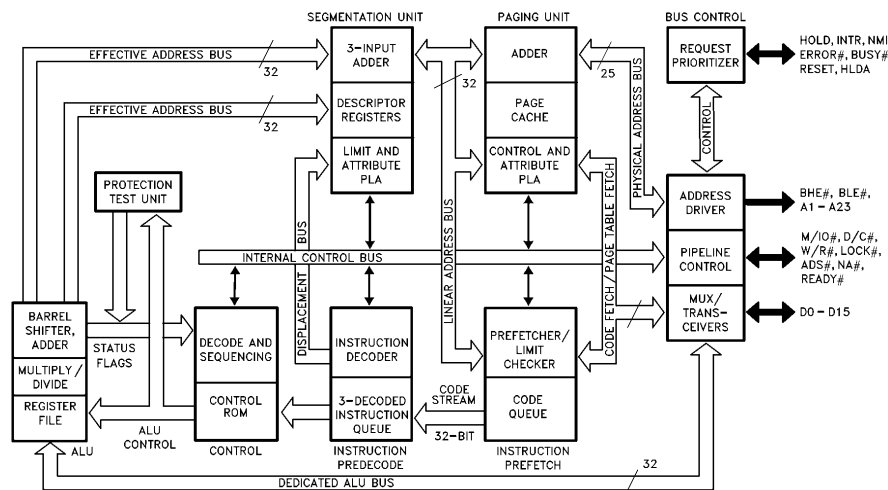




## Intel386™ SX MICROPROCESSOR

- **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 Intel386™ DX Processor
  - Object Code Compatible with 8086, 80186, 80286, and Intel386™ Processors
- **High Performance 16-Bit Data Bus**
  - 16, 20, 25 and 33 MHz Clock
  - Two-Clock Bus Cycles
  - 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 80286 and Intel386 DX 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
- **Numerics Support with the Intel387™ SX Math CoProcessor**
- **On-Chip Debugging Support Including Breakpoint Registers**
- **Complete System Development Support**
  - Software: C, PL/M, Assembler
  - Debuggers: PMON-386 DX, ICETM-386 SX
- **High Speed CHMOS IV Technology**
- **Operating Frequency:**
  - Standard (Intel386 SX -33, -25, -20, -16) Min/Max Frequency (4/33, 4/25, 4/20, 4/16) MHz
  - Low Power (Intel386 SX -33, -25, -20, -16, -12) Min/Max Frequency (2/33, 2/25, 2/20, 2/16, 2/12) MHz
- **100-Pin Plastic Quad Flatpack Package**  
(See Packaging Outlines and Dimensions # 231369)

The Intel386™ SX Microprocessor is an entry-level 32-bit CPU with a 16-bit external data bus and a 24-bit external address bus. The Intel386 SX CPU brings the vast software library of the Intel386™ Architecture to entry-level systems. It provides the performance benefits of a 32-bit programming architecture with the cost savings associated with 16-bit hardware systems.



240187-47

Intel386™ SX Pipelined 32-Bit Microarchitecture

\*Other brands and names are the property of their respective owners.  
Information in this document is provided in connection with Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products. Intel retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.  
COPYRIGHT © INTEL CORPORATION, 1995  
January 1994  
Order Number: 240187-008



## Intel386™ SX MicroProcessor

CONTENTS	PAGE
<b>1.0 PIN DESCRIPTION</b> .....	3
<b>2.0 BASE ARCHITECTURE</b> .....	6
2.1 Register Set .....	6
2.2 Instruction Set .....	10
2.3 Memory Organization .....	11
2.4 Addressing Modes .....	12
2.5 Data Types .....	15
2.6 I/O Space .....	15
2.7 Interrupts and Exceptions .....	17
2.8 Reset and Initialization .....	20
2.9 Testability .....	20
2.10 Debugging Support .....	21
<b>3.0 REAL MODE ARCHITECTURE</b> .....	22
3.1 Memory Addressing .....	22
3.2 Reserved Locations .....	23
3.3 Interrupts .....	23
3.4 Shutdown and Halt .....	23
3.5 LOCK Operations .....	23
<b>4.0 PROTECTED MODE ARCHITECTURE</b> .....	24
4.1 Addressing Mechanism .....	24
4.2 Segmentation .....	24
4.3 Protection .....	29
4.4 Paging .....	33
4.5 Virtual 8086 Environment .....	36

CONTENTS	PAGE
<b>5.0 FUNCTIONAL DATA</b> .....	39
5.1 Signal Description Overview .....	39
5.2 Bus Transfer Mechanism .....	45
5.3 Memory and I/O Spaces .....	45
5.4 Bus Functional Description .....	45
5.5 Self-test Signature .....	63
5.6 Component and Revision Identifiers .....	63
5.7 Coprocessor Interfacing .....	63
<b>6.0 PACKAGE THERMAL SPECIFICATIONS</b> .....	64
<b>7.0 ELECTRICAL SPECIFICATIONS</b> .....	64
7.1 Power and Grounding .....	64
7.2 Maximum Ratings .....	65
7.3 D.C. Specifications .....	66
7.4 A.C. Specifications .....	68
7.5 Designing for ICE™-Intel386 SX Emulator .....	78
<b>8.0 DIFFERENCES BETWEEN THE Intel386™ SX CPU and the Intel386™ DX CPU</b> .....	79
<b>9.0 INSTRUCTION SET</b> .....	80
9.1 Intel386™ SX CPU Instruction Encoding and Clock Count Summary .....	80
9.2 Instruction Encoding .....	95

## 1.0 PIN DESCRIPTION

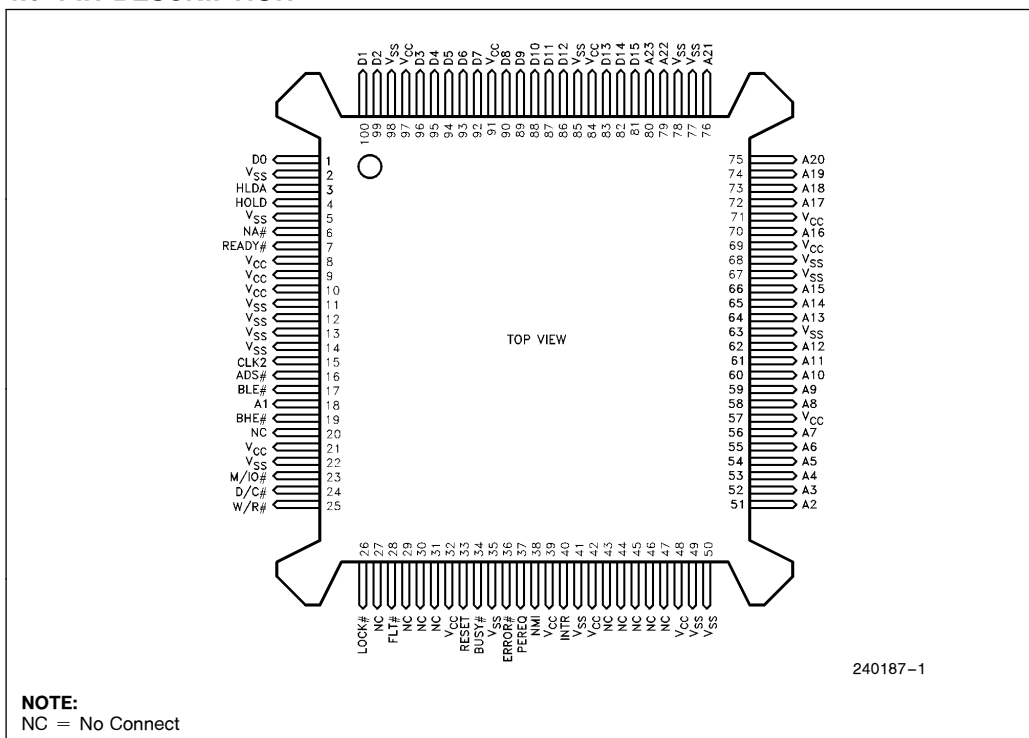


Figure 1.1. Intel386™ SX Microprocessor Pin out Top View

Table 1.1. Alphabetical Pin Assignments

Address		Data		Control		N/C	V <sub>CC</sub>	V <sub>SS</sub>
A <sub>1</sub>	18	D <sub>0</sub>	1	ADS#	16	20	8	2
A <sub>2</sub>	51	D <sub>1</sub>	100	BHE#	19	27	9	5
A <sub>3</sub>	52	D <sub>2</sub>	99	BLE#	17	29	10	11
A <sub>4</sub>	53	D <sub>3</sub>	96	BUSY#	34	30	21	12
A <sub>5</sub>	54	D <sub>4</sub>	95	CLK2	15	31	32	13
A <sub>6</sub>	55	D <sub>5</sub>	94	D/C#	24	43	39	14
A <sub>7</sub>	56	D <sub>6</sub>	93	ERROR#	36	44	42	22
A <sub>8</sub>	58	D <sub>7</sub>	92	FLT#	28	45	48	35
A <sub>9</sub>	59	D <sub>8</sub>	90	HLDA	3	46	57	41
A <sub>10</sub>	60	D <sub>9</sub>	89	HOLD	4	47	69	49
A <sub>11</sub>	61	D <sub>10</sub>	88	INTR	40		71	50
A <sub>12</sub>	62	D <sub>11</sub>	87	LOCK#	26		84	63
A <sub>13</sub>	64	D <sub>12</sub>	86	M/IO#	23		91	67
A <sub>14</sub>	65	D <sub>13</sub>	83	NA#	6		97	68
A <sub>15</sub>	66	D <sub>14</sub>	82	NMI	38			77
A <sub>16</sub>	70	D <sub>15</sub>	81	PEREQ	37			78
A <sub>17</sub>	72			READY#	7			85
A <sub>18</sub>	73			RESET	33			98
A <sub>19</sub>	74			W/R#	25			
A <sub>20</sub>	75							
A <sub>21</sub>	76							
A <sub>22</sub>	79							
A <sub>23</sub>	80							

**1.0 PIN DESCRIPTION** (Continued)

The following are the Intel386™ SX Microprocessor 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	<b>CLK2</b> provides the fundamental timing for the Intel386 SX Microprocessor. For additional information see <b>Clock</b> .
RESET	I	33	<b>RESET</b> suspends any operation in progress and places the Intel386 SX Microprocessor in a known reset state. See <b>Interrupt Signals</b> for additional information.
D <sub>15</sub> –D <sub>0</sub>	I/O	81-83,86-90, 92-96,99-100,1	<b>Data Bus</b> inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See <b>Data Bus</b> for additional information.
A <sub>23</sub> –A <sub>1</sub>	O	80-79,76-72,70, 66-64,62-58, 56-51,18	<b>Address Bus</b> outputs physical memory or port I/O addresses. See <b>Address Bus</b> for additional information.
W/R #	O	25	<b>Write/Read</b> is a bus cycle definition pin that distinguishes write cycles from read cycles. See <b>Bus Cycle Definition Signals</b> for additional information.
D/C #	O	24	<b>Data/Control</b> 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 <b>Bus Cycle Definition Signals</b> for additional information.
M/IO #	O	23	<b>Memory/IO</b> is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See <b>Bus Cycle Definition Signals</b> for additional information.
LOCK #	O	26	<b>Bus Lock</b> 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 <b>Bus Cycle Definition Signals</b> for additional information.
ADS #	O	16	<b>Address Status</b> indicates that a valid bus cycle definition and address (W/R #, D/C #, M/IO #, BHE #, BLE # and A <sub>23</sub> –A <sub>1</sub> ) are being driven at the Intel386 SX Microprocessor pins. See <b>Bus Control Signals</b> for additional information.
NA #	I	6	<b>Next Address</b> is used to request address pipelining. See <b>Bus Control Signals</b> for additional information.
READY #	I	7	<b>Bus Ready</b> terminates the bus cycle. See <b>Bus Control Signals</b> for additional information.
BHE #, BLE #	O	19,17	<b>Byte Enables</b> indicate which data bytes of the data bus take part in a bus cycle. See <b>Address Bus</b> for additional information.

## 1.0 PIN DESCRIPTION (Continued)

Symbol	Type	Pin	Name and Function
HOLD	I	4	<b>Bus Hold Request</b> input allows another bus master to request control of the local bus. See <b>Bus Arbitration Signals</b> for additional information.
HLDA	O	3	<b>Bus Hold Acknowledge</b> output indicates that the Intel386 SX Microprocessor has surrendered control of its local bus to another bus master. See <b>Bus Arbitration Signals</b> for additional information.
INTR	I	40	<b>Interrupt Request</b> is a maskable input that signals the Intel386 SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See <b>Interrupt Signals</b> for additional information.
NMI	I	38	<b>Non-Maskable Interrupt Request</b> is a non-maskable input that signals the Intel386 SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See <b>Interrupt Signals</b> for additional information.
BUSY#	I	34	<b>Busy</b> signals a busy condition from a processor extension. See <b>Coprocessor Interface Signals</b> for additional information.
ERROR#	I	36	<b>Error</b> signals an error condition from a processor extension. See <b>Coprocessor Interface Signals</b> for additional information.
PEREQ	I	37	<b>Processor Extension Request</b> indicates that the processor has data to be transferred by the Intel386 SX Microprocessor. See <b>Coprocessor Interface Signals</b> for additional information.
FLT#	I	28	<b>Float</b> is an input which forces all bidirectional and output signals, including HLDA, to the tri-state condition. This allows the electrically isolated Intel386SX PQFP to use ONCE (On-Circuit Emulation) method without removing it from the PCB. See <b>Float</b> for additional information.
N/C	-	20, 27, 29-31, 43-47	<b>No Connects</b> 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 Intel386 SX Microprocessor.
V <sub>CC</sub>	I	8-10,21,32,39 42,48,57,69, 71,84,91,97	<b>System Power</b> provides the + 5V nominal DC supply input.
V <sub>SS</sub>	I	2,5,11-14,22 35,41,49-50, 63,67-68, 77-78,85,98	<b>System Ground</b> provides the 0V connection from which all inputs and outputs are measured.

