the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 80386SX is reset before T_h is exited.

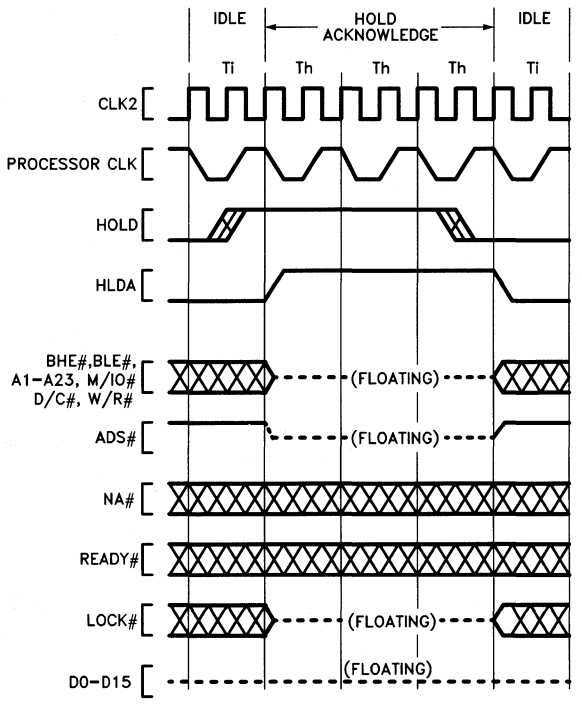
RESET DURING HOLD ACKNOWLEDGE

RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the 80386SX drives its pins to defined states during reset, as in **Table 5.5 Pin State During Reset**, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 80386SX enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 80386SX would otherwise perform its first bus cycle.

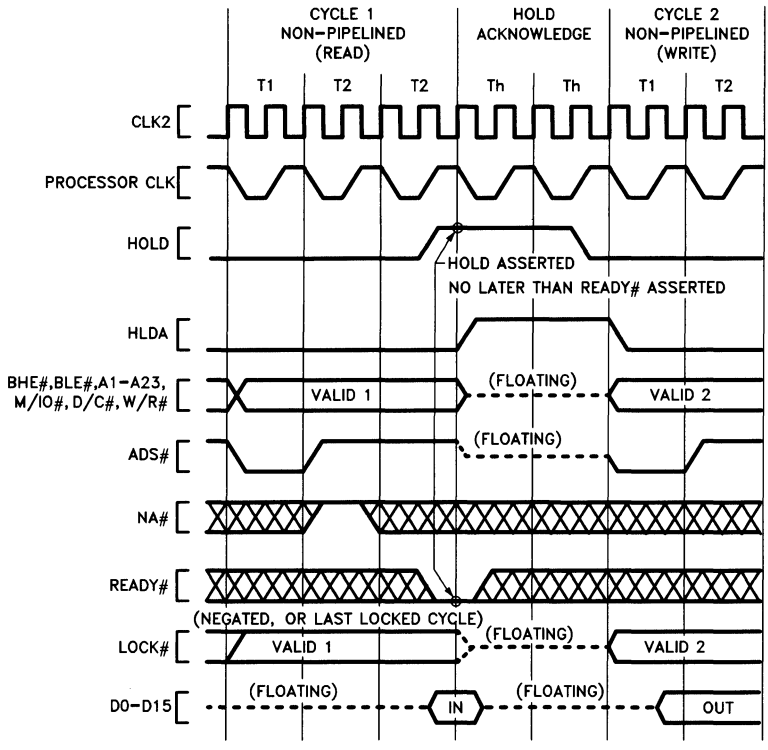
BUS ACTIVITY DURING AND FOLLOWING RESET

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.



NOTE:
For maximum design flexibility the 80386SX has no internal pullup resistors on its outputs. Your design may require an external pullup on ADS# and other 80386SX outputs to keep them negated during float periods.

Figure 5.16. Requesting Hold from Idle Bus



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

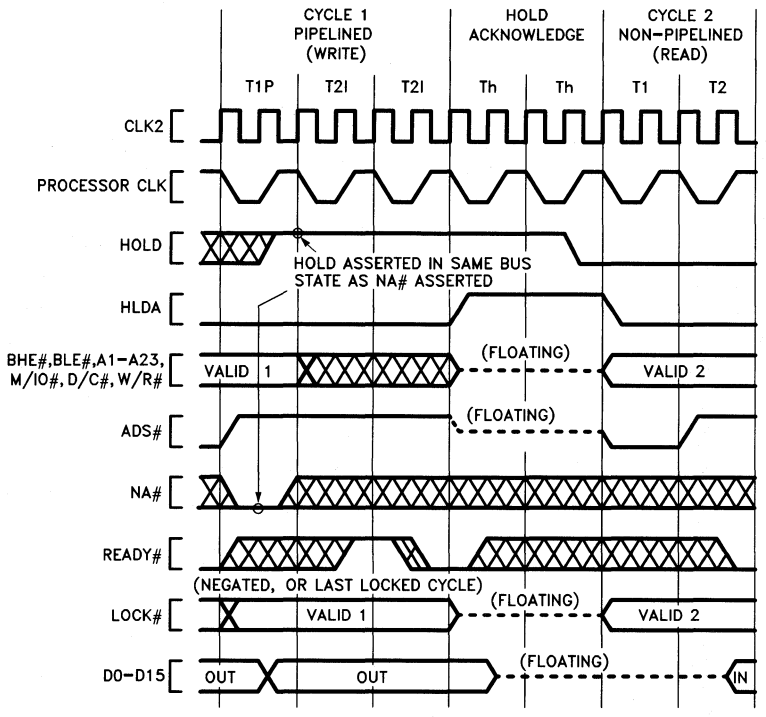
Figure 5.17. Requesting Hold from Active Bus (NA# inactive)

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 80386SX, and at least 80 CLK2 periods if 80386SX 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.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26}, the internal processor clock phase is defined at that time as illustrated by Figure 5.19 and Figure 7.7.

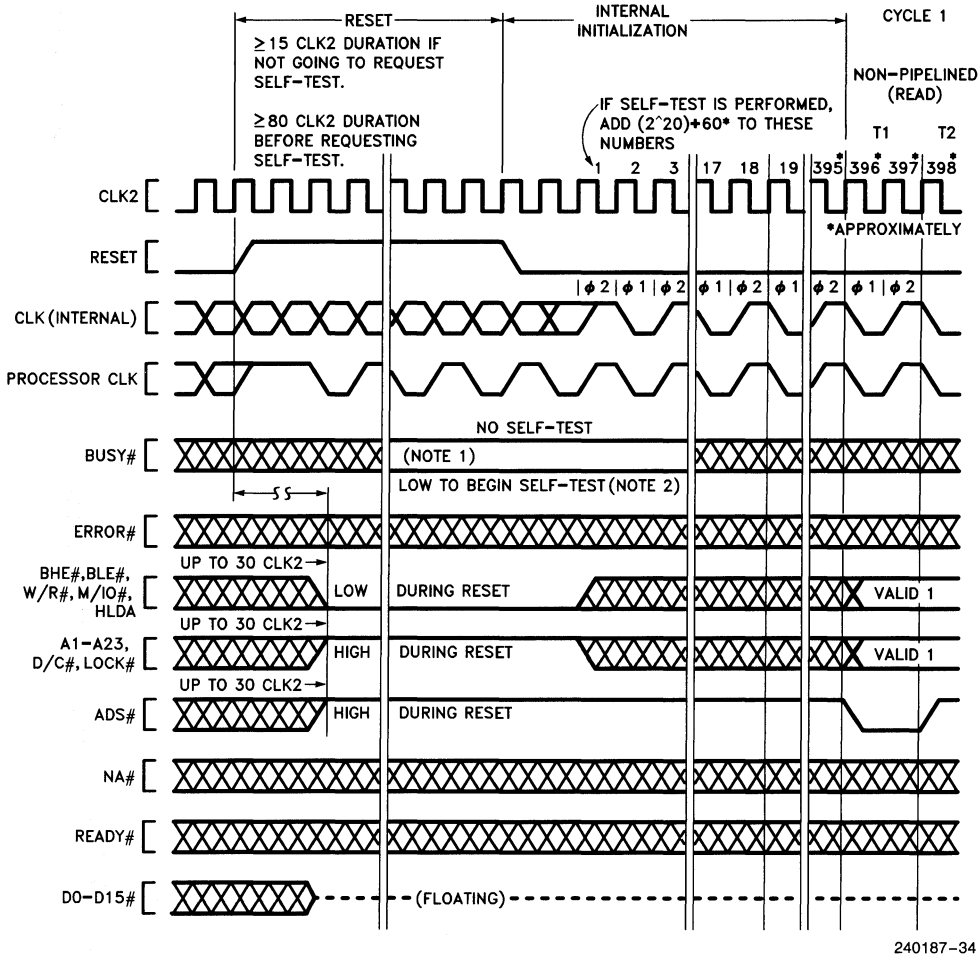
An 80386SX self-test may be requested at the time RESET goes inactive by having the BUSY# input at a LOW level as shown in Figure 5.19. The self-test requires (2^{20} + approximately 60) CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 80386SX attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 80386SX performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.



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.

Figure 5.18. Requesting Hold from Idle Bus (NA# active)



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 80386SX output remain in their reset state as shown here.

Figure 5.19. Bus Activity from Reset Until First Code Fetch

5.5 Self-test Signature

Upon completion of self-test (if self-test was requested by driving BUSY# LOW at the falling edge of RESET) the EAX register will contain a signature of 00000000H indicating the 80386SX passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all 80386SX revision levels. Any non-zero signature indicates the 80386SX unit is faulty.

5.6 Component and Revision Identifiers

To assist 80386SX users, the 80386SX after reset holds a component identifier and revision identifier in its DX register. The upper 8 bits of DX hold 23H as identification of the 80386SX component (The lower nibble, 03H, refers to the 80386 Architecture. The upper nibble, 02H, refers to the second member of the 80386 family). The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 80386SX revision identifier will track that of the 80386 where possible.