## INTRODUCTION

The Intel386 SX Microprocessor is 100% object code compatible with the Intel386 DX, 286 and 8086 microprocessors. Systems based on the Intel386 SX CPU can access the world's largest existing micro-computer software base, including the growing 32-bit software base.

Instruction pipelining and a high performance ALU ensure short average instruction execution times and high system throughput.

The integrated memory management unit (MMU) includes an address translation cache, multi-tasking hardware, and a four-level hardware-enforced protection mechanism to support operating systems. The virtual machine capability of the Intel386 SX CPU allows simultaneous execution of applications from multiple operating systems.

The Intel386 SX CPU 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.

The Low Power Intel386 SX CPU brings the benefits of the Intel386 Microprocessor 32-bit architecture to Laptop and Notebook personal computer applications. With its power saving 2 MHz sleep-mode and extended functional temperature range of 0°C to 100°C T<sub>CASE</sub>, the Lower Power Intel386 SX CPU specifically satisfies the power consumption and heat dissipation requirements of today's small form factor computers.

## 2.0 BASE ARCHITECTURE

The Intel386 SX Microprocessor consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and 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 Intel386 SX Microprocessor has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the Intel386 SX Microprocessor 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.

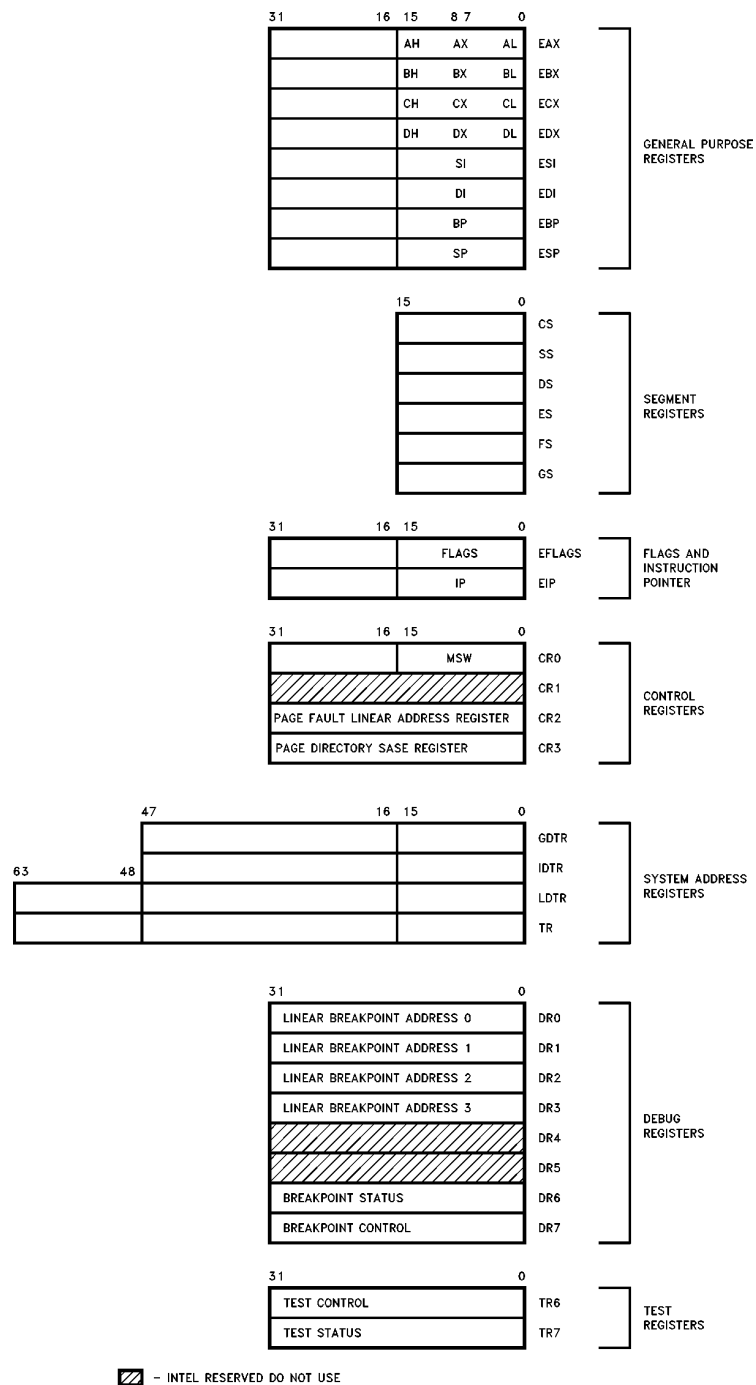
Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each such task behaves with 8086 semantics, thus allowing 8086 software (an application program or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host Intel386 SX Microprocessor operating system by use of paging.

Finally, to facilitate system hardware designs, the Intel386 SX Microprocessor bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

## 2.1 Register Set

The Intel386 SX Microprocessor 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.



240187-2

Figure 2.1. Intel386™ SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor. 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/Intel386 SX/Intel386 DX CPU's 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 Intel386 SX Microprocessor. Their use is discussed in **Testability**.

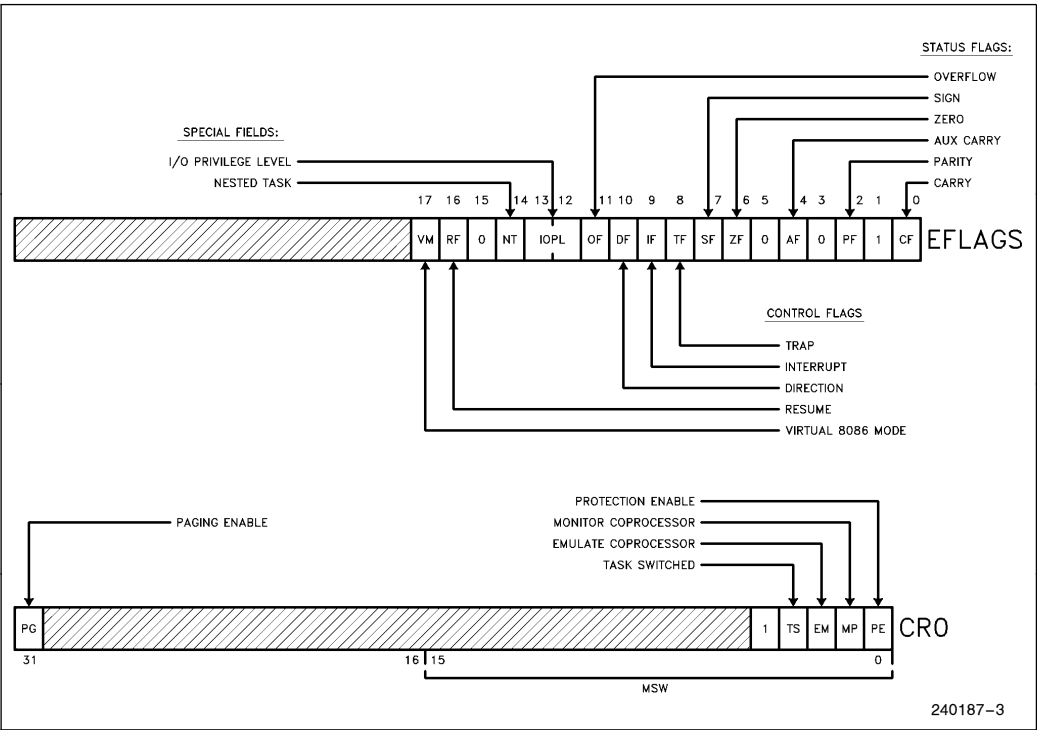


Figure 2.2. Status and Control Register Bit Functions



## 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 Intel386 SX Microprocessor. 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.

## CONTROL REGISTERS

The Intel386 SX Microprocessor 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 2.1. 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.
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 the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12,13	IOPL	I/O Privilege Level—Indicates the maximum Current Privilege Level (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. See Section 4.2 for a further discussion and definitions on various privilege levels.
14	NT	Nested Task—Set if the execution of the current task is nested within another task. Cleared otherwise.
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 Intel386 SX Microprocessor will switch to virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes.



Table 2.2. CR0 Definitions

Bit Position	Name	Function
0	PE	Protection mode enable—places the Intel386 SX Microprocessor 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 instructions are listed in Table 9.1 **Instruction Set Clock Count Summary**.

All Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor (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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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** 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.

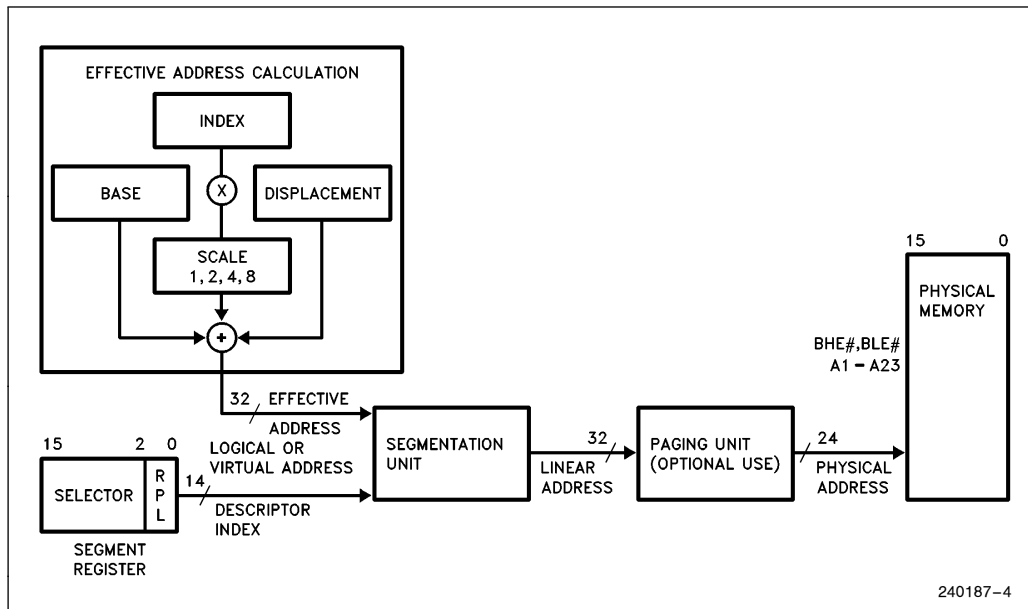


Figure 2.3. Address Translation

### SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the Intel386 SX Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes ( $2^{32}$  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 ad-

dress 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 Intel386 SX Microprocessor 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:

**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.

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, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET 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

### 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:

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

- Direct Mode:** The operand's offset is contained as part of the instruction as an 8, 16 or 32-bit displacement.
- Register Indirect Mode:** A BASE register contains the address of the operand.
- Based Mode:** A BASE register's contents are added to a DISPLACEMENT to form the operand's offset.
- 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.
- 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.
- 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.

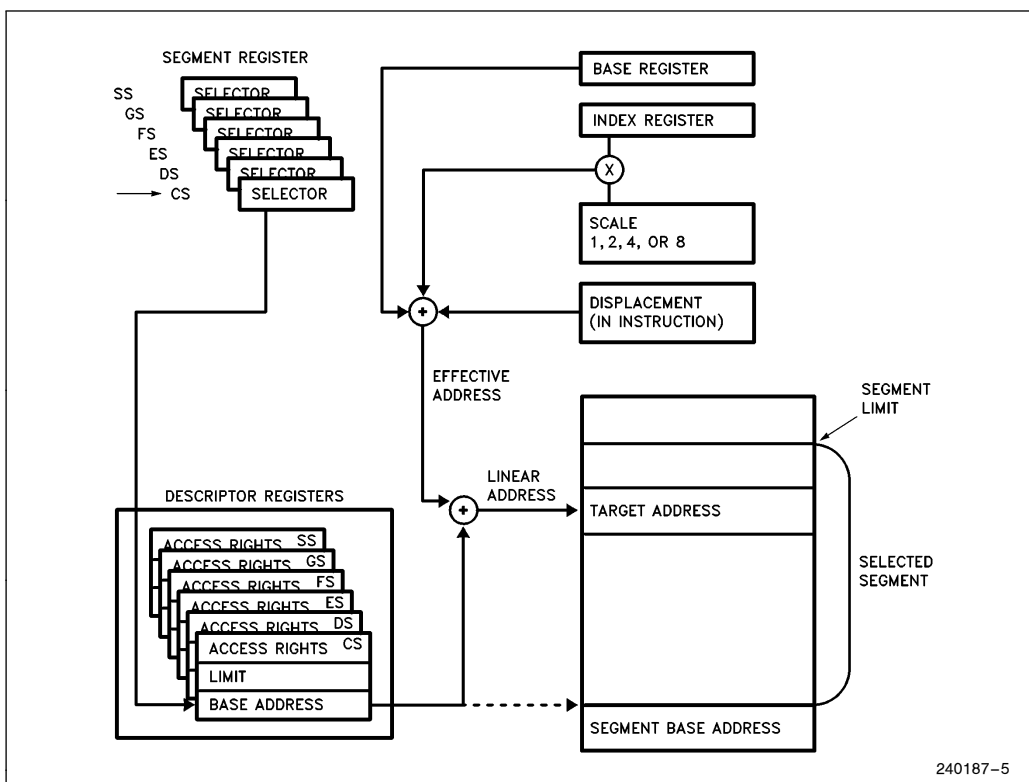


Figure 2.4. Addressing Mode Calculations

#### DIFFERENCES BETWEEN 16 AND 32 BIT ADDRESSES

In order to provide software compatibility with the 8086 and the 80286, the Intel386 SX Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor addressing modes.

When executing 32-bit code, the Intel386 SX Microprocessor uses either 8 or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8 or 16-bits, and the base and index register conform to the 80286 model. Table 2.4 illustrates the differences.

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

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

## 2.5 Data Types

The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor, 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 Intel386 SX Microprocessor is coupled with its numerics coprocessor, the Intel387 SX, 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 Intel387 SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the Intel386 SX Microprocessor and the Intel387 SX.

## 2.6 I/O Space

The Intel386 SX Microprocessor has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the Intel386 SX Microprocessor 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.