The revision identifier is intended to assist 80386SX users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 5.7. Component and Revision Identifier History

80386SX Stepping name	Revision Identifier
A0	04H
B	05H

5.7 Coprocessor Interfacing

The 80386SX provides an automatic interface for the Intel 80387SX numeric floating-point coprocessor. The 80387SX coprocessor uses an I/O mapped interface driven automatically by the 80386SX and assisted by three dedicated signals: BUSY#, ERROR# and PEREQ.

As the 80386SX begins supporting a coprocessor instruction, it tests the BUSY# and ERROR# signals to determine if the coprocessor can accept its next instruction. Thus, the BUSY# and ERROR# inputs eliminate the need for any 'preamble' bus cycles for communication between processor and coprocessor. The 80387SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 80386SX WAIT opcode (9BH) for 80387SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 80386SX based systems by 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 80386SX instructions for high-speed coprocessor communication. The BUSY# and ERROR# inputs of the 80386SX may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 80386SX WAIT opcode (9BH). The WAIT instruction will wait until the BUSY# input is inactive (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the ERROR# pin is active when the BUSY# goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the 80386SX's on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the 80386SX IOPL (I/O Privilege Level) mechanism.

The 80387SX numeric coprocessor interface is I/O mapped as shown in Table 5.8. Note that the 80387SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 80386SX supports the 80387SX coprocessor, the 80386SX automatically generates bus cycles to the coprocessor interface addresses.

Table 5.8. Numeric Coprocessor Port Addresses

Address in 80386SX I/O Space	80387SX Coprocessor Register
8000F8H	Opcode Register
8000FCH/8000FEH*	Operand Register

*Generated as 2nd bus cycle during D word transfer.

To correctly map the 80387SX registers to the appropriate I/O addresses, connect the CMD0 and CMD1 lines of the 80387SX as listed in Table 5.9.

Table 5.9. Connections for CMD0 and CMD1 Inputs for the 80387SX

Signal	Connection
CMD0	Connect to latched version of 80386SX A2 signal
CMD1	Connect to ground.

Software Testing for Coprocessor Presence

When software is used to test for coprocessor (80387SX) presence, it should use only the following coprocessor opcodes: FINIT, FNINIT, FSTCW mem, FSTSW mem and FSTSW AX. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 80386SX's CR0 register.

6.0 PACKAGE THERMAL SPECIFICATIONS

The 80386SX is specified for operation when case temperature is within the range of 0°C–85°C. The case temperature may be measured in any environment, to determine whether the 80386SX is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_j = T_c + P*\theta_{jc}$$

$$T_a = T_j - P*\theta_{ja}$$

$$T_c = T_a + P*[\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in table 6.1 for the 100 lead fine pitch. θ_{ja} is given at various airflows. Table 6.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching 'fins' or a 'heat sink' to the package. P is calculated by using the maximum hot Icc.

7.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 80386SX, and its electrical specifications.

7.1 Power and Grounding

The 80386SX is implemented in CHMOS III technology and has modest power requirements. However, its high clock frequency and 47 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, 14 Vcc and 18 Vss pins separately feed functional units of the 80386SX.

Power and ground connections must be made to all external Vcc and GND pins of the 80386SX. On the circuit board, all Vcc pins should be connected on a Vcc plane and all Vss pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 80386SX. The 80386SX driving its 24-bit address bus and 16-bit data bus 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 80386SX and decoupling capacitors as much as possible.

Table 6.1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}.

Package	θjc	θ_{ja} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100 Lead Fine Pitch	7	33	27	24	21	18	17

Table 6.2: Maximum T_a at various airflows.

Package	T_A(°C) versus Airflow - ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100 Lead Fine Pitch	33	45	51	57	63	65

Max. T_A calculated at 5.0V and max I_{CC}.

Table 7.1. Recommended Resistor Pull-ups to Vcc

Pin	Signal	Pull-up Value	Purpose
16	ADS#	20 K-Ohm ± 10%	Lightly pull ADS# inactive during 80386SX hold acknowledge states
26	LOCK#	20 K-Ohm ± 10%	Lightly pull LOCK# inactive during 80386SX hold acknowledge states

RESISTOR RECOMMENDATIONS

The ERROR# and BUSY# inputs have internal pull-up resistors of approximately 20 K-Ohms and the PEREQ input has an internal pull-down resistor of approximately 20 K-Ohms built into the 80386SX to keep these signals inactive when the 80387SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 7.1 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. N/C pins should always remain **unconnected. Connection of N/C pins to Vcc or Vss will result in component malfunction or incompatibility with future steppings of the 80386SX.**

Particularly when not using interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND:

Pin	Signal
40	INTR
38	NMI
4	HOLD

If not using address pipelining, connect pin 6, NA#, through a pull-up in the range of 20 K-Ohms to Vcc.

7.2 Maximum Ratings

Table 7.2. Maximum Ratings

Parameter	Maximum Rating
Storage temperature	−65 °C to 150 °C
Case temperature under bias	−65 °C to 110 °C
Supply voltage with respect to Vss	−.5V to 6.5V
Voltage on other pins	−.5V to (Vcc+.5)V

Table 7.2 gives stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in section **7.3, D.C. Specifications**, and section **7.4, A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 80386SX contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

7.3 D.C. Specifications

Functional operating range: Vcc = 5V ± 10%; T_{CASE} = 0°C to 85°C

Table 7.3. 80386SX D.C. Characteristics

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	Vcc+0.3	V	
V_{ILC}	CLK2 Input LOW Voltage	−0.3	+0.8	V	
V_{IHC}	CLK2 Input HIGH Voltage	Vcc−0.8	Vcc+0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 4mA: A_{23}–A_1,D_{15}–D_0 I_{OL} = 5mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA		0.45 0.45	V V	
V_{OH}	Output high voltage I_{OH} = −1mA: A_{23}–A_1,D_{15}–D_0 I_{OH} = −0.2 mA: A_{23}–A_1,D_{15}–D_0 I_{OH} = −0.9mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA I_{OH} = −0.18 mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA	2.4 Vcc−0.5 2.4 Vcc−0.5		V V V	
I_{LI}	Input leakage current (for all pins except PEREQ, BUSY# and ERROR#)		±15	μA	0V ≤ V_{IN} ≤ Vcc
I_{IH}	Input Leakage Current (PEREQ pin)		200	μA	V_{IH} = 2.4V, Note 1
I_{IL}	Input Leakage Current (BUSY# and ERROR# pins)		−400	μA	V_{IL} = 0.45V, Note 2
I_{LO}	Output leakage current		±15	μA	0.45V ≤ V_{OUT} ≤ Vcc
Icc	Supply current (CLK2 = 32 MHz)		400	mA	Icc typ = 300mA, Note 3
C_{IN}	Input capacitance		10	pF	Fc = 1 MHz, Note 4
C_{OUT}	Output or I/O capacitance		12	pF	Fc = 1 MHz, Note 4
C_{CLK}	CLK2 Capacitance		20	pF	Fc = 1 MHz, Note 4

Tested at the minimum operating frequency of the part.

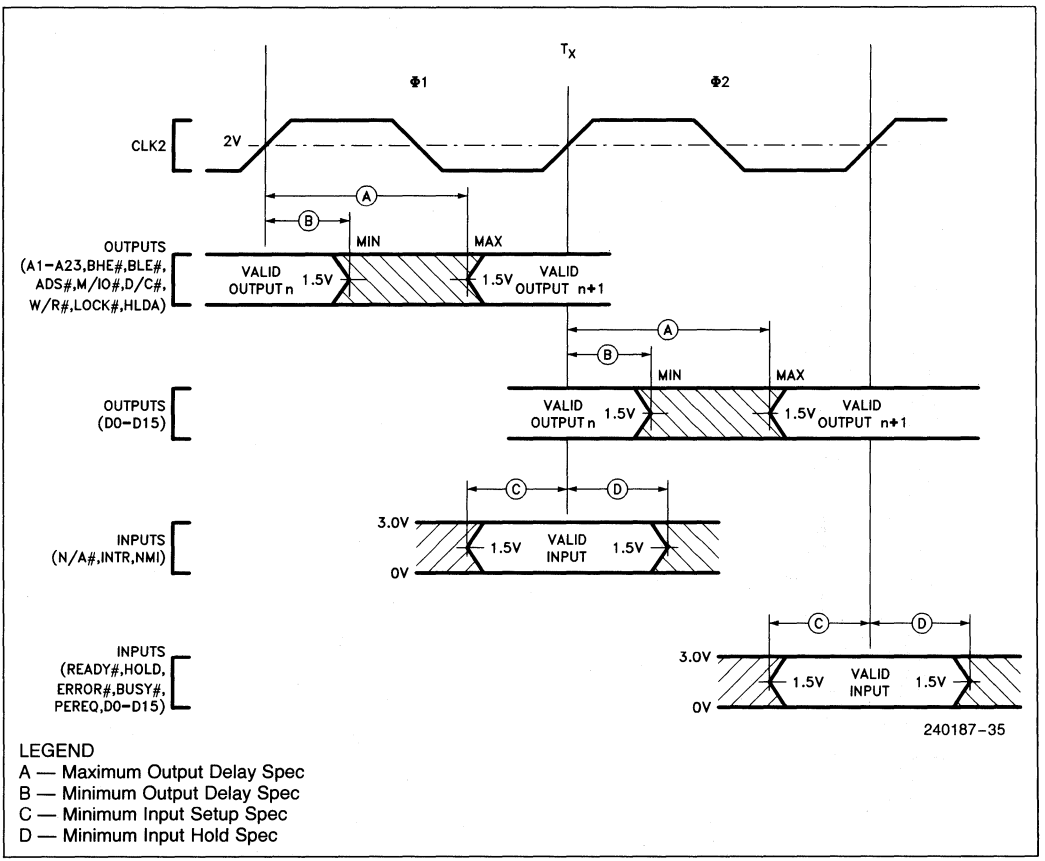
NOTES:
1. PEREQ input has an internal pull-down resistor.
2. BUSY# and ERROR# inputs each have an internal pull-up resistor.
3. Icc max measurement at worst case load, frequency, Vcc and temperature.
4. Not 100% tested.

7.4 A.C. Specifications

The A.C. specifications given in Table 7.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. spec measurement is defined by Figure 7.1. Inputs must be driven to the voltage levels indicated by Figure 7.1 when A.C. specifications are measured. 80386SX output delays are specified with minimum and maximum limits measured as shown. The minimum 80386SX delay times are hold times provided to external circuitry. 80386SX input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 80386SX operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BHE#, BLE#, A_{23}–A_1 and HLDA only change at the beginning of phase one. D_{15}–D_0 (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ and D_{15}–D_0 (read cycles) inputs are sampled at the beginning of phase one. The NA#, 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

Figure 7.1. Drive Levels and Measurement Points for A.C. Specifications

A.C. SPECIFICATONS TABLES

Functional operating range: Vcc = 5V ± 10%; T_{CASE} = 0°C to 85°C

Table 7.4. 80386SX A.C. Characteristics at 16 MHz

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating frequency	4	16	MHz		Half CLK2 Freq
t_1	CLK2 period	31	125	ns	7.3	
t_{2a}	CLK2 HIGH time	9		ns	7.3	at 2V(3)
t_{2b}	CLK2 HIGH time	5		ns	7.3	at (Vcc − 0.8)V(3)
t_{3a}	CLK2 LOW time	9		ns	7.3	at 2V(3)
t_{3b}	CLK2 LOW time	7		ns	7.3	at 0.8V(3)
t_4	CLK2 fall time		8	ns	7.3	(Vcc − 0.8)V to 0.8V(3)
t_5	CLK2 rise time		8	ns	7.3	0.8V to (Vcc − 0.8)V(3)
t_6	A_{23}–A_1 valid delay	4	36	ns	7.5	C_L = 120pF(4)
t_7	A_{23}–A_1 float delay	4	40	ns	7.6	(Note 1)
t_8	BHE#, BLE#, LOCK# valid delay	4	36	ns	7.5	C_L = 75pF(4)
t_9	BHE#, BLE#, LOCK# float delay	4	40	ns	7.6	(Note 1)
t_{10}	W/R#, M/IO#, D/C#, ADS# valid delay	6	33	ns	7.5	C_L = 75pF(4)
t_{11}	W/R#, M/IO#, D/C# ADS# float delay	6	35	ns	7.6	(Note 1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	40	ns	7.5	C_L = 120pF(4)
t_{13}	D_{15}–D_0 Write Data Float Delay	4	35	ns	7.6	(Note 1)
t_{14}	HLDA valid delay	6	33	ns	7.5	C_L = 75pF(4)
t_{15}	NA# setup time	5		ns	7.4	
t_{16}	NA# hold time	21		ns	7.4	
t_{19}	READY# setup time	19		ns	7.4	
t_{20}	READY# hold time	4		ns	7.4	
t_{21}	D_{15}–D_0 Read Data setup time	9		ns	7.4	
t_{22}	D_{15}–D_0 Read Data hold time	6		ns	7.4	
t_{23}	HOLD setup time	26		ns	7.4	
t_{24}	HOLD hold time	5		ns	7.4	
t_{25}	RESET setup time	13		ns	7.7	
t_{26}	RESET hold time	4		ns	7.7	

Functional operating range: Vcc = 5V ± 10%; T_{CASE} = 0°C to 85°C

Table 7.4. 80386SX A.C. Characteristics at 16 MHz (Continued)

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{27}	NMI, INTR setup time	16		ns	7.4	(Note 2)
t_{28}	NMI, INTR hold time	16		ns	7.4	(Note 2)
t_{29}	PEREQ, ERROR#, BUSY# setup time	16		ns	7.4	(Note 2)
t_{30}	PEREQ, ERROR#, BUSY# hold time	5		ns	7.4	(Note 2)

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 CLK2 period.
3: These are not tested. They are guaranteed by design characterization.
4: Tested with C_L set at 50 pf and derated to support the indicated distributed capacitive load. See figure 7.8 for the capacitive derating curve.