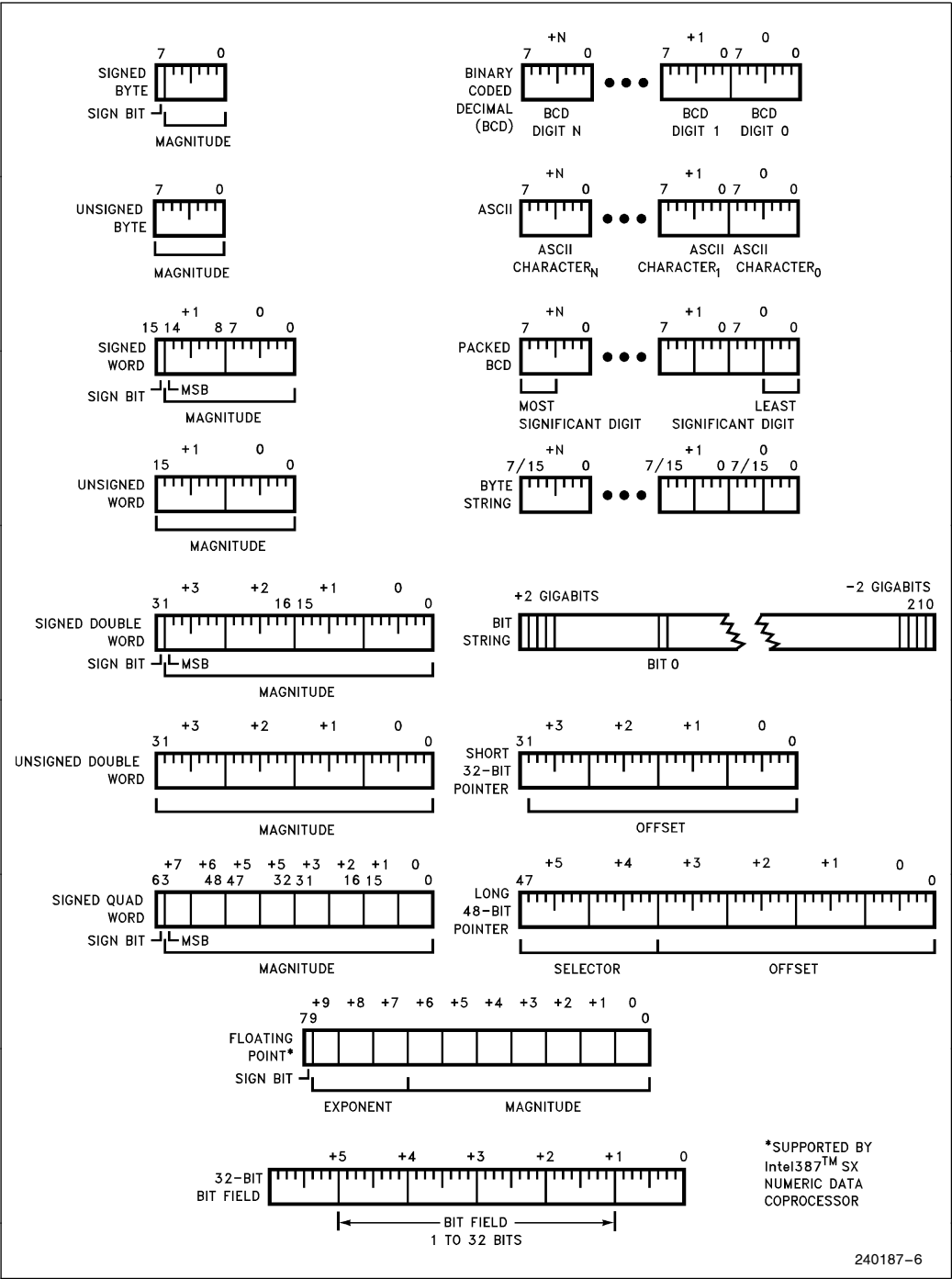


Figure 2.5. Intel386™ SX Microprocessor Supported Data Types



## 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 Intel386 SX Microprocessor and shows where the return address points to.

**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	33–255	INT n	NO	TRAP

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

The Intel386 SX Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode, 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 Intel386 SX Microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the Intel386 SX Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

### Maskable Interrupt

Maskable interrupts are the most common way 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).

Interrupts through interrupt gates automatically reset IF, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGS register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

### Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the  $n^{\text{th}}$  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**.

## 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 Intel386 SX Microprocessor invokes the NMI service routine first. If maskable interrupts are still enabled after the NMI service routine has been invoked, then the Intel386 SX Microprocessor will invoke the appropriate interrupt service routine.

As the Intel386 SX Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is re-

peated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

## INSTRUCTION RESTART

The Intel386 SX Microprocessor fully supports restarting all instructions after Faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.6), the Intel386 SX Microprocessor invokes the appropriate exception service routine. The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for 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)	uuuuuu10H	
Instruction Pointer (EIP)	0000FFFFH	
Code Segment (CS)	F000H	Note 2
Data Segment (DS)	0000H	Note 3
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	Note 3
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX register	0000H	Note 4
EDX register	component and stepping ID	Note 5
All other registers	undefined	Note 6

**NOTES:**

1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. The Code Segment Register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
3. The Data and Extra Segment Registers (DS, ES) will have their Base Address set to 00000000H and Limit set to 0FFFFH.
4. 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.
5. EDX register always holds component and stepping identifier.
6. 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor attempts to invoke the Page Fault service routine). Of course, in any functional system, not only in Intel386 SX Microprocessor-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 Intel386 SX Microprocessor will then start executing instructions near the top of physical memory, at location 0FFFFFF0H. When the first Intersegment Jump or Call is executed, address lines A<sub>20</sub>–A<sub>23</sub> will drop LOW for CS-relative memory cycles, and the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive, the Intel386 SX Microprocessor will start executing instructions at the top of physical memory.

**2.9 Testability**

The Intel386 SX Microprocessor, like the Intel386 Microprocessor, offers testability features which include a self-test and direct access to the page translation cache.

**SELF-TEST**

The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor can be tested during self-test.

Self-Test is initiated on the Intel386 SX Microprocessor when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is LOW. The self-test takes about 2<sup>20</sup> clocks, or approximately 33 milliseconds with a 16 MHz Intel386 SX CPU. 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.

## TLB TESTING

The Intel386 SX Microprocessor 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 Intel386™ SX Microprocessor Programmer’s Reference Manual.

## 2.10 Debugging Support

The Intel386 SX Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint opcode (0CCH).
2. The single-step capability provided by the TF bit in the flag register.
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 Intel386 SX Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The Intel386 SX Microprocessor 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.

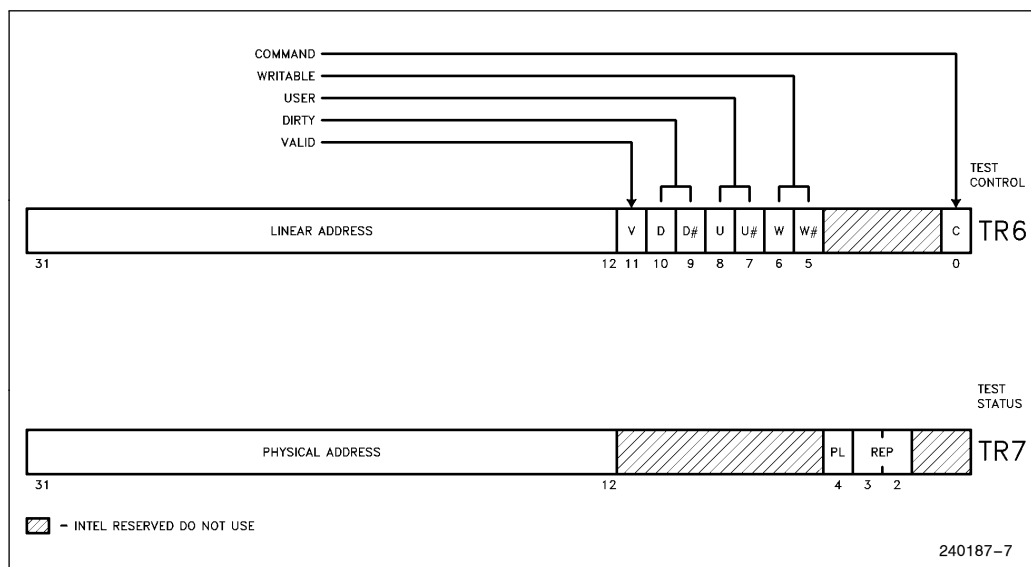


Figure 2.6. Test Registers

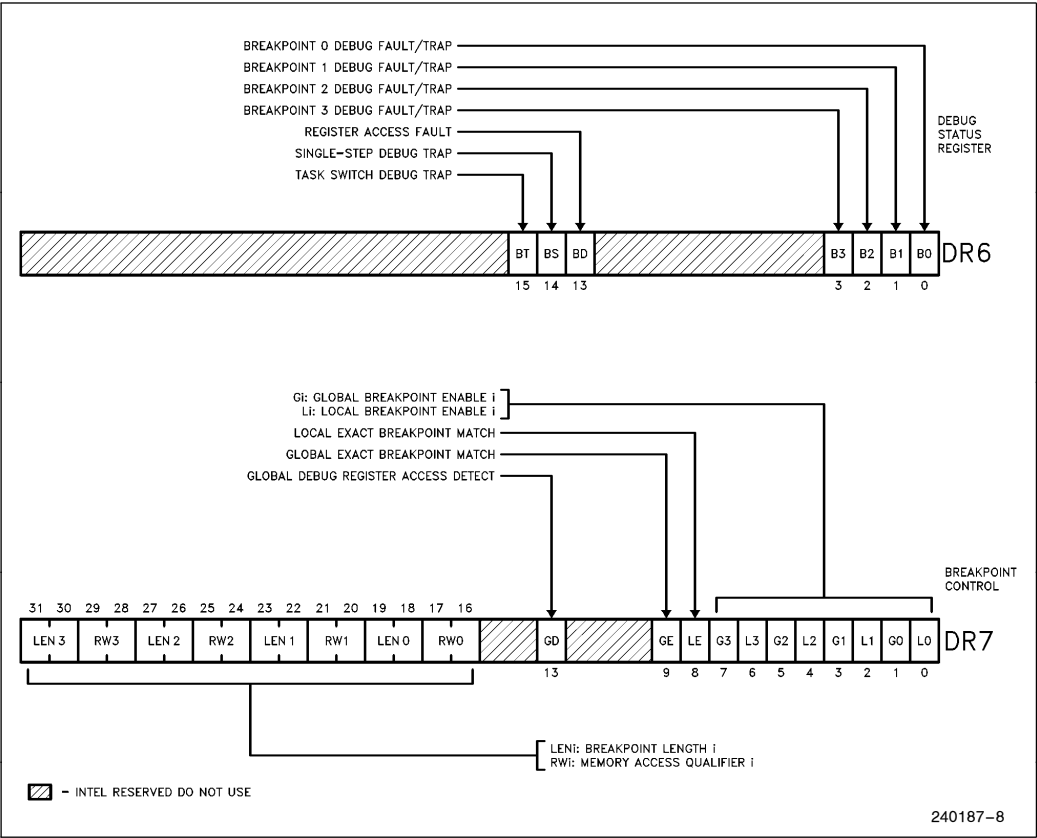


Figure 2.7. Debug Registers

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 Intel386 SX Microprocessor. 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 Intel386 SX Microprocessor in Real Mode is 64K bytes so 32-bit addresses must have a value less than 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 Intel386 SX Microprocessor 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 0FFFF0H 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, FLT#, INTR with interrupts enabled (IF = 1), or RESET will force the Intel386 SX Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

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

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 than 0005H). Otherwise, shutdown can only be exited by a processor reset.

### 3.5 LOCK Operation

The LOCK prefix on the Intel386 SX Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor. 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 Intel386 SX Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes ( $2^{32}$  bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes ( $2^{46}$  bytes)). In addition, Protected Mode allows the Intel386 SX Microprocessor to run all of the existing Intel386 DX CPU (using only 16 megabytes of physical memory), 80286 and 8086 CPU's 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 Intel386 SX Microprocessor remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's 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 Intel386 SX Microprocessor, as 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor system. There are three types of tables 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.