A.C. TEST LOADS

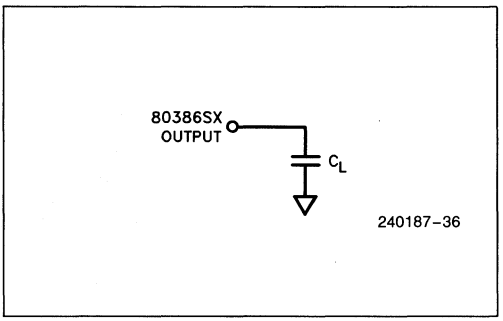


Figure 7.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

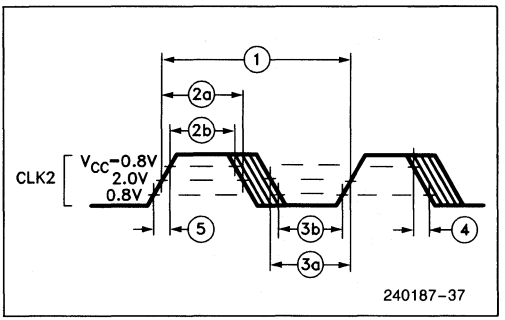


Figure 7.3. CLK2 Waveform

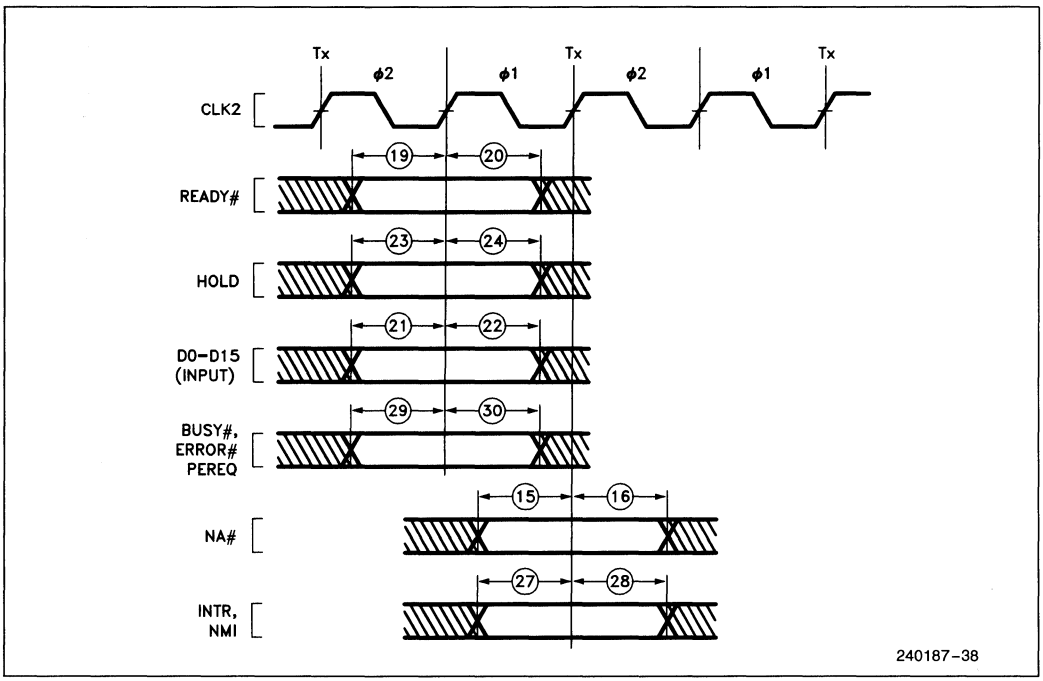


Figure 7.4. A.C. Timing Waveforms—Input Setup and Hold Timing

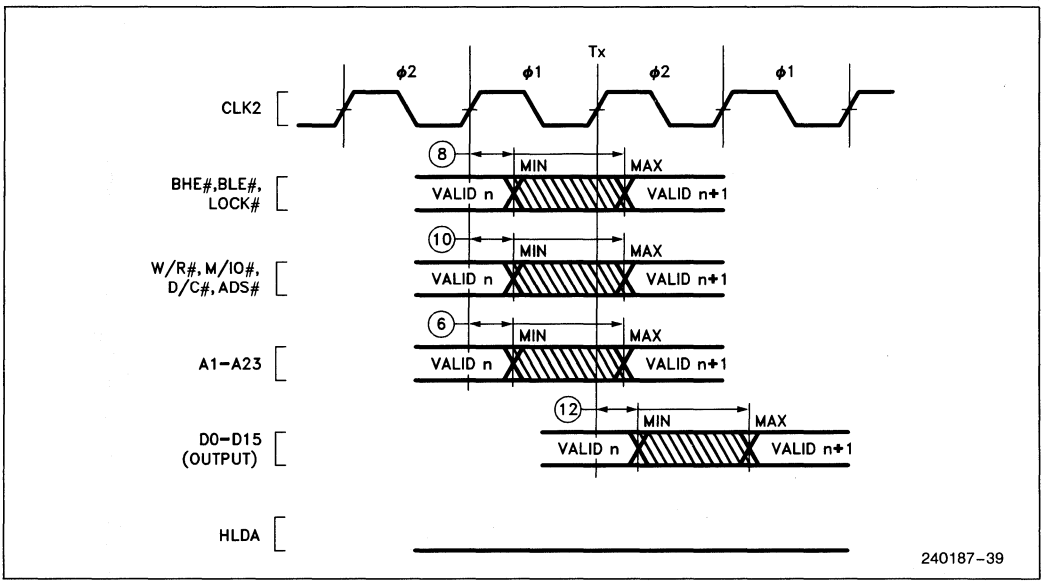


Figure 7.5. A.C. Timing Waveforms—Output Valid Delay Timing

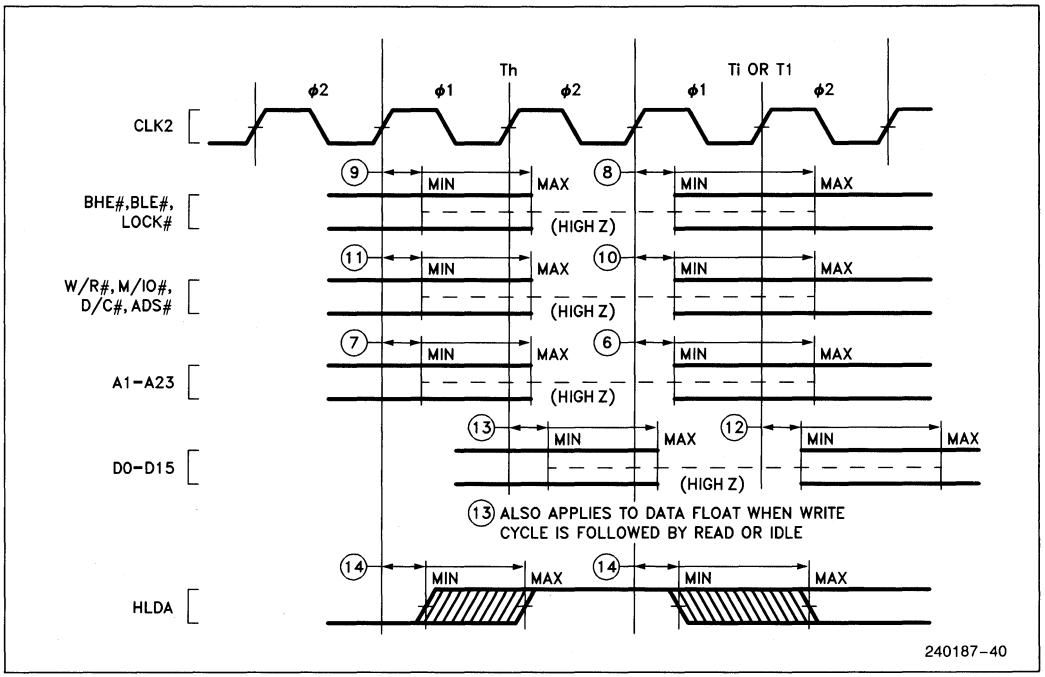


Figure 7.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing

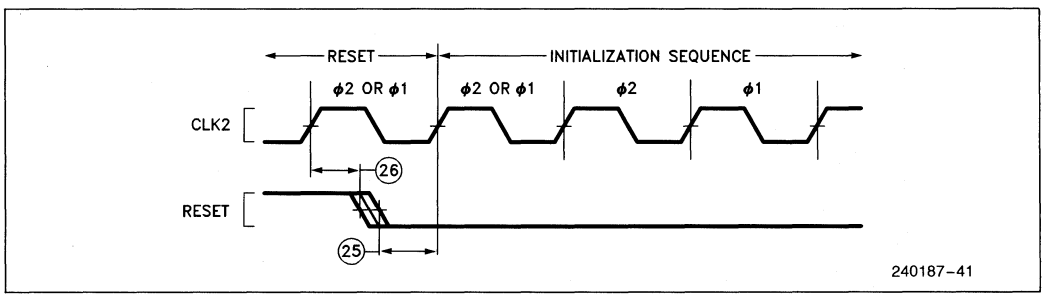


Figure 7.7. A.C. Timing Waveforms—RESET Setup and Hold Timing and Internal Phase

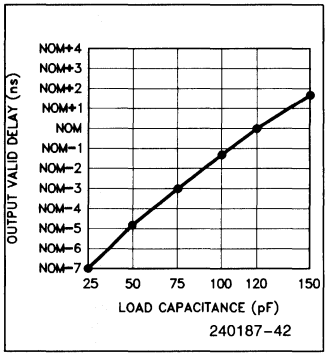


Figure 7.8. Capacitive Derating Curve

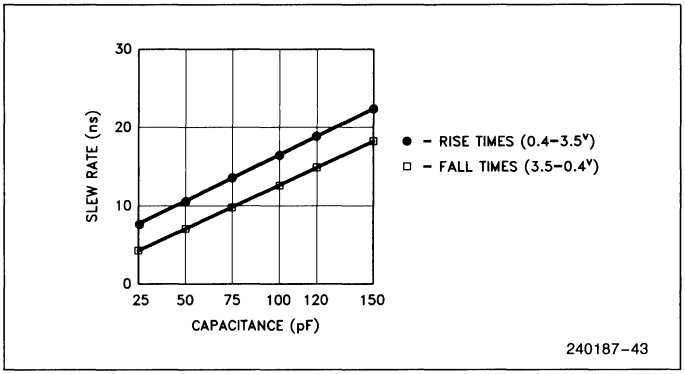


Figure 7.9. CMOS Level Slew Rates for Output Buffers

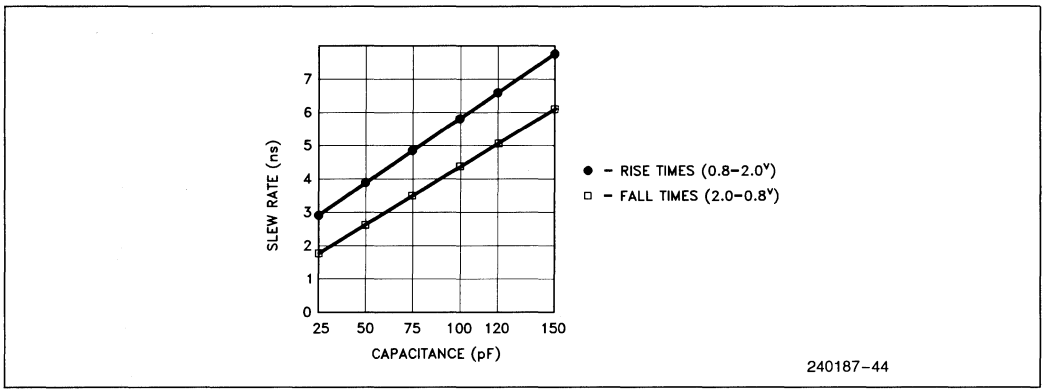


Figure 7.10. TTL Level Slew Rates for Output Buffers

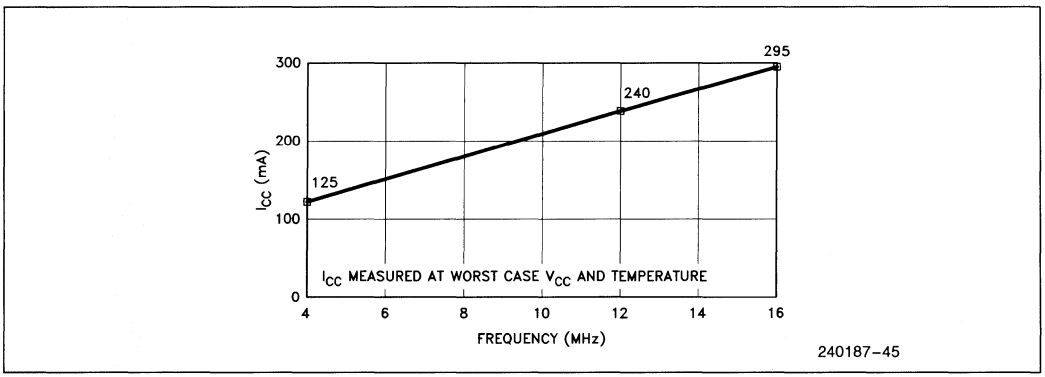


Figure 7.11. Typical Icc vs Frequency

7.5 Designing for ICE™-386SX Use (Preliminary Data)

The 80386SX in-circuit emulator product is ICE-386SX. This emulator is designed to connect to the user's target by an adapter that will replace the normal 80386SX component in a socket on the user's system. Because of the high operating frequency of 80386SX systems and of the ICE-386SX, there is no buffering between the 80386SX emulation processor in the ICE-386SX probe and the target system. A direct result of the non-buffered interconnect is that the ICE-386SX shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE hardware. In order for ICE-386SX to be functional in the user's system the design must satisfy the following restrictions:

1. The user bus controller must only drive the data bus during valid read cycles of the 80386SX or while the processor is in the hold state.
2. Before the user system drives the address bus, the system must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The RESET signal to the emulation processor is delayed in emulation by 2 or 4 CLK2 cycles (correct phase is guaranteed).

In addition to the above considerations, the ICE-386SX processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 80386SX system.

Capacitive loading: ICE-386SX adds up to 27 pF to each 80386SX signal.

Drive Requirements: ICE-386SX adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 80386SX emulation processor, which has standard drive and loading capability listed in Tables 7.3 and 74..

Power Requirements: For noise immunity and CMOS latch- up protection the ICE-386SX processor module is powered by the user system. The circuitry on the processor module draws up to 1.5 Amps plus the maximum 80386SX Icc from the user 80386SX socket.

80386SX Location and Orientation: The ICE-386SX processor module may require lateral clearance illustrated in Figure 7.12, viewed from above the 80386SX socket. The ICE-386SX processor module alone requires vertical clearance of 1.25 inches above the height of the surrounding circuitry. The optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figure 7.12, adds 0.5 inches to the vertical clearance requirement.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-386SX is susceptible to errors on the user's bus. The OIB allows the ICE-386SX to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 16 MHz.

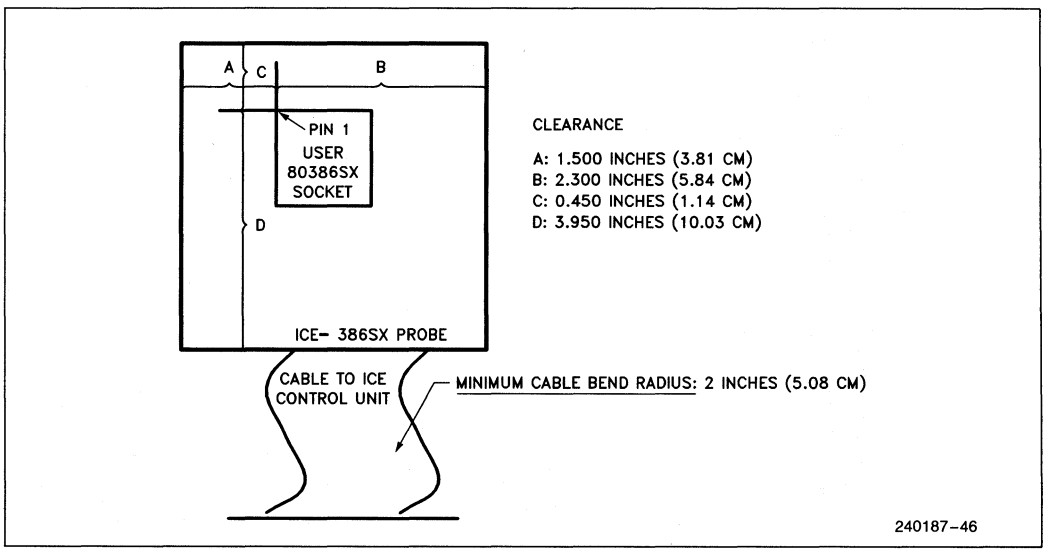


Figure 7.12: Preliminary ICE™-386SX Processor Module Clearance Requirements (inches)

8.0 DIFFERENCES BETWEEN THE 80386SX AND THE 80386

The following are the major differences between the 80386SX and the 80386:

1. The 80386SX generates byte selects on BHE# and BLE# (like the 8086 and 80286) to distinguish the upper and lower bytes on its 16-bit data bus. The 80386 uses four byte selects, BE0#-BE3#, to distinguish between the different bytes on its 32-bit bus.
2. The 80386SX has no bus sizing option. The 80386 can select between either a 32-bit bus or a 16-bit bus by use of the BS16# input. The 80386SX has a 16-bit bus size.
3. The NA# pin operation in the 80386SX is identical to that of the NA# pin on the 80386 with one exception: the 80386's NA# pin cannot be activated on 16-bit bus cycles (where BS16# is LOW in the 80386 case), whereas NA# can be activated on any 80386SX bus cycle.
4. The contents of all 80386SX registers at reset are identical to the contents of the 80386 registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

   in 80386, after reset: DH = 3 indicates 80386; DL = revision number;

   in 80386SX, after reset: DH = 23H indicates 80386SX; DL = revision number.
5. The 80386 uses A_{31} and M/IO# as selects for the numerics coprocessor. The 80386SX uses A_{23} and M/IO# as selects.
6. The 80386 prefetch unit fetches code in four-byte units. The 80386SX prefetch unit reads two bytes as one unit (like the 80286). In BS16 mode, the 80386 takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 80386 will fetch all four bytes before addressing the new request.
7. Both 80386 and 80386SX have the same logical address space. The only difference is that the 80386 has a 32-bit physical address space and the 80386SX has a 24-bit physical address space. The 80386SX has a physical memory address space of up to 16 megabytes instead of the 4 gigabytes available to the 80386. Therefore, in 80386SX systems, the operating system must be aware of this physical memory limit and should allocate memory for applications programs within this limit. If an 80386 system uses only the lower 16 megabytes of physical address, then there will be no extra effort required to migrate 80386 software to the 80386SX. Any application which uses more than 16 megabytes of memory can run on the 80386SX if the operating system utilizes the 80386SX's paging mechanism. In spite of this difference in physical address space, the 80386SX and 80386 CPUs can run the same operating systems and applications within their respective physical memory constraints.

9.0 INSTRUCTION SET

This section describes the 80386SX instruction set. Table 9.1 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 80386SX instructions.

9.1 80386SX Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 9.1 below, by the processor clock period (e.g. 62.5 ns for an 80386SX operating at 16 MHz). The actual clock count of an 80386SX program will average 5% more than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, 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 all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses

— If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:
  2* clocks for read or write
  4** clocks for read and write

— If instructions accesses a 32-bit operand on odd address add:
  4* clocks for read or write
  8** clocks for read and write

Wait States

Wait states add 1 clock per wait state to instruction execution for each data access.

Table 8-1. 80386SX Instruction Set Clock Count Summary

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
GENERAL DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1000100w mod reg r/m	2/2	2/2*	b	h
Register/Memory to Register	1000101w mod reg r/m	2/4	2/4*	b	h
Immediate to Register/Memory	1100011w mod000 r/m immediate data	2/2	2/2*	b	h
Immediate to Register (short form)	1011 w reg immediate data	2	2		
Memory to Accumulator (short form)	1010000w full displacement	4*	4*	b	h
Accumulator to Memory (short form)	1010001w full displacement	2*	2*	b	h
Register Memory to Segment Register	10001110 mod sreg3 r/m	2/5	22/23	b	h, i, j
Segment Register to Register/Memory	10001100 mod sreg3 r/m	2/2	2/2	b	h
MOVSX = Move With Sign Extension					
Register From Register/Memory	00001111 1011111w mod reg r/m	3/6*	3/6*	b	h
MOVZX = Move With Zero Extension					
Register From Register/Memory	00001111 1011011w mod reg r/m	3/6*	3/6*	b	h
PUSH = Push:					
Register/Memory	11111111 mod110 r/m	5/7*	7/9*	b	h
Register (short form)	01010 reg	2	4	b	h
Segment Register (ES, CS, SS or DS) (short form)	000 sreg2 110	2	4	b	h
Segment Register (ES, CS, SS, DS, FS or GS)	00001111 10 sreg3 000	2	4	b	h
Immediate	011010s0 immediate data	2	4	b	h
PUSHA = Push All	01100000	18	34	b	h
POP = Pop					
Register/Memory	10001111 mod000 r/m	5/7	7/9	b	h
Register (short form)	01011 reg	6	6	b	h
Segment Register (ES, CS, SS or DS) (short form)	000 sreg 2 111	7	25	b	h, i, j
Segment Register (ES, CS, SS or DS), FS or GS	00001111 10 sreg 3 001	7	25	b	h, i, j
POPA = Pop All	01100001	24	40	b	h
XCHG = Exchange					
Register/Memory With Register	1000011w mod reg r/m	3/5**	3/5**	b, f	f, h
Register With Accumulator (short form)	10010 reg	3	3		
IN = Input from:		**Clk Count Virtual 8086 Mode**			
Fixed Port	1110010w port number	†26	12*	6*/26*	s/t,m
Variable Port	1110110w	†27	13*	7*/27*	s/t,m
OUT = Output to:					
Fixed Port	1110011w port number	†24	10*	4*/24*	s/t,m
Variable Port	1110111w	†25	11*	5*/25*	s/t,m
LEA = Load EA to Register	10001101 mod reg r/m		2	2	

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SEGMENT CONTROL					
LDS = Load Pointer to DS	11000101 | mod reg r/m	7*	28*	b	h, i, j
LES = Load Pointer to ES	11000100 | mod reg r/m	7*	28*	b	h, i, j
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	7*	31*	b	h, i, j
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	7*	28*	b	h, i, j
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	7*	28*	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	11111000	2	2		
CLD = Clear Direction Flag	11111100	2	2		
CLI = Clear Interrupt Enable Flag	11111010	8	8		m
CLTS = Clear Task Switched Flag	00001111 | 00000110	5	5	c	l
CMC = Complement Carry Flag	11110101	2	2		
LAHF = Load AH into Flag	10011111	2	2		
POPF = Pop Flags	10011101	5	5	b	h, n
PUSHF = Push Flags	10011100	4	4	b	h
SAHF = Store AH into Flags	10011110	3	3		
STC = Set Carry Flag	11111001	2	2		
STD = Set Direction Flag	11111001 | 11111101	2	2		
STI = Set Interrupt Enable Flag	11111011	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	000000dw | mod reg r/m	2	2		
Register to Memory	0000000w | mod reg r/m	7**	7**	b	h
Memory to Register	0000001w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw | mod 000 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0000010w | immediate data	2	2		
ADC = Add With Carry					
Register to Register	000100dw | mod reg r/m	2	2		
Register to Memory	0001000w | mod reg r/m	7**	7**	b	h
Memory to Register	0001001w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw | mod 010 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0001010w | immediate data	2	2		
INC = Increment					
Register/Memory	1111111w | mod 000 r/m	2/6**	2/6**	b	h
Register (short form)	01000 reg	2	2		
SUB = Subtract					
Register from Register	001010dw | mod reg r/m	2	2		

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
Register from Memory	0010100w | mod reg r/m	7**	7**	b	h
Memory from Register	0010101w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000sw | mod 101 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0010110w | immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	000110dw | mod reg r/m	2	2		
Register from Memory	0001100w | mod reg r/m	7**	7**	b	h
Memory from Register	0001101w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000sw | mod 011 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0001110w | immediate data	2	2		
DEC Decrement					
Register/Memory	1111111w | reg 001 r/m	2/6	2/6	b	h
Register (short form)	01001 reg	2	2		
CMP Compare					
Register with Register	001110dw | mod reg r/m	2	2		
Memory with Register	0011100w | mod reg r/m	5*	5*	b	h
Register with Memory	0011101w | mod reg r/m	6*	6*	b	h
Immediate with Register/Memory	100000sw | mod 111 r/m | immediate data	2/5*	2/5*	b	h
Immediate with Accumulator (short form)	0011110w | immediate data	2	2		
NEG Change Sign	1111011w | mod 011 r/m	2/6*	2/6*	b	h
AAA ASCII Adjust for Add	00110111	4	4		
AAS ASCII Adjust for Subtract	00111111	4	4		
DAA Decimal Adjust for Add	00100111	4	4		
DAS Decimal Adjust for Subtract	00101111	4	4		
MUL Multiply (unsigned)					
Accumulator with Register/Memory	1111011w | mod 100 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		12-41/17-46*	12-41/17-46*	b, d	d, h
IMUL Integer Multiply (signed)					
Accumulator with Register/Memory	1111011w | mod 101 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		12-41/17-46*	12-41/17-46*	b, d	d, h
Register with Register/Memory	00001111 | 10101111 | 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		12-41/17-46*	12-41/17-46*	b, d	d, h
Register/Memory with Immediate to Register	011010s1 | mod reg r/m | immediate data				
-Word		13-26	13-26/14-27	b, d	d, h
-Doubleword		13-42	13-42/16-45	b, d	d, h