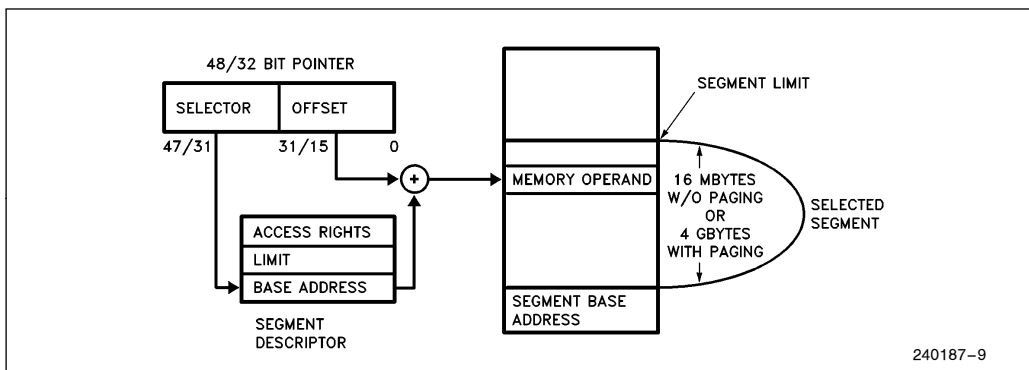


Figure 4.1. Protected Mode Addressing

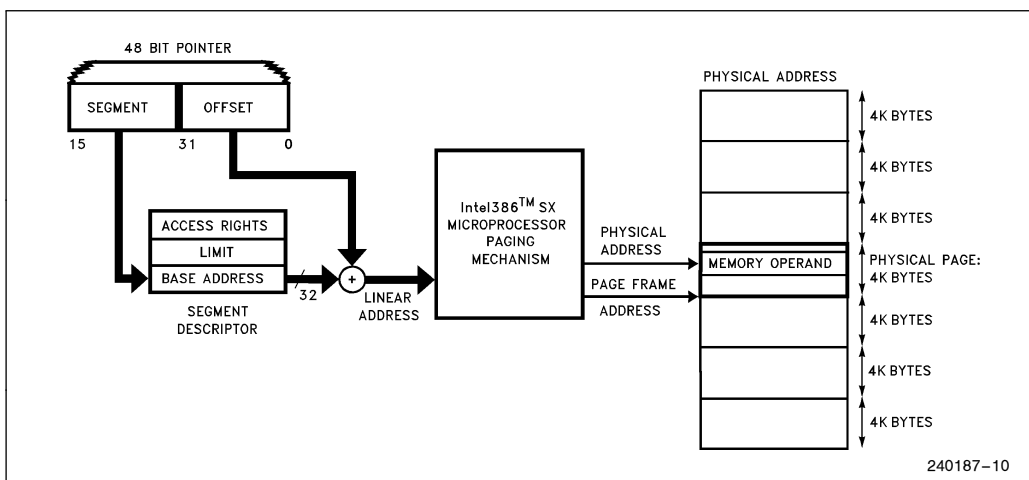


Figure 4.2. Paging and Segmentation

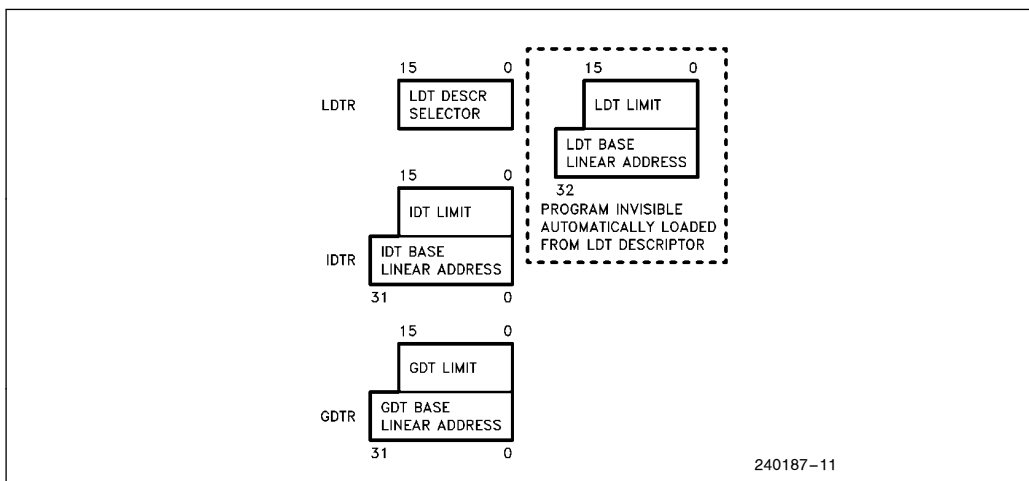


Figure 4.3. Descriptor Table Registers



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 Intel386 SX CPU 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor have three attribute fields in common: the P bit, the DPL bit, and the S bit. The P

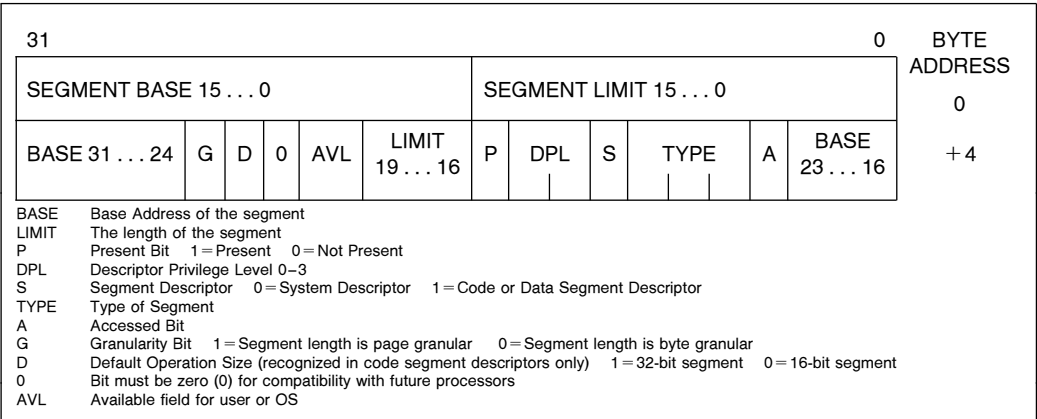


Figure 4.4. Segment Descriptors



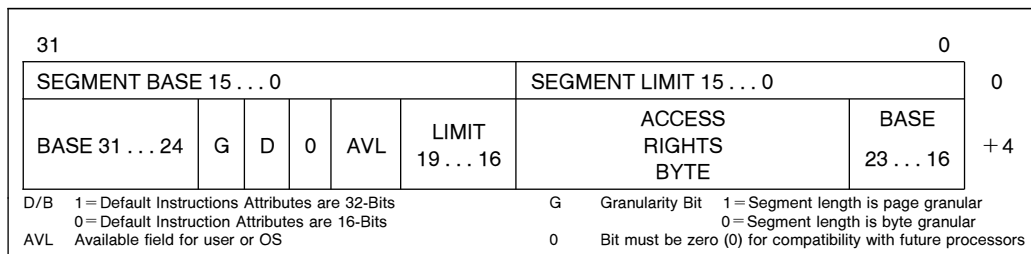
(Present) Bit is 1 if the segment is loaded in physical memory. If P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level, DPL, is a two bit field which specifies the protection level, 0–3, associated with a segment.

The Intel386 SX Microprocessor 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 seg-

ment 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 limit are not used.
6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor
3	Executable (E)	E = 0 Descriptor type is data segment:
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:
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.

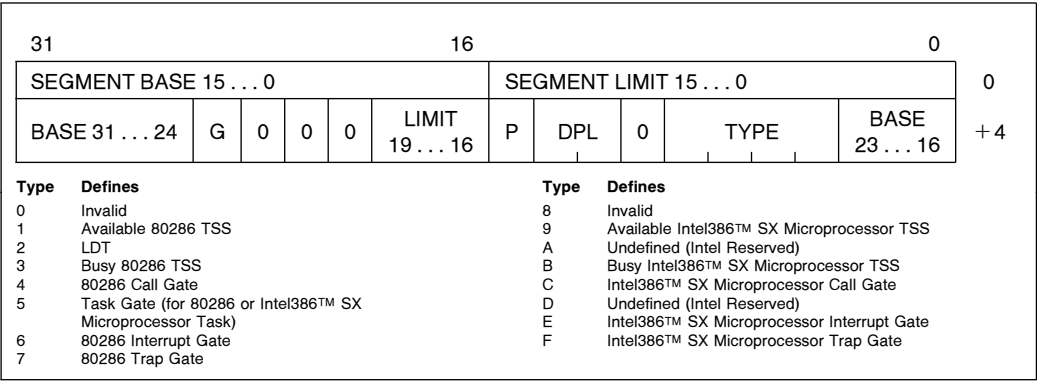


Figure 4.6. System Descriptors

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. Intel386 SX system descriptors (which are the same as Intel386 DX CPU system descriptors) contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

Differences Between Intel386™ SX Microprocessor and 80286 Descriptors

In order to provide operating system compatibility with the 80286 the Intel386 SX CPU supports all of the 80286 segment descriptors. The 80286 system segment descriptors contain a 24-bit base address and 16-bit limit, while the Intel386 SX CPU 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 80286 call gates and 32-bit quantities for Intel386 SX CPU 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.



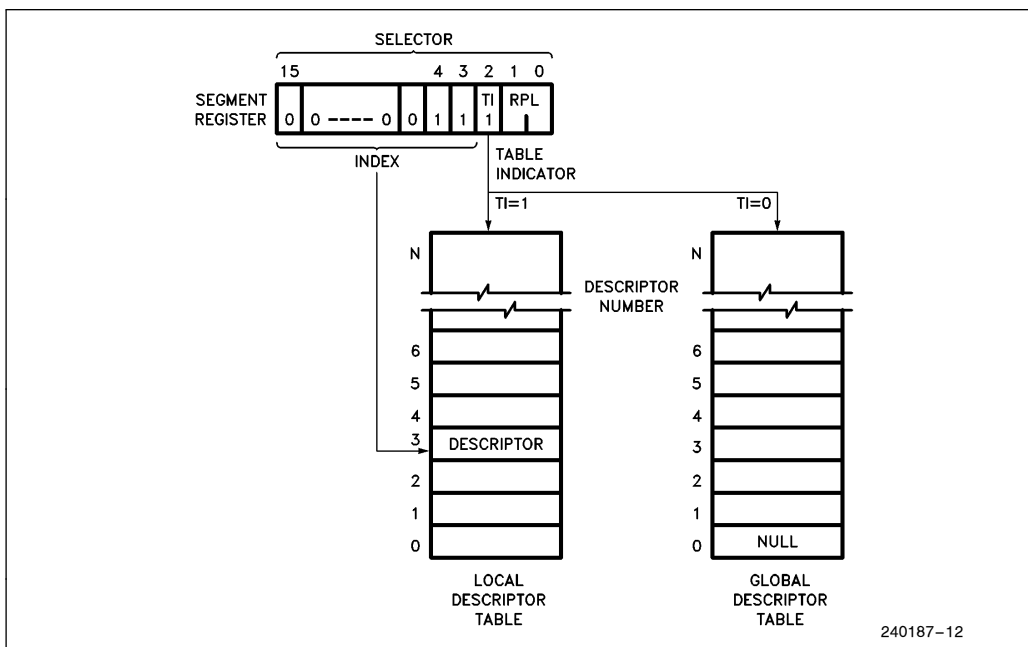


Figure 4.7. Example Descriptor Selection

### 4.3 Protection

The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged level.

#### RULES OF PRIVILEGE

The Intel386 SX Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

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

#### PRIVILEGE LEVELS

At any point in time, a task on the Intel386 SX Microprocessor 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 than 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 Intel386 SX Microprocessor 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.

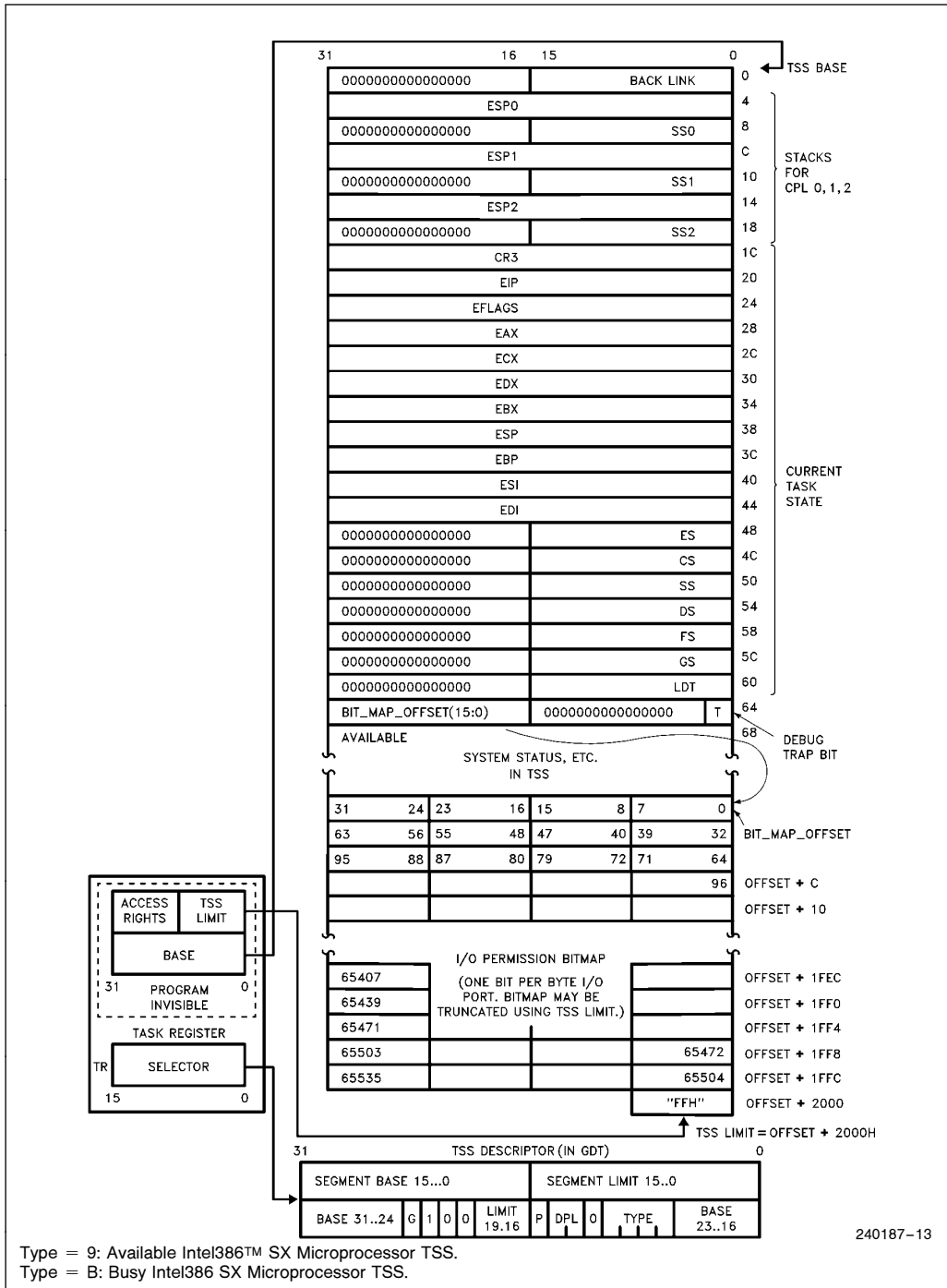


Figure 4.8. Intel386™ SX Microprocessor TSS and TSS Registers

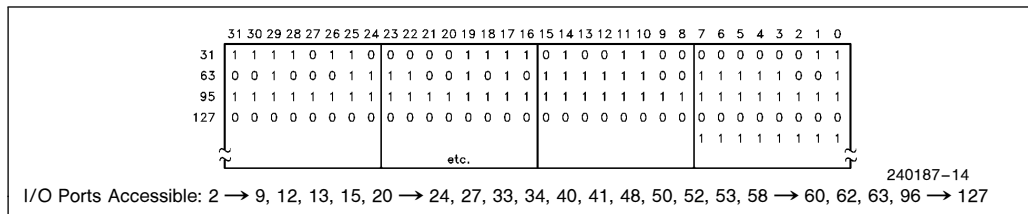


Figure 4.9. Sample I/O Permission Bit Map

**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 Intel386 SX Microprocessor directly supports this operation by providing a task switch instruction in hardware. The 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 execution state. A task gate descriptor contains a TSS selector. The Intel386 SX Microprocessor supports both the 80286 and Intel386 SX CPU TSSs. The limit of a Intel386 SX Microprocessor TSS must be greater than 64H (2BH for an 80286 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 Intel386 SX Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with 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 Intel386 SX Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The 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 Intel386 SX Microprocessor switches task, it sets the TS bit. The Intel386 SX Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the Intel386 SX Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T=1 then upon entry to a new task a debug exception 1 will be generated.



## INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the Intel386 SX Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values. The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and 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 Intel386 SX Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the Intel386 SX Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the Intel386 SX Microprocessor 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.

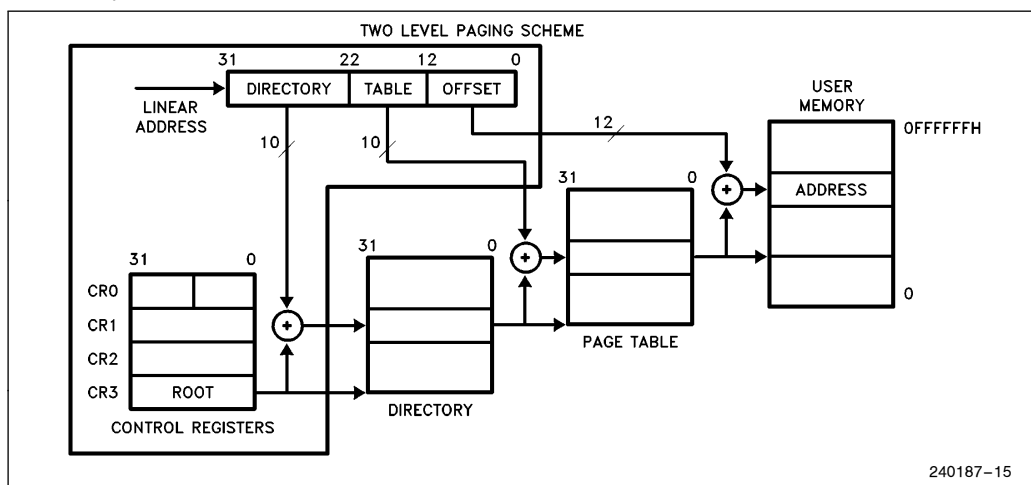


Figure 4.10. Paging Mechanism

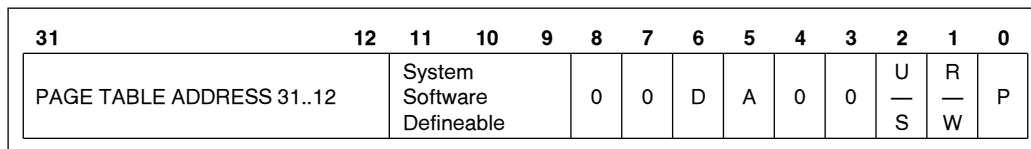


Figure 4.11. Page Directory Entry (Points to Page Table)

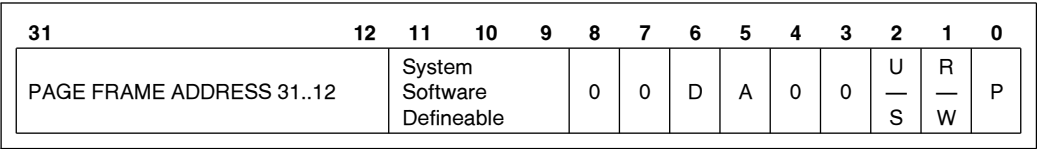


Figure 4.12. Page Table Entry (Points to Page)

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 Intel386 SX CPU'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<sub>31</sub>–A<sub>22</sub>) 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 a Intel386 DX CPU system the upper 20 bits will select one of 2<sup>20</sup> page tables, but for a Intel386 SX Microprocessor system the upper 20 bits only select one of 2<sup>12</sup> page tables. Again, this is because the Intel386 SX Microprocessor is limited to a 24-bit physical address and the upper 8 bits (A<sub>24</sub>–A<sub>31</sub>) 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 ad-

dress. The contents of a Page Table Entry are shown in figure 4.12. The middle 10 bits of the linear address (A<sub>21</sub>–A<sub>12</sub>) 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 Intel386 DX CPU system the upper 20 bits will select one of 2<sup>20</sup> Page Frames, but for an Intel386 SX Microprocessor system the upper 20 bits only select one of 2<sup>12</sup> Page Frames. Again, this is because the Intel386 SX Microprocessor is limited to a 24-bit physical address space and the upper 8 bits (A<sub>24</sub>–A<sub>31</sub>) 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 Intel386 SX CPU 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 Intel386 SX CPU, 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 Intel386 SX Microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User, which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2). Programs executing at Level 0, 1 or 2 bypass the page protection, although segmentation-based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory Entry. The U/S and R/W bits in the 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 Intel386 SX Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the Intel386 SX Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB coupled with a 4K page size results in coverage of 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 Intel386 SX Microprocessor will read the appropriate Page Directory Entry. If P=1 on the Page Directory Entry, indicating that the page table is in memory, then the Intel386 SX Microprocessor will read the appropriate

Page Table Entry and set the Access bit. If P=1 on the Page Table Entry, indicating that the page is in memory, the Intel386 SX Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. 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.

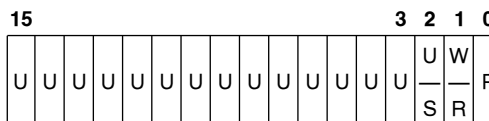


Figure 4.13. Page Fault Error Code Format

**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 Intel386 SX Microprocessor's prefetch queue. This ensures that all instructions executed after the address mode change will generate correct addresses.

The Intel386 SX Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

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

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating 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 Intel386 SX Microprocessor 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 Intel386 SX CPU's protection mechanism.

### VIRTUAL 8086 ADDRESSING MECHANISM

One of the major differences between Intel386 SX CPU 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor. 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 Intel386 SX Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications.

### 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 any time  $CPL \geq 0$ ) 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 Intel386 SX Microprocessor 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.

In protected mode when an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether  $CPL \leq IOPL$ . 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 Intel386 SX CPU 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 Intel386 SX Microprocessor operating system. The Intel386 SX Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

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

The Intel386 SX Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The Intel386 SX Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the Intel386 SX Microprocessor operating system.

An Intel386 SX Microprocessor 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 Intel386 SX Microprocessor task whose Intel386 SX CPU 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 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 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 Intel386 SX CPU 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 Intel386 SX CPU 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 Intel386 SX CPU 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 Intel386 SX CPU TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in Virtual 8086 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 Intel386 SX CPU Trap Gate (Type 14), or Intel386 SX CPU 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. Intel386 SX CPU 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 Intel386 SX CPU 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 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 inter-

rupted. 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 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 interrupt 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 Intel386 SX Microprocessor features a straightforward functional interface to the external hardware. The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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. Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor, 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 Intel386 SX Microprocessor 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<sub>25</sub> Reset Setup Time' and 't<sub>26</sub> Reset Hold Time.' The values of these parameters can be found in Table 7.4.



CLOCK (CLK2)

CLK2 provides the fundamental timing for the Intel386 SX Microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, 'phase one' and 'phase two'. Each CLK2 period is a phase of the internal clock. Figure 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<sub>15</sub>–D<sub>0</sub>)

These three-state bidirectional signals provide the general purpose data path between the Intel386 SX Microprocessor 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.

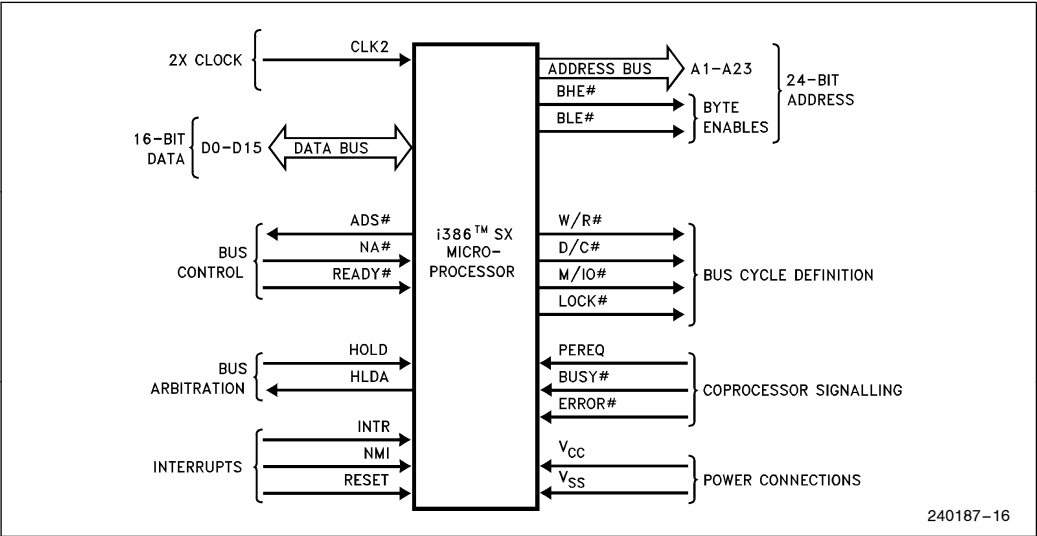


Figure 5.1. Functional Signal Groups

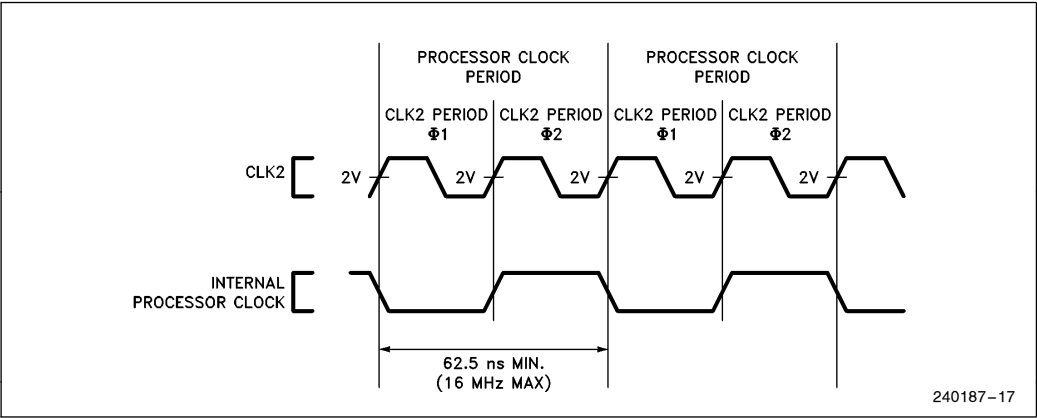


Figure 5.2. CLK2 Signal and Internal Processor Clock



### ADDRESS BUS (A<sub>23</sub>–A<sub>1</sub>, BHE #, BLE #)

These three-state outputs provide physical memory addresses or I/O port addresses. A<sub>23</sub>–A<sub>16</sub> are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A<sub>22</sub>–A<sub>16</sub> are driven LOW, and A<sub>23</sub> 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 Intel386 SX Microprocessor for coprocessor commands is 8000F8H, the I/O addresses driven by the Intel386 SX Microprocessor for coprocessor data are 8000FCH or 8000FEH for cycles to the Intel387™ SX.

The address bus is capable of addressing 16 megabytes of physical memory space (000000H through FFFFFFFH), 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<sub>15</sub>–D<sub>8</sub> and BLE # applies to D<sub>7</sub>–D<sub>0</sub>. 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 by the clock edge 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 <sub>15</sub> –D <sub>8</sub>
1	0	Byte transfer on lower byte of the data bus, D <sub>7</sub> –D <sub>0</sub>
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: Shutdown: Address = 2 Address = 0 BHE # = 1 BHE # = 1 BLE # = 0 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<sub>23</sub>–A<sub>1</sub>) are being driven at the Intel386 SX Microprocessor 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** and **Pipelined Address** 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 Intel386 SX Microprocessor 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<sub>19</sub> and t<sub>20</sub> 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<sub>23</sub>–A<sub>1</sub>, W/R#, D/C# and M/IO# from the Intel386 SX Microprocessor 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<sub>15</sub> and t<sub>16</sub> for correct operation. See **Pipelined Address** and **Read and Write Cycles** 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** for additional information.

#### Bus Hold Request (HOLD)

This input indicates some device other than the Intel386 SX Microprocessor requires bus master-ship. When control is granted, the Intel386 SX Microprocessor floats A<sub>23</sub>–A<sub>1</sub>, BHE#, BLE#, D<sub>15</sub>–D<sub>0</sub>, 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor floated outputs have internal pull-up resistors. See **Resistor Recommendations** 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<sub>23</sub> and t<sub>24</sub> for correct operation.

#### Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor. The other output signals or bidirectional signals (D<sub>15</sub>–D<sub>0</sub>, BHE#, BLE#, A<sub>23</sub>–A<sub>1</sub>, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the re-

questing 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** for additional information.

When the HOLD signal is made inactive, the Intel386 SX Microprocessor 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 <sub>23</sub> –A <sub>1</sub> , BHE#, BLE#, D <sub>15</sub> –D <sub>0</sub>

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 Intel386 SX Microprocessor will not honor a HOLD request until the current bus operation is complete.

The Intel386 SX Microprocessor breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK# signal is not asserted. The Intel386 SX Microprocessor breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again, the LOCK# signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The Intel386 SX Microprocessor 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.