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
DIV = Divide (Unsigned)					
Accumulator by Register/Memory	1111011w mod110 r/m				
Divisor—Byte		14/17	14/17	b,e	e,h
—Word		22/25	22/25	b,e	e,h
—Doubleword		38/43	38/43	b,e	e,h
IDIV = Integer Divide (Signed)					
Accumulator By Register/Memory	1111011w mod111 r/m				
Divisor—Byte		19/22	19/22	b,e	e,h
—Word		27/30	27/30	b,e	e,h
—Doubleword		43/48	43/48	b,e	e,h
AAD = ASCII Adjust for Divide	11010101 00001010	19	19		
AAM = ASCII Adjust for Multiply	11010100 00001010	17	17		
CBW = Convert Byte to Word	10011000	3	3		
CWD = Convert Word to Double Word	10011001	2	2		
LOGIC					
Shift Rotate Instructions					
Not Through Carry **(ROL, ROR, SAL, SAR, SHL,** and **SHR)**					
Register/Memory by 1	1101000w mod TTT r/m	3/7**	3/7**	b	h
Register/Memory by CL	1101001w mod TTT r/m	3/7*	3/7*	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m immed 8-bit data	3/7*	3/7*	b	h
Through Carry **(RCL** and **RCR)**					
Register/Memory by 1	1101000w mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by CL	1101001w mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m immed 8-bit data	9/10*	9/10*	b	h
	TTT Instruction: 000 ROL; 001 ROR; 010 RCL; 011 RCR; 100 SHL/SAL; 101 SHR; 111 SAR				
SHLD = Shift Left Double					
Register/Memory by Immediate	00001111 10100100 mod reg r/m immed 8-bit data	3/7**	3/7**		
Register/Memory by CL	00001111 10100101 mod reg r/m	3/7**	3/7**		
SHRD = Shift Right Double					
Register/Memory by Immediate	00001111 10101100 mod reg r/m immed 8-bit data	3/7**	3/7**		
Register/Memory by CL	00001111 10101101 mod reg r/m	3/7**	3/7**		
AND = And					
Register to Register	001000dw mod reg r/m	2	2		

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
LOGIC (Continued)						
Register to Memory	0010000w mod reg r/m		7**	7**	b	h
Memory to Register	0010001w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 100 r/m immediate data		2/7*	2/7**	b	h
Immediate to Accumulator (Short Form)	0010010w immediate data		2	2		
TEST = And Function to Flags, No Result						
Register/Memory and Register	1000010w mod reg r/m		2/5*	2/5*	b	h
Immediate Data and Register/Memory	1111011w mod 000 r/m immediate data		2/5*	2/5*	b	h
Immediate Data and Accumulator (Short Form)	1010100w immediate data		2	2		
OR = Or						
Register to Register	000010dw mod reg r/m		2	2		
Register to Memory	0000100w mod reg r/m		7**	7**	b	h
Memory to Register	0000101w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 001 r/m immediate data		2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0000110w immediate data		2	2		
XOR = Exclusive Or						
Register to Register	001100dw mod reg r/m		2	2		
Register to Memory	0011000w mod reg r/m		7**	7**	b	h
Memory to Register	0011001w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 110 r/m immediate data		2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0011010w immediate data		2	2		
NOT = Invert Register/Memory	1111011w mod 010 r/m		2/6**	2/6**	b	h
STRING MANIPULATION						
CMPS = Compare Byte Word	1010011w		10*	10*	b	h
INS = Input Byte/Word from DX Port	0110110w	†29		9*/29**	b	s/t, h, m
LODS = Load Byte/Word to AL/AX/EAX	1010110w		5	5*	b	h
MOVS = Move Byte Word	1010010w		7	7**	b	h
OUTS = Output Byte/Word to DX Port	0110111w	†28	14	8*/28*	b	s/t, h, m
SCAS = Scan Byte Word	1010111w		7*	7*	b	h
STOS = Store Byte/Word from AL/AX/EX	1010101w		4*	4*	b	h
XLAT = Translate String	11010111		5*	5*		h
REPEATED STRING MANIPULATION Repeated by Count in CX or ECX						
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w		5 + 9n**	5 + 9n**	b	h

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
REPEATED STRING MANIPULATION (Continued)						
REPNE CMPS = Compare String						
(Find Match)	11110010 1010011w		5+9n**	5+9n**	b	h
REP INS = Input String	11110010 0110110w	†	13+6n*	7+6n*/ 27+6n*	b	s/t, h, m
REP LODS = Load String	11110010 1010110w		5+6n*	5+6n*	b	h
REP MOVS = Move String	11110010 1010010w		7+4n*	7+4n**	b	h
REP OUTS = Output String	11110010 0110111w	†	12+5n*	6+5n*/ 26+5n*	b	s/t, h, m
REPE SCAS = Scan String						
(Find Non-AL/AX/EAX)	11110011 1010111w		5+8n*	5+8n*	b	h
REPNE SCAS = Scan String						
(Find AL/AX/EAX)	11110010 1010111w		5+8n*	5+8n*	b	h
REP STOS = Store String	11110010 1010101w		5+5n*	5+5n*	b	h
BIT MANIPULATION						
BSF = Scan Bit Forward	00001111 10111100 mod reg r/m		10+3n*	10+3n**	b	h
BSR = Scan Bit Reverse	00001111 10111101 mod reg r/m		10+3n*	10+3n**	b	h
BT = Test Bit						
Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data		3/6*	3/6*	b	h
Register/Memory, Register	00001111 10100011 mod reg r/m		3/12*	3/12*	b	h
BTC = Test Bit and Complement						
Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10111011 mod reg r/m		6/13*	6/13*	b	h
BTR = Test Bit and Reset						
Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10110011 mod reg r/m		6/13*	6/13*	b	h
BTS = Test Bit and Set						
Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10101011 mod reg r/m		6/13*	6/13*	b	h
CONTROL TRANSFER						
CALL = Call						
Direct Within Segment	11101000 full displacement		7+m*	9+m*	b	r
Register/Memory						
Indirect Within Segment	11111111 mod 010 r/m		7+m*/10+m*	9+m/ 12+m*	b	h, r
Direct Intersegment	10011010 unsigned full offset, selector		17+m*	42+m*	b	j,k,r

NOTE:
† 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 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			64+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)			98+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)			106+8x+m		h,j,k,r
From 286 Task to 286 TSS			285		h,j,k,r
From 286 Task to 386 TSS			310		h,j,k,r
From 286 Task to Virtual 8086 Task (386 TSS)			229		h,j,k,r
From 386 Task to 286 TSS			285		h,j,k,r
From 386 Task to 386 TSS			392		h,j,k,r
From 386 Task to Virtual 8086 Task (386 TSS)			309		h,j,k,r
Indirect Intersegment	11111111 mod011 r/m		46+m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)					
Via Call Gate to Same Privilege Level			68+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)			102+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)			110+8x+m		h,j,k,r
From 286 Task to 286 TSS					h,j,k,r
From 286 Task to 386 TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 TSS)					h,j,k,r
From 386 Task to 286 TSS					h,j,k,r
From 386 Task to 386 TSS			399		h,j,k,r
From 386 Task to Virtual 8086 Task (386 TSS)					h,j,k,r
JMP = Unconditional Jump					
Short	11101011 8-bit displacement		7+m		r
Direct within Segment	11101001 full displacement		7+m		r
Register/Memory Indirect within Segment	11111111 mod100 r/m		9+m/14+m	b	h,r
Direct Intersegment	11101010 unsigned full offset, selector		31+m		j,k,r
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			53+m		h,j,k,r
From 286 Task to 286 TSS					h,j,k,r
From 286 Task to 386 TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 TSS)					h,j,k,r
From 386 Task to 286 TSS					h,j,k,r
From 386 Task to 386 TSS					h,j,k,r
From 386 Task to Virtual 8086 Task (386 TSS)			395		h,j,k,r
Indirect Intersegment	11111111 mod101 r/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 286 Task to 286 TSS					h,j,k,r
From 286 Task to 386 TSS					h,j,k,r
From 286 Task to Virtual 8086 Task (386 TSS)					h,j,k,r
From 386 Task to 286 TSS					h,j,k,r
From 386 Task to 386 TSS			328		h,j,k,r
From 386 Task to Virtual 8086 Task (386 TSS)					h,j,k,r

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
RET = Return from CALL:					
Within Segment	11000011		12+m	b	g, h, r
Within Segment Adding Immediate to SP	11000010 | 16-bit displ		12+m	b	g, h, r
Intersegment	11001011		36+m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010 | 16-bit displ		36+m	b	g, h, j, k, r
Protected Mode Only (RET):					
to Different Privilege Level					
Intersegment			72		h, j, k, r
Intersegment Adding Immediate to SP			72		h, j, k, r
CONDITIONAL JUMPS					
NOTE: Times Are Jump "Taken or Not Taken"					
JO = Jump on Overflow					
8-Bit Displacement	01110000 | 8-bit displ	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 displ	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 displ	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 displ	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 displ	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 displ	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 displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000110 | full displacement	7+m or 3	7+m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-Bit Displacement	01110111 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000111 | full displacement	7+m or 3	7+m or 3		r
JS = Jump on Sign					
8-Bit Displacement	01111000 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001000 | full displacement	7+m or 3	7+m or 3		r

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	01111001 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001001 | full displacement	7+m or 3	7+m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	01111010 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001010 | full displacement	7+m or 3	7+m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	01111011 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001011 | full displacement	7+m or 3	7+m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	01111100 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001100 | full displacement	7+m or 3	7+m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	01111101 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001101 | full displacement	7+m or 3	7+m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	01111110 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001110 | full displacement	7+m or 3	7+m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	01111111 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001111 | full displacement	7+m or 3	7+m or 3		r
JCXZ = Jump on CX Zero	11100011 | 8-bit displ	9+m or 5	9+m or 5		r
JECXZ = Jump on ECX Zero	11100011 | 8-bit displ	9+m or 5	9+m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times	11100010 | 8-bit displ	11+m	11+m		r
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displ	11+m	11+m		r
LOOPNZ/LOOPNE = Loop While Not Zero	11100000 | 8-bit displ	11+m	11+m		r
CONDITIONAL BYTE SET NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	00001111 | 10010000 | mod000 r/m	4/5*	4/5*		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	00001111 | 10010001 | mod000 r/m	4/5*	4/5*		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	00001111 | 10010010 | mod000 r/m	4/5*	4/5*		h

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL BYTE SET (Continued)					
SETNB = Set Byte on Not Below/Above or Equal					
To Register/Memory	00001111 | 10010011 | mod000 r/m	4/5*	4/5*		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 | 10010100 | mod000 r/m	4/5*	4/5*		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 | 10010101 | mod000 r/m	4/5*	4/5*		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 | 10010110 | mod000 r/m	4/5*	4/5*		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 | 10010111 | mod000 r/m	4/5*	4/5*		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 | 10011000 | mod000 r/m	4/5*	4/5*		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 | 10011001 | mod000 r/m	4/5*	4/5*		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 | 10011010 | mod000 r/m	4/5*	4/5*		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 | 10011011 | mod000 r/m	4/5*	4/5*		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 | 10011100 | mod000 r/m	4/5*	4/5*		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 | 01111101 | mod000 r/m	4/5*	4/5*		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 | 10011110 | mod000 r/m	4/5*	4/5*		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 | 10011111 | mod000 r/m	4/5*	4/5*		h
ENTER = Enter Procedure	11001000 | 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		14	14	b	h
L > 1		17 + 8(n − 1)	17 + 8(n − 1)	b	h
LEAVE = Leave Procedure	11001001	4	4	b	h

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 | type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set	11001110				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	
Bound = Interrupt 5 if Detect Value Out of Range	01100010 | 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			71		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			438		g, j, k, r
From 286 Task to 386 TSS via Task Gate			465		g, j, k, r
From 286 Task to virt 8086 md via Task Gate			382		g, j, k, r
From 386 Task to 286 TSS via Task Gate			440		g, j, k, r
From 386 Task to 386 TSS via Task Gate			467		g, j, k, r
From 386 Task to virt 8086 md via Task Gate			384		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			445		g, j, k, r
From virt 8086 md to 386 TSS via Task Gate			472		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			275		
INT: TYPE 3					
Via Interrupt or Trap Gate to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			382		g, j, k, r
From 286 Task to 386 TSS via Task Gate			409		g, j, k, r
From 286 Task to Virt 8086 md via Task Gate			326		g, j, k, r
From 386 Task to 286 TSS via Task Gate			384		g, j, k, r
From 386 Task to 386 TSS via Task Gate			411		g, j, k, r
From 386 Task to Virt 8086 md via Task Gate			328		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			389		g, j, k, r
From virt 8086 md to 386 TSS via Task Gate			416		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		
INTO:					
Via Interrupt or Trap Grate to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			384		g, j, k, r
From 286 Task to 386 TSS via Task Gate			411		g, j, k, r
From 286 Task to virt 8086 md via Task Gate			328		g, j, k, r
From 386 Task to 286 TSS via Task Gate			386		g, j, k, r
From 386 Task to 386 TSS via Task Gate			413		g, j, k, r
From 386 Task to virt 8086 md via Task Gate			329		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			391		g, j, k, r
From virt 8086 md to 386 TSS via Task Gate			418		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS (Continued)					
BOUND:					
Via Interrupt or Trap Gate to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			358		g, j, k, r
From 286 Task to 386 TSS via Task Gate			388		g, j, k, r
From 268 Task to virt 8086 Mode via Task Gate			335		g, j, k, r
From 386 Task to 286 TSS via Task Gate			368		g, j, k, r
From 386 Task to 386 TSS via Task Gate			398		g, j, k, r
From 368 Task to virt 8086 Mode via Task Gate			347		g, j, k, r,
From virt 8086 Mode to 286 TSS via Task Gate			368		g, j, k, r
From virt 8086 Mode to 386 TSS via Task Gate			398		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		
INTERRUPT RETURN					
IRET = Interrupt Return	11001111	24			g, h, j, k, r
Protected Mode Only (IRET)					
To the Same Privilege Level (within task)			42		g, h, j, k, r
To Different Privilege Level (within task)			86		g, h, j, k, r
From 286 Task to 286 TSS			285		h, j, k, r
From 286 Task to 386 TSS			318		h, j, k, r
From 286 Task to Virtual 8086 Task			267		h, j, k, r
From 286 Task to Virtual 8086 Mode (within task)			113		
From 386 Task to 286 TSS			324		h, j, k, r
From 386 Task to 386 TSS			328		h, j, k, r
From 386 Task to Virtual 8086 Task			377		h, j, k, r
From 386 Task to Virtual 8086 Mode (within task)			113		
PROCESSOR CONTROL					
HLT = HALT	11110100	5	5		I
MOV = Move to and From Control/Debug/Test Registers					
CR0/CR2/CR3 from register	00001111 00100010 11 eee reg	10/4/5	10/4/5		I
Register From CR0–3	00001111 00100000 11 eee reg	6	6		I
DR0–3 From Register	00001111 00100011 11 eee reg	22	22		I
DR6–7 From Register	00001111 00100011 11 eee reg	16	16		I
Register from DR6–7	00001111 00100001 11 eee reg	14	14		I
Register from DR0–3	00001111 00100001 11 eee reg	22	22		I
TR6–7 from Register	00001111 00100110 11 eee reg	12	12		I
Register from TR6–7	00001111 00100100 11 eee reg	12	12		I
NOP = No Operation	10010000	3	3		
WAIT = Wait until BUSY # pin is negated	10011011	6	6		

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	11011TTT | mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 80387SX data sheet for clock counts			h
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 010 r/m	11*	11*	b, c	h, l
LIDT = Load Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 011 r/m	11*	11*	b, c	h, l
LLDT = Load Local Descriptor					
Table Register to Register/Memory	00001111 | 00000000 | mod 010 r/m	N/A	20/24*	a	g, h, j, l
LMSW = Load Machine Status Word					
From Register/Memory	00001111 | 00000001 | mod 110 r/m	10/13	10/13*	b, c	h, l
LSL = Load Segment Limit					
From Register/Memory	00001111 | 00000011 | mod reg r/m				
Byte-Granular Limit		N/A	20/21*	a	g, h, j, p
Page-Granular Limit		N/A	25/26*	a	g, h, j, p
LTR = Load Task Register					
From Register/Memory	00001111 | 00000000 | mod 001 r/m	N/A	23/27*	a	g, h, j, l
SGDT = Store Global Descriptor					
Table Register	00001111 | 00000001 | mod 000 r/m	9*	9*	b, c	h
SIDT = Store Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 001 r/m	9*	9*	b, c	h
SLDT = Store Local Descriptor Table Register					
To Register/Memory	00001111 | 00000000 | mod 000 r/m	N/A	2/2*	a	h

Table 8-1. 80386SX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SMSW = **Store Machine Status Word**	00001111 | 00000001 | mod 100 r/m	2/2*	2/2*	b, c	h, l
STR = **Store Task Register** To Register/Memory	00001111 | 00000000 | mod 001 r/m	N/A	2/2*	a	h
VERR = **Verify Read Access** Register/Memory	00001111 | 00000000 | mod 100 r/m	N/A	10/11*	a	g, h, j, p
VERW = **Verify Write Access**	00001111 | 00000000 | mod 101 r/m	N/A	15/16*	a	g, h, j, p

INSTRUCTION NOTES FOR TABLE 8-1

Notes a through c apply to 80386SX Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).

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 80386SX Real Address Mode and 80386SX Protected Virtual Address Mode:

d. The 80386SX 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. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to 80386SX 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 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.

s/t. The instruction will execute in s clocks if CPL $\leq$ IOPL. If CPL > IOPL, the instruction will take t clocks.

DOMESTIC SALES OFFICES

ALABAMA

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

ARIZONA

†Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980

†Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815

CALIFORNIA

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

†Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040

†Intel Corp.
1510 Arden Way, Suite 101
Sacramento 95815
Tel: (916) 920-8096

†Intel Corp.
4350 Executive Drive
Suite 105
San Diego 92121
Tel: (619) 452-5880

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

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

COLORADO

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

†Intel Corp.*
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

†Intel Corp.
26 Mill Plain Road
2nd Floor
Danbury 06811
Tel: (203) 748-3130
TWX: 710-456-1199

FLORIDA

†Intel Corp.
6363 N.W. 6th Way, Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: 305-772-8193

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (305) 240-8000
FAX: 305-240-8097

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

GEORGIA

†Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

†Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60173
Tel: (312) 310-8031

INDIANA

†Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623

IOWA

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

KANSAS

†Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

†Intel Corp.*
7321 Parkway Drive South
Suite C
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

†Intel Corp.
7833 Walker Drive
Suite 550
Greenbelt 20770
Tel: (301) 441-1020

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (617) 692-3222
TWX: 710-343-6333

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8096

MINNESOTA

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

MISSOURI

†Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

†Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

†Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111
FAX: 201-740-0626

NEW MEXICO

†Intel Corp.
8500 Menaul Boulevard N.E.
Suite B 295
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.
127 Main Street
Binghamton 13905
Tel: (607) 773-0337
FAX: 607-723-2677

†Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391

†Intel Corp.*
300 Motor Parkway
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

†Intel Corp.
15 Myers Corner Road
Suite 2B
Hollowbrook Park
Wappinger Falls 12590
Tel: (914) 297-6161
TWX: 510-248-0060

NORTH CAROLINA

†Intel Corp.
5700 Executive Drive
Suite 213
Charlotte 28212
Tel: (704) 568-8966

†Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

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

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

OKLAHOMA

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

OREGON

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

PENNSYLVANIA

†Intel Corp.*
455 Pennsylvania Avenue
Suite 230
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

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

PUERTO RICO

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

TEXAS

†Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

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

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

UTAH

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

VIRGINIA

†Intel Corp.
1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

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

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

WISCONSIN

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

CANADA

BRITISH COLUMBIA

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

ONTARIO

†Intel Semiconductor of Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
TLX: 053-4115

†Intel Semiconductor of Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
TLX: 06983574
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of Canada, Ltd.
620 St. John Boulevard
Pointe Claire H9R 3K2
Tel: (514) 694-9130
TWX: 514-694-9134

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35816
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group Inc.
4825 University Square
Huntsville 35816
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 968-1461
TWX: 910-950-0077

Kierulff Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
FAX: 602-252-9109

Wyle Distribution Group
17855 N. Black Canyon Highway
Phoenix 85023
Tel: (602) 866-2888
FAX: 602-866-6937

CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
FAX: 818-772-8930

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-0862

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94089
Tel: (408) 745-6600
FAX: 408-743-4770

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
FAX: 714-838-4151

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
FAX: 714-754-6007

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94089
Tel: (408) 743-3300
FAX: 408-745-6679

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
FAX: 619-277-6136

†Hamilton/Avnet Electronics
9650 Desoto Avenue
Chatsworth 91311
Tel: (818) 700-1222, 6500
FAX: 818-700-6553

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150
FAX: 916-925-3478

†Hamilton/Avnet Electronics
3002 G Street
Ontario 91311
Tel: (714) 989-9411
FAX: 714-980-7129

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 90230
Tel: (213) 558-2458
FAX: 213-558-2248

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
FAX: 714-641-4122

CALIFORNIA (Cont'd.)