#### 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 Intel386 SX Microprocessor and its Intel387™ SX 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 Intel386 SX Microprocessor. In response, the Intel386 SX Microprocessor transfers information between the coprocessor and memory. Because the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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  $t_{30}$ , rela-

tive 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 Intel386 SX Microprocessor performs an internal self-test (see **Bus Activity During and Following Reset**). If BUSY# is sampled HIGH, no self-test is performed.

#### Coprocessor Error (ERROR#)

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 Intel386 SX Microprocessor when a coprocessor instruction is encountered, and if active, the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX CPU Flag Register IF bit. When the Intel386 SX Microprocessor responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D7–D0 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 Intel386 SX Microprocessor'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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor'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 Intel386 SX Microprocessor encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the Intel386 SX Microprocessor'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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor.

## RESET

This input signal suspends any operation in progress and places the Intel386 SX Microprocessor in a known reset state. The Intel386 SX Microprocessor 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, except FLT#, 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor.

**Table 5.5. Pin State (Bus Idle) During Reset**

Pin Name	Signal Level During Reset
ADS#	1
D <sub>15</sub> –D <sub>0</sub>	Float
BHE#, BLE#	0
A <sub>23</sub> –A <sub>1</sub>	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 Intel386 SX Microprocessor address signals are designed to simplify external system hardware. Higher-order address bits are provided by A<sub>23</sub>–A<sub>1</sub>. 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 <sub>7</sub> –D <sub>0</sub>	(byte 0 — least significant)
BHE#	D <sub>15</sub> –D <sub>8</sub>	(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<sub>23</sub> and M/IO# signals.

## 5.4 Bus Functional Description

The Intel386 SX Microprocessor 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<sub>23</sub>–A<sub>1</sub>. A status signal, ADS#, indicates

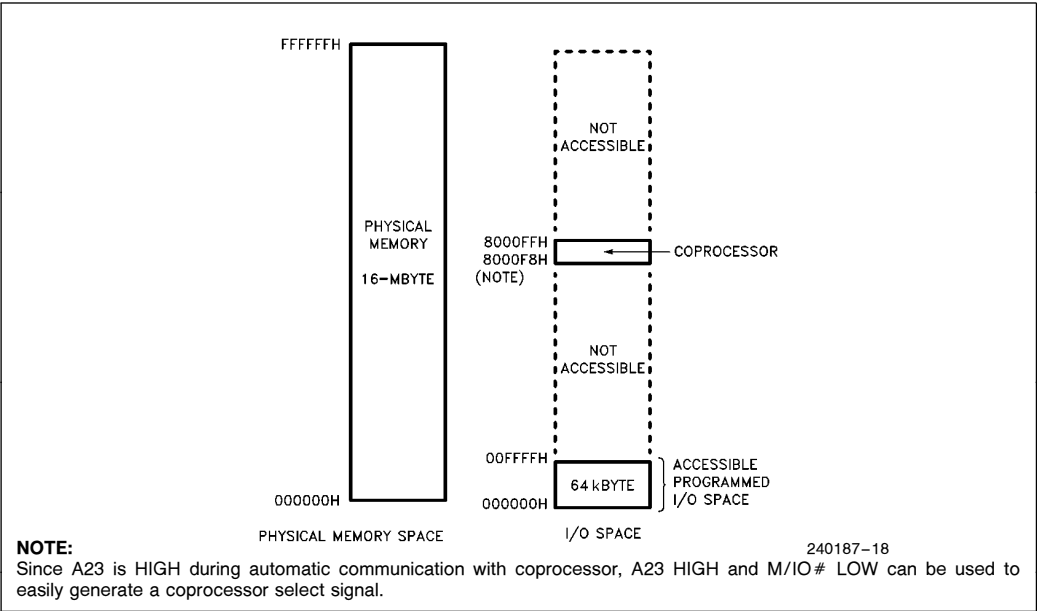


Figure 5.3. Physical Memory and I/O Spaces

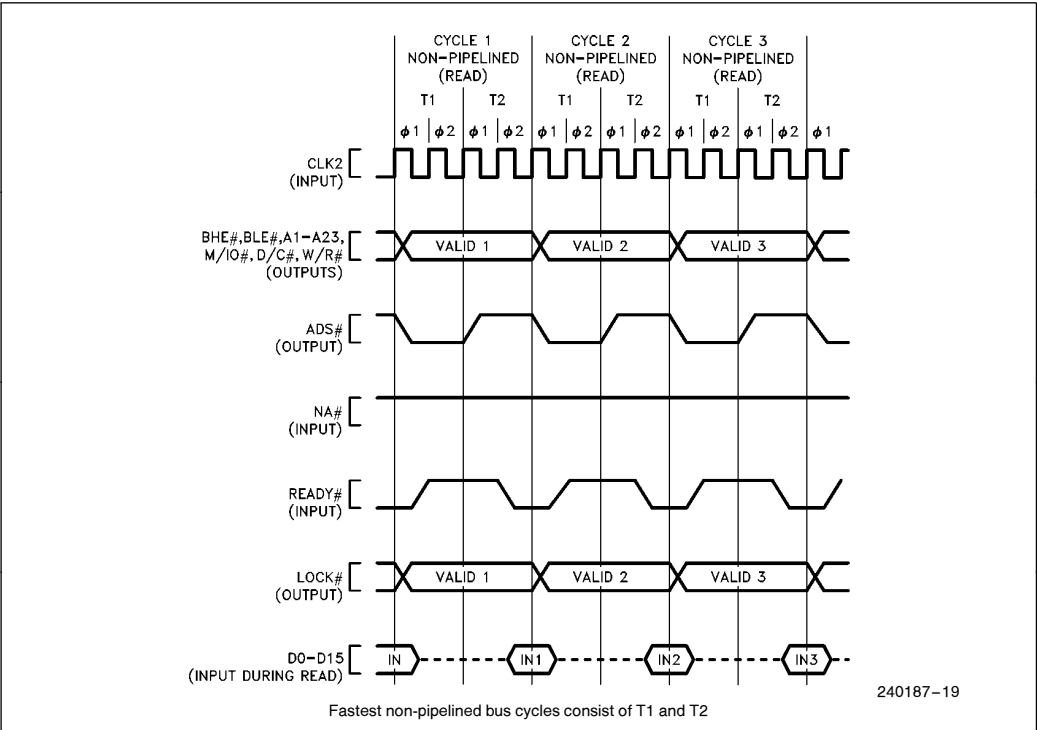


Figure 5.4. Fastest Read Cycles with Non-pipelined Address Timing

when the Intel386 SX Microprocessor issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as 'the bus'. When active, the bus performs one of the bus cycles below:

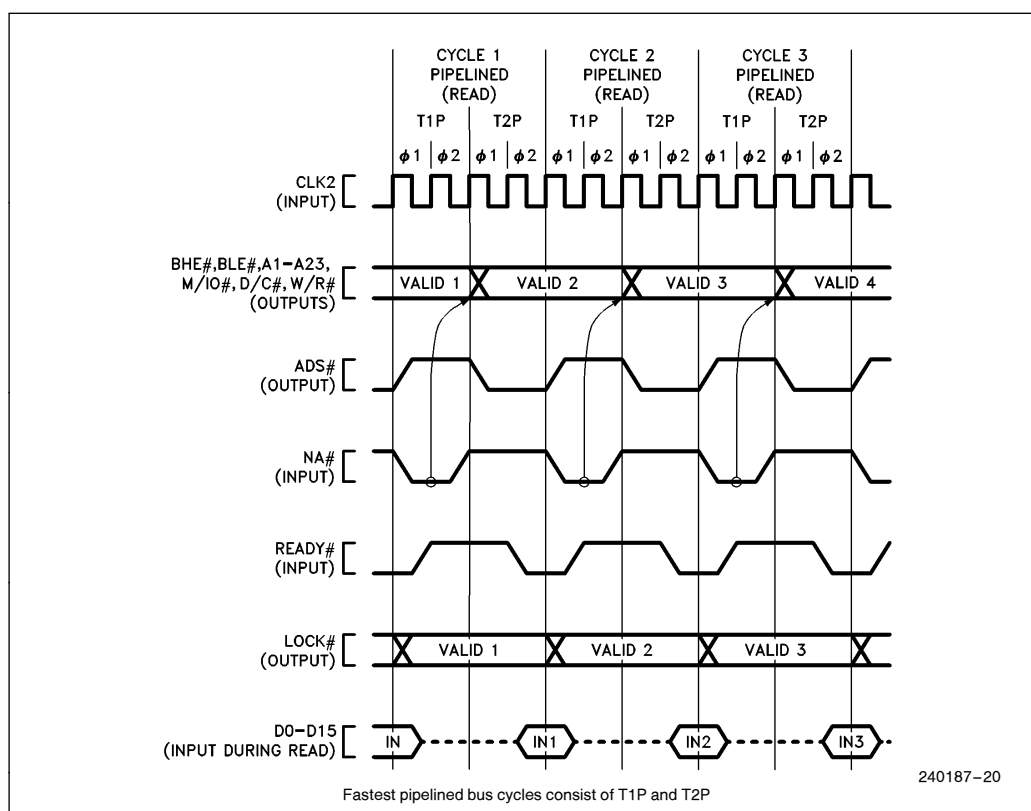
1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space
5. Read from I/O space (or coprocessor)
6. Write to I/O space (or coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

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

When the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 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 Intel386 SX Microprocessor **READY#** input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If **READY#** is not immediately asserted however, T2 states are repeated indefinitely until the **READY#** input is sampled 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<sub>23</sub>–A<sub>1</sub>**) 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 Intel386 SX Microprocessor

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.

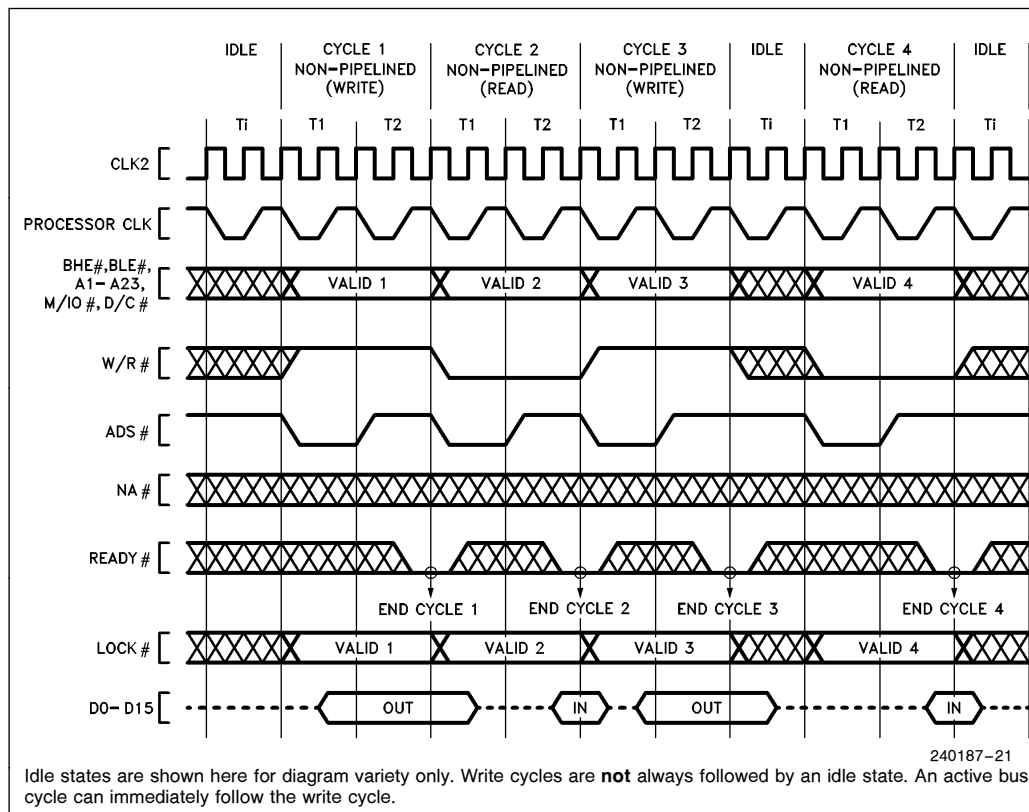


Figure 5.6. Various Bus Cycles with Non-Pipelined Address (zero wait states)

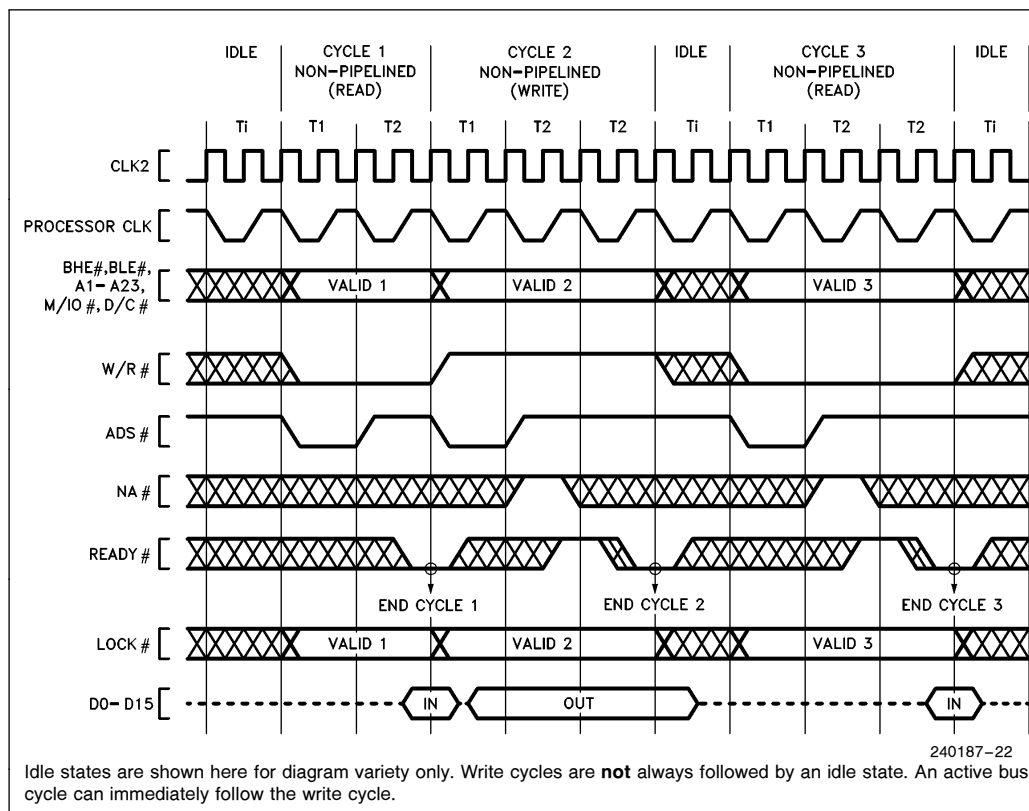


Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined address timing. However the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the Intel386 SX Microprocessor has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#.