Kierulff Electronics, Inc.
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300
FAX: 714-821-8420

†Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 971-2600
FAX: 408-947-3432

†Kierulff Electronics, Inc.
14242 Chamber Rd.
Tustin 92680
Tel: (714) 731-5711
FAX: 714-669-4235

†Kierulff Electronics, Inc.
9800 Variel St.
Chatsworth 91311
Tel: (213) 725-0325
FAX: 818-407-0803

Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
FAX: 818-880-5510

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
FAX: 714-863-0473

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282
FAX: 916-638-1491

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-371-9592
FAX: 619-565-9171 ext. 274

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
FAX: 408-727-5896

Wyle Military
18910 Teller Avenue
Irvine 92715
Tel: (714) 851-9958
TWX: 310-371-9127
FAX: 714-851-8366

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-895-9038

COLORADO

Arrow Electronics, Inc.
1390 S. Potomac Street
Suite 136
Aurora 80012
Tel: (303) 696-1111

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
FAX: 203-797-2866

†Pioneer Northeast Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
350 Fairway Drive
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

Arrow Electronics, Inc.
1001 N.W. 62nd St., Ste. 108
Ft. Lauderdale 33309
Tel: (305) 475-4297
TWX: 510-955-9456

†Arrow Electronics, Inc.
1530 Bottlebrush N.E.
Palm Bay 32905
Tel: (305) 725-1480

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TLX: 510-956-3097

Hamilton/Avnet Electronics
3245 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
FAX: 305-628-3888 ext. 40

†Pioneer Electronics
337 N. Lake Blvd., Ste. 1000
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

Pioneer Electronics
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway
Suite A
Norcross 30071
Tel: (404) 449-8252
FAX: 404-242-6827

Hamilton/Avnet Electronics
5825 D. Peachtree Corners East
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer Electronics
3100 F. Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60173
Tel: (312) 397-3440
FAX: 312-397-3550

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

Kierulff Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (312) 250-0500
FAX: 312-250-0916

MTI Systems Sales
1100 West Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

INDIANA (Cont'd.)

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
FAX: 913-541-7951

Pioneer Electronics
10551 Lackman Rd.
Lenexa 66215
Tel: (913) 492-0500
FAX: 913-492-7832

KENTUCKY

Hamilton/Avnet Electronics
805-A Newtown Circle
Lexington 40511
Tel: (606) 259-1475
FAX: 606-252-3238

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Rd., Suite H
Rivers Center
Columbia 21046
Tel: (301) 995-6002
TWX: 710-236-9005
FAX: 301-381-3854

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
FAX: 301-995-3593

†Mesa Technology Corp.
9720 Patuxent Woods Dr.
Columbia 21046
Tel: (301) 720-5020
TWX: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 532-3701
TWX: 710-393-0382

Kierulff Electronics, Inc.
13 Fortune Dr.
Billerica 01821
Tel: (617) 667-8331
TWX: 710-390-1449
FAX: 617-663-1754

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48108
Tel: (313) 971-8220
FAX: 313-971-2633

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775
FAX: 313-522-2624

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921
FAX: 616-243-0028

MICHIGAN (Cont'd.)

Pioneer Electronics
4505 Broadmoor Ave. S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†Pioneer Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
FAX: 612-830-1856

Hamilton/Avnet Electronics
12400 White Water Drive
Minnetonka 55343
Tel: (612) 932-0600
FAX: 612-932-0613

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
FAX: 612-935-1921

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63146
Tel: (314) 567-6888
FAX: 314-567-1164

†Hamilton/Avnet Electronics
13743 Shoreline Court East
Earth City 63045
Tel: (314) 344-1200
FAX: 314-291-8889

Kierulff Electronics, Inc.
11804 Borman Dr.
St. Louis 63146
Tel: (314) 997-5966
FAX: 314-567-0860

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
FAX: 603-668-3484

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03103
Tel: (603) 624-9400
FAX: 603-624-2402

NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln Drive East
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-5632

†Arrow Electronics, Inc.
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-4962

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262
FAX: 609-751-8624

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
FAX: 201-575-3454

†MTI Systems Sales
37 Kulick Rd.
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Microcomputer System Technical Distributor Center

DOMESTIC DISTRIBUTORS

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

Arrow Electronics, Inc.
25 Hub Drive
Melville 11747
Tel: (516) 694-6800
TWX: 510-224-6126
FAX: 516-391-1401

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
FAX: 716-427-0735

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
FAX: 516-231-1072

Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 475-9130
FAX: 716-475-9119

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
FAX: 315-432-0740

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
FAX: 516-434-7426

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
FAX: 516-625-3039

†Pioneer Northeast Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

†Pioneer Northeast Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184
FAX: 516-921-2143

†Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greens Dairy Road
Raleigh 27604
Tel: (919) 876-3132
FAX: 919-876-3132, ext. 200

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27609
Tel: (919) 878-0819
TWX: 510-928-1836

NORTH CAROLINA (Cont'd.)

Pioneer Electronics
9801 A-Southern Pine Blvd.
Charlotte 28217
Tel: (704) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
FAX: 513-435-2049

†Arrow Electronics, Inc.,
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
FAX: 216-248-1106

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 439-6700
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge Rd., Bldg. A
Solon 44139
Tel: (216) 349-5100
FAX: 216-349-1894

†Pioneer Electronics
4433 Interpoint Blvd.
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211
FAX: 216-587-3906

OKLAHOMA

Arrow Electronics, Inc.
3158 S. 108 East Ave., Ste. 210
Tulsa 74146
Tel: (918) 665-7700
FAX: 918-665-7700

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
FAX: 503-645-0611

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
FAX: 503-636-1327

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
FAX: 503-640-5846

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000
FAX: 412-856-5777

Hamilton/Avnet Electronics
2800 Liberty Ave., Bldg. E
Pittsburgh 15222
Tel: (412) 281-4150
FAX: 412-281-8662

PENNSYLVANIA (Cont'd.)

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122
FAX: 412-963-8255

†Pioneer Electronics
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778
FAX: 215-674-3107

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: 214-248-7208

†Arrow Electronics, Inc.
10899 Kinghurst Dr.
Suite 100
Houston 77099
Tel: (713) 530-4700
FAX: 713-568-8518

†Arrow Electronics, Inc.
2227 W. Braker Lane
Austin 78758
Tel: (512) 835-4180
FAX: 512-832-9875

†Hamilton/Avnet Electronics
18074 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
FAX: 512-339-6232

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75038
Tel: (214) 550-6111
FAX: 214-550-6172

†Hamilton/Avnet Electronics
4850 Wright Road, Ste. 190
Stafford 77477
Tel: (713) 240-7733
FAX: 713-240-0582

Kierulff Electronics, Inc.
2010 Merritt Drive
Garland 75040
Tel: (214) 840-0110
FAX: 214-278-0928

†Pioneer Electronics
1826-D Kramer Lane
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
FAX: 713-988-1732

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
FAX: 801-974-9675

Kierulff Electronics, Inc.
1946 W. Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913
FAX: 801-972-0200

Wyle Distribution Group
1325 West 2200 South
Suite E
Salt Lake City 84119
Tel: (801) 974-9953
FAX: 801-972-2524

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
FAX: 206-746-3740

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
FAX: 206-643-0086

Wyle Distribution Group
1750 132nd Ave., N.E.
Bellevue 98005
Tel: (206) 453-8300
FAX: 206-453-4071

WISCONSIN

†Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
FAX: 414-784-9509

Kierulff Electronics, Inc.
2238-E W. Bluemound Rd.
Waukesha 53186
Tel: (414) 784-8160
FAX: 414-784-0409

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 250-9380
FAX: 403-250-1591

Zentronics
6815 8th Street, N.E., Ste. 100
Calgary T2E 7H7
Tel: (403) 295-8838
FAX: 403-295-8714

BRITISH COLUMBIA

Hamilton/Avnet Electronics
2550 Boundary Rd., Ste. 115
Burnaby V5M 323
Tel: (604) 437-8667
FAX: 604-437-4712

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

MANITOBA

Zentronics
60-1313 Border Street
Winnipeg R3H 0X4
Tel: (204) 694-1957
FAX: 204-633-9255

ONTARIO

Arrow Electronics Inc.
1093 Meyerside Drive
Unit 2
Mississauga L5T 1M4
Tel: (416) 672-7769
FAX: 416-672-0489

Arrow Electronics Inc.
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-5
Mississauga L4V 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

Hamilton/Avnet Electronics
3688 Nashua Dr.
Units 9 and 10
Mississauga L4V 1M5
Tel: (416) 677-0484
FAX: 416-677-0627

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7J5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
FAX: 416-451-8320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-6350

SASKATCHEWAN

Zentronics
173-1222 Alberta Avenue
Saskatoon S7K 1R4
Tel: (306) 955-2202, 2207
FAX: 306-244-3731

QUEBEC

†Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
FAX: 514-341-4821

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
FAX: 418-687-5348

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
FAX: 514-335-2481

Zentronics
817 McCafrey St.
St. Laurent H4T 1N4
Tel: (514) 737-9700
FAX: 514-737-5212

†Microcomputer System Technical Distributor Center

UNITED STATES, Intel Corporation
3065 Bowers Ave., Santa Clara, CA 95051
Tel: (408) 765-8080

JAPAN, Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi, Ibaraki, 300-26
Tel: 029747-8511

FRANCE, Intel Corporation S.a.r.P.
1, Rue Edison, BP 303, 78054 Saint-Quentin-en-Yvelines Cedex
Tel: (33) 1-30 57 70 00

UNITED KINGDOM, Intel Corporation (U.K.) Ltd.
Pipers Way, Swindon, Wiltshire, England SN3 1RJ
Tel: (0793) 696000

WEST GERMANY, Intel Semiconductor GmbH
Seidlstrasse 27, D-8000 Muenchen 2
Tel: (89) 53891

HONG KONG, Intel Semiconductor Ltd.
10/F East Tower, Bond Center, Queensway, Central
Tel: (5) 8444-555

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

Printed in U.S.A./0588/SM8409/6K/BL LD
Microprocessors