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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor terminates it. When a read cycle is acknowledged, the Intel386 SX Microprocessor latches the information present at its data pins. When a write cycle is acknowledged, the Intel386 SX CPU's write data remains valid throughout phase one of the next bus state, to provide write data hold time.



**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 Intel386 SX Microprocessor floats its data signals to allow driving by the external device being addressed. **The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, Ti, or in the hold acknowledge state Th.

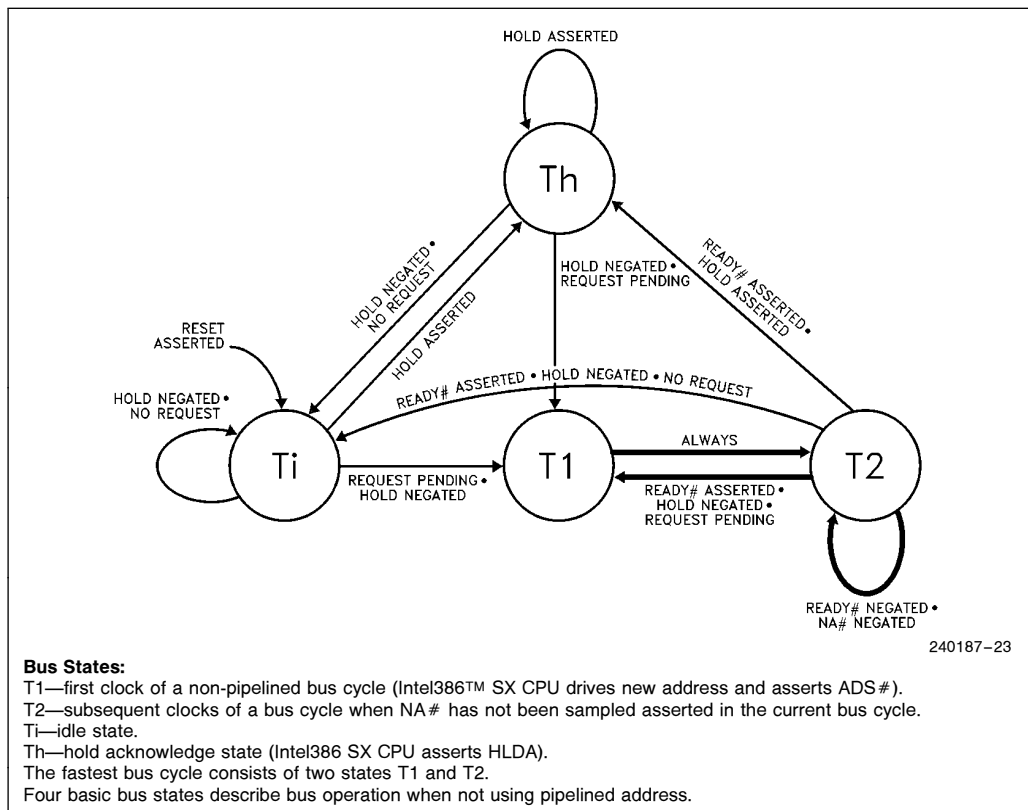


Figure 5.8. Bus States (not using pipelined address)

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<sub>i</sub> if there is no bus request pending, or T<sub>H</sub> if the HOLD input is being asserted.

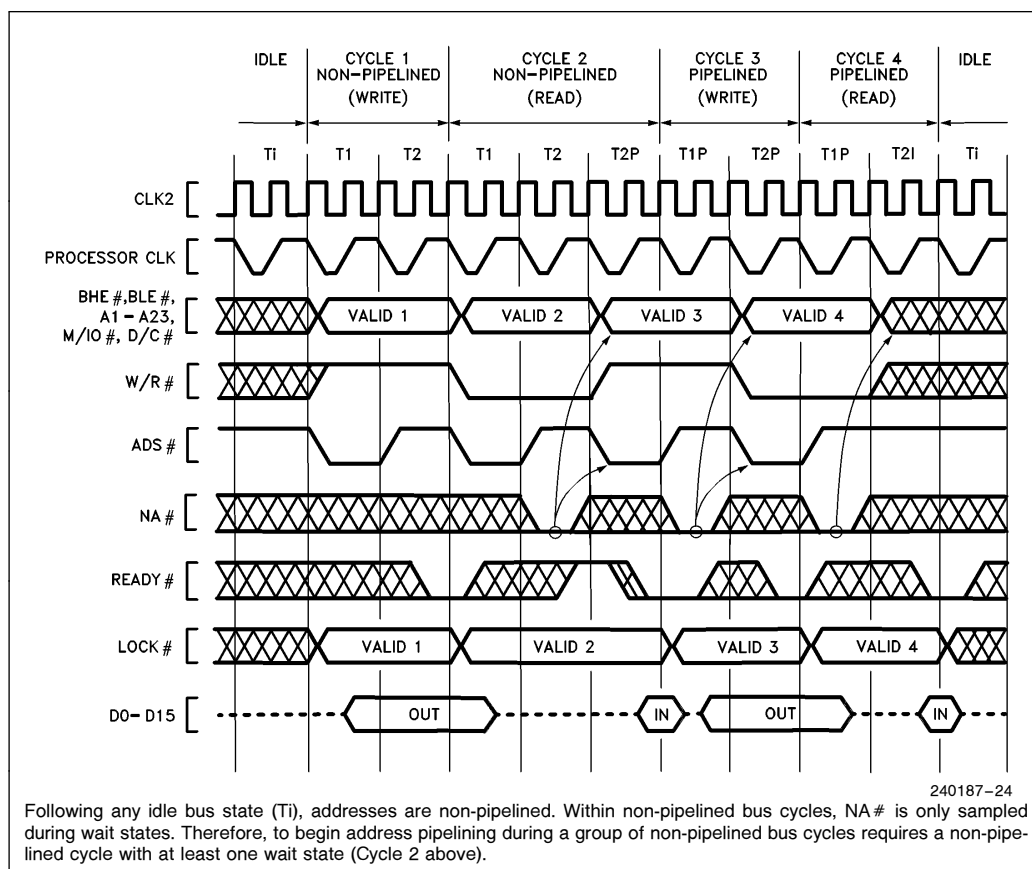
Use of pipelined address allows the Intel386 SX Microprocessor to enter three additional bus states not shown in Figure 5.8. Figure 5.12 is the complete bus state diagram, including pipelined address cycles.

### Pipelined Address

Address pipelining is the option of requesting the address and the bus cycle definition of the next in-

ternally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the Intel386 SX Microprocessor 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).



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

If NA# is sampled active, the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 ADS# output will remain stable on the address pins for at least two processor clock periods. The Intel386 SX Microprocessor 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).

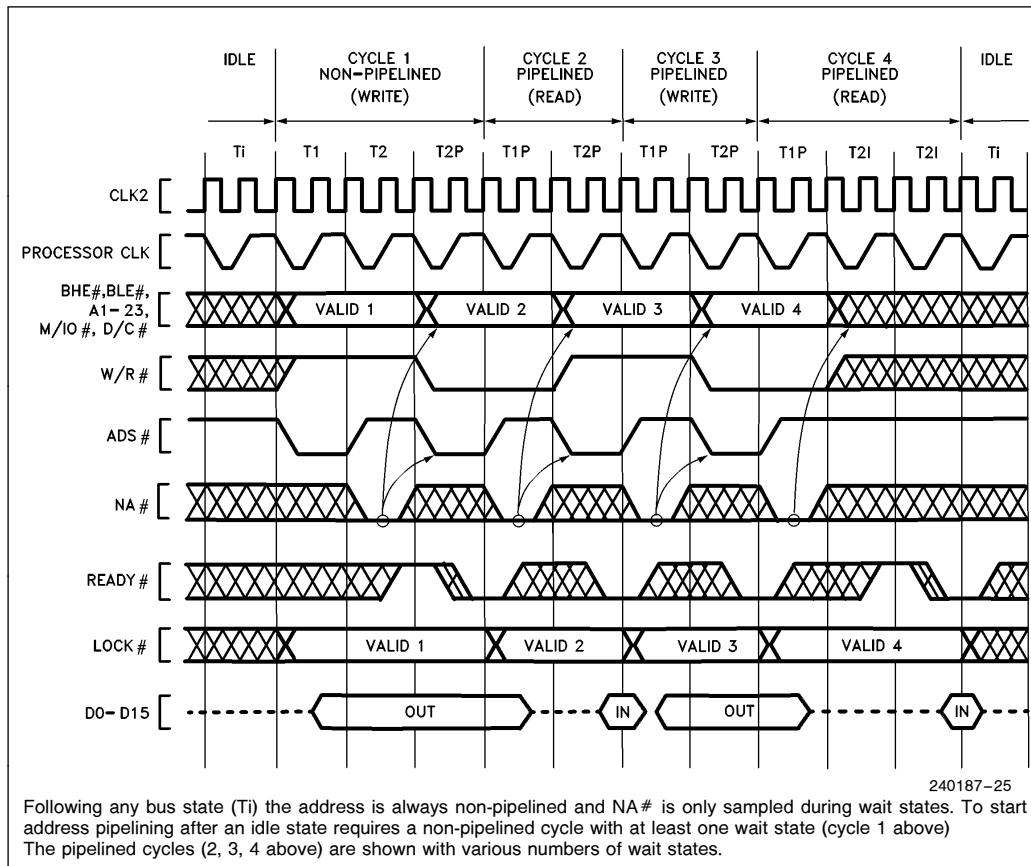


Figure 5.10. Fastest Transition to Pipelined Address Following Idle Bus State

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.

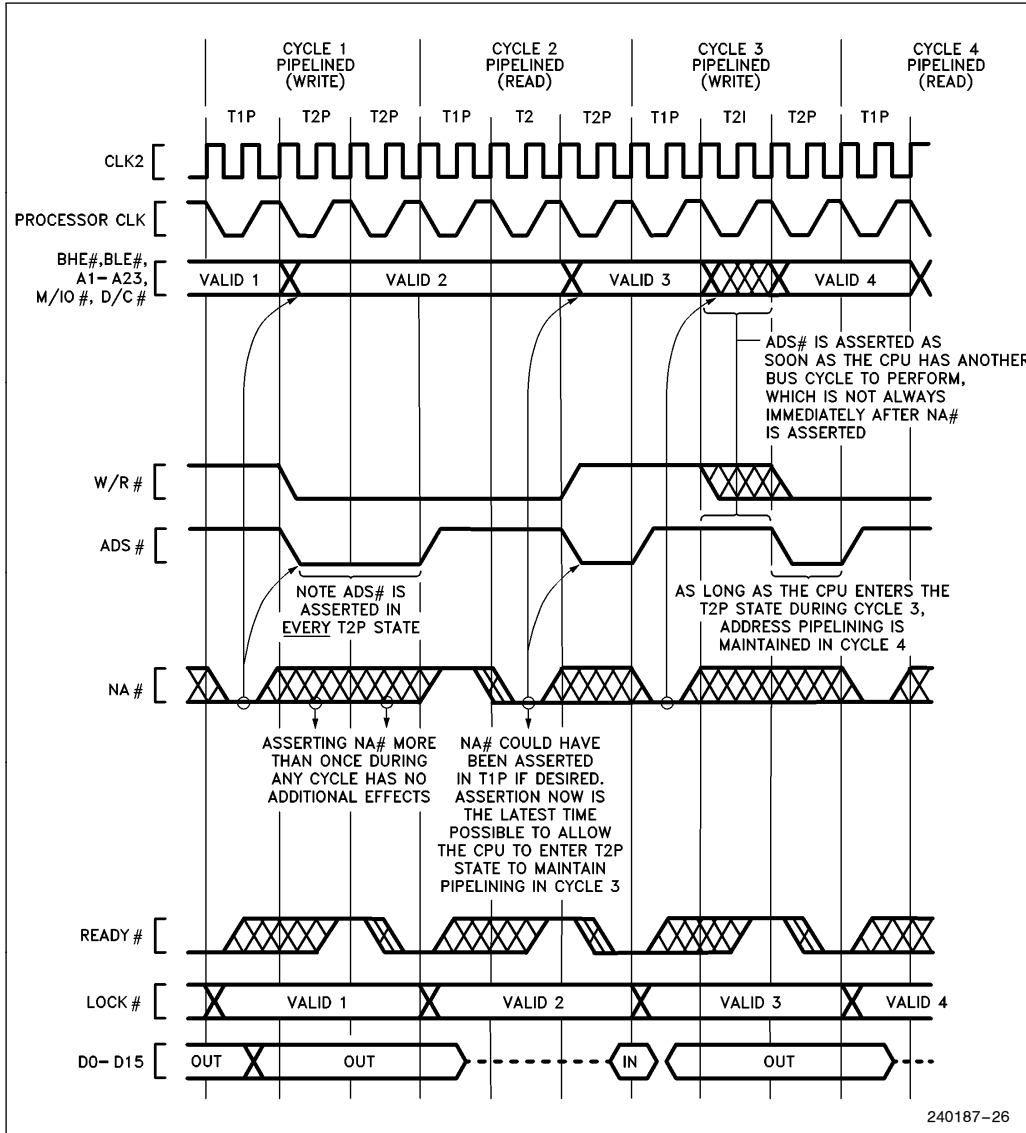


Figure 5.11. Details of Address Pipelining During Cycles with Wait States

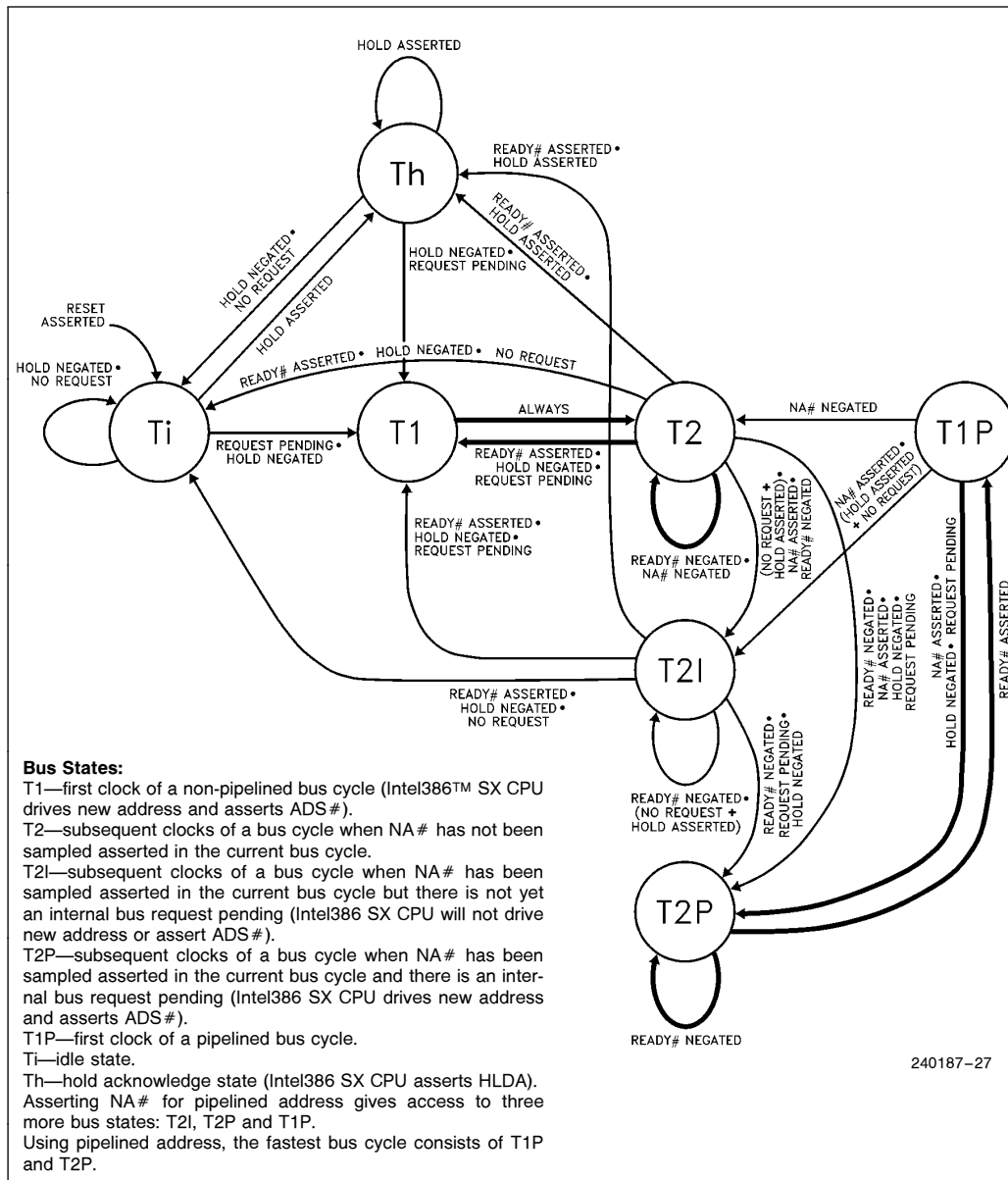


Figure 5.12. Complete Bus States (including pipelined address)

## 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:

<b><math>T_i</math></b>	<b><math>T_i</math></b>	<b><math>T_i</math></b>	<b><math>T1 - T2 - T2P</math></b>	<b><math>T1P - T2P</math></b>
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:

<b><math>T_h</math></b>	<b><math>T_h</math></b>	<b><math>T_h</math></b>	<b><math>T1 - T2 - T2P</math></b>	<b><math>T1P - T2P</math></b>
hold		non-pipelined		pipelined
acknowledge		cycle		cycle
states				

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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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.

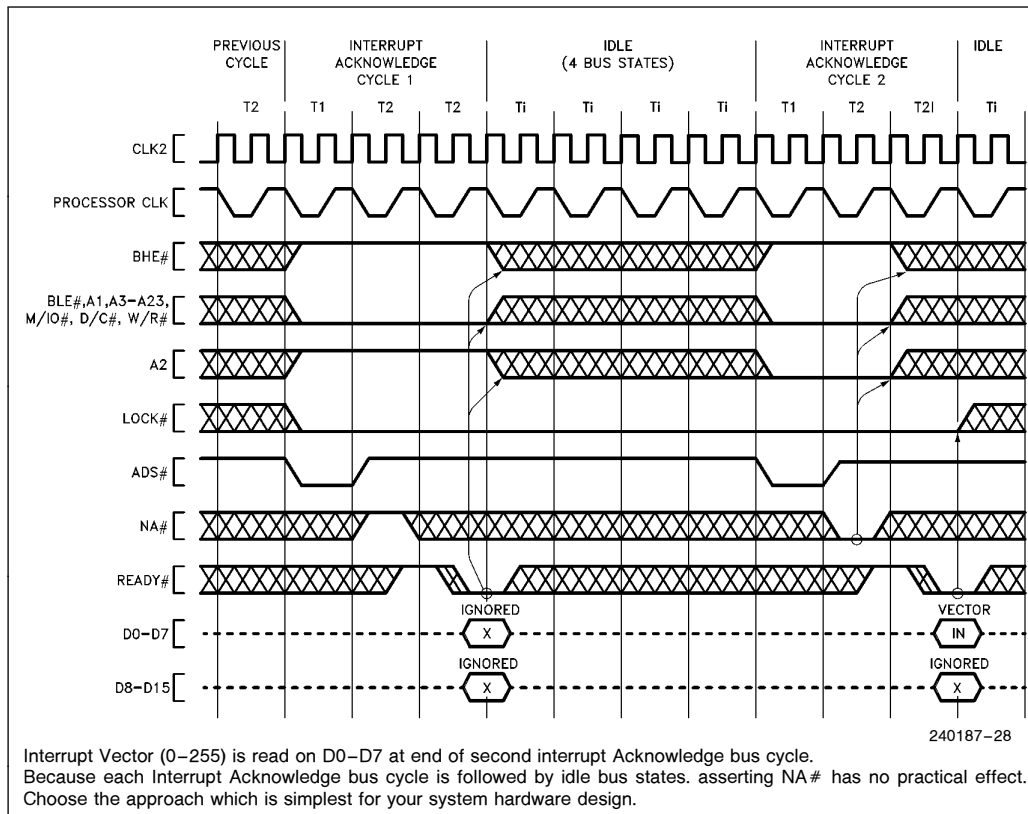
**INTERRUPT ACKNOWLEDGE (INTA) CYCLES**

In response to an interrupt request on the INTR input when interrupts are enabled, the Intel386 SX Microprocessor performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled active.

The state of A<sub>2</sub> distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A<sub>23</sub>–A<sub>3</sub>, A<sub>1</sub>, BLE# LOW, A<sub>2</sub> and BHE# HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A<sub>23</sub>–A<sub>1</sub>, 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<sub>i</sub>, are inserted by the Intel386 SX Microprocessor between the two interrupt acknowledge cycles for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D<sub>15</sub>–D<sub>0</sub> float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the Intel386 SX Microprocessor will read an external interrupt vector from D<sub>7</sub>–D<sub>0</sub> of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.



**Figure 5.13. Interrupt Acknowledge Cycles**



### HALT INDICATION CYCLE

The 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 Intel386 SX Microprocessor resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.

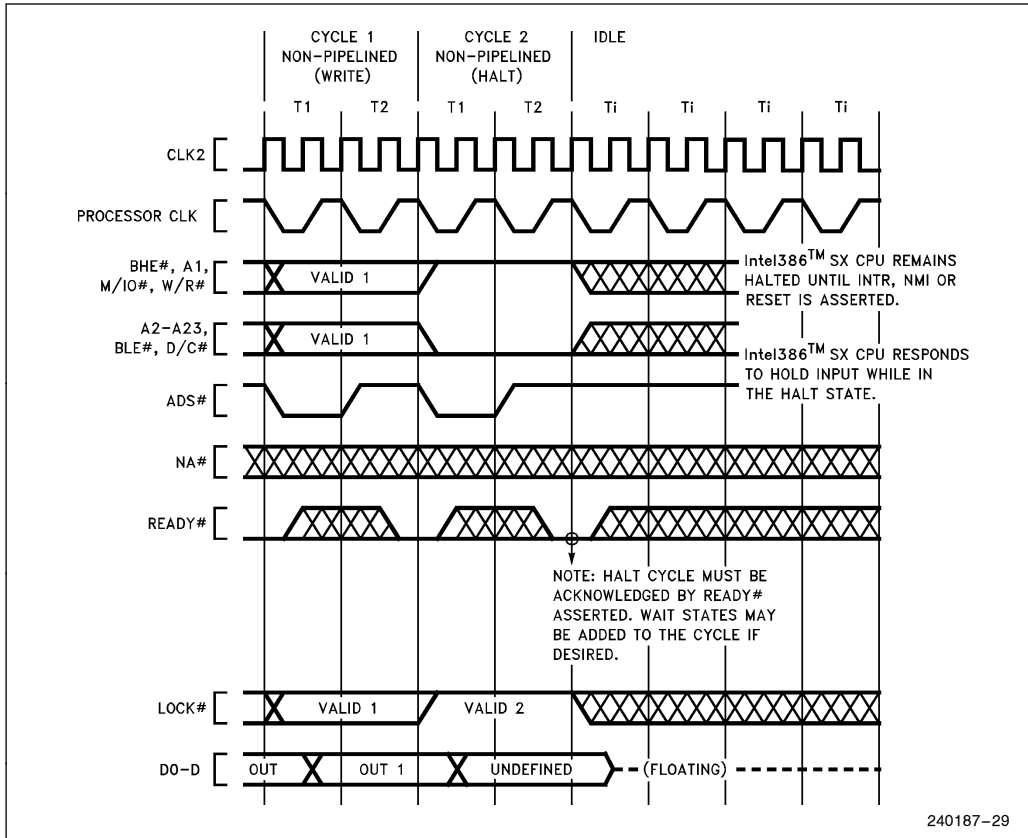


Figure 5.14. Example Halt Indication Cycle from Non-Pipelined Cycle



SHUTDOWN INDICATION CYCLE

The Intel386 SX Microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **Bus Cycle Definition Signals** and an address of 0. The shutdown indication cycle must be acknowledged by **READY#** asserted. A shutdown Intel386 SX Microprocessor 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 Intel386 SX Microprocessor floats all outputs or bidirectional signals, except for **HLDA**. **HLDA** is asserted as long as the Intel386 SX Microprocessor remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except **HOLD**, **FLT#** and **RESET** are ignored.

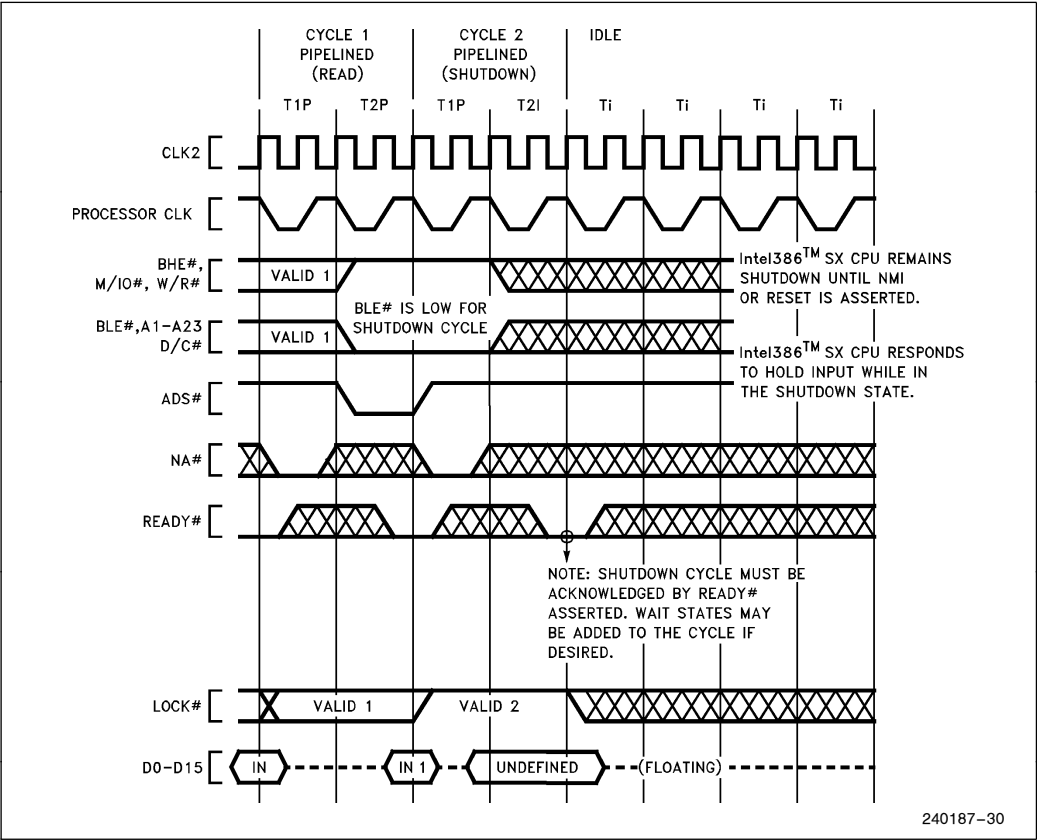


Figure 5.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle



$T_H$  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  $T_1$  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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor 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 Intel386 SX Microprocessor enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the Intel386 SX Microprocessor would otherwise perform its first bus cycle.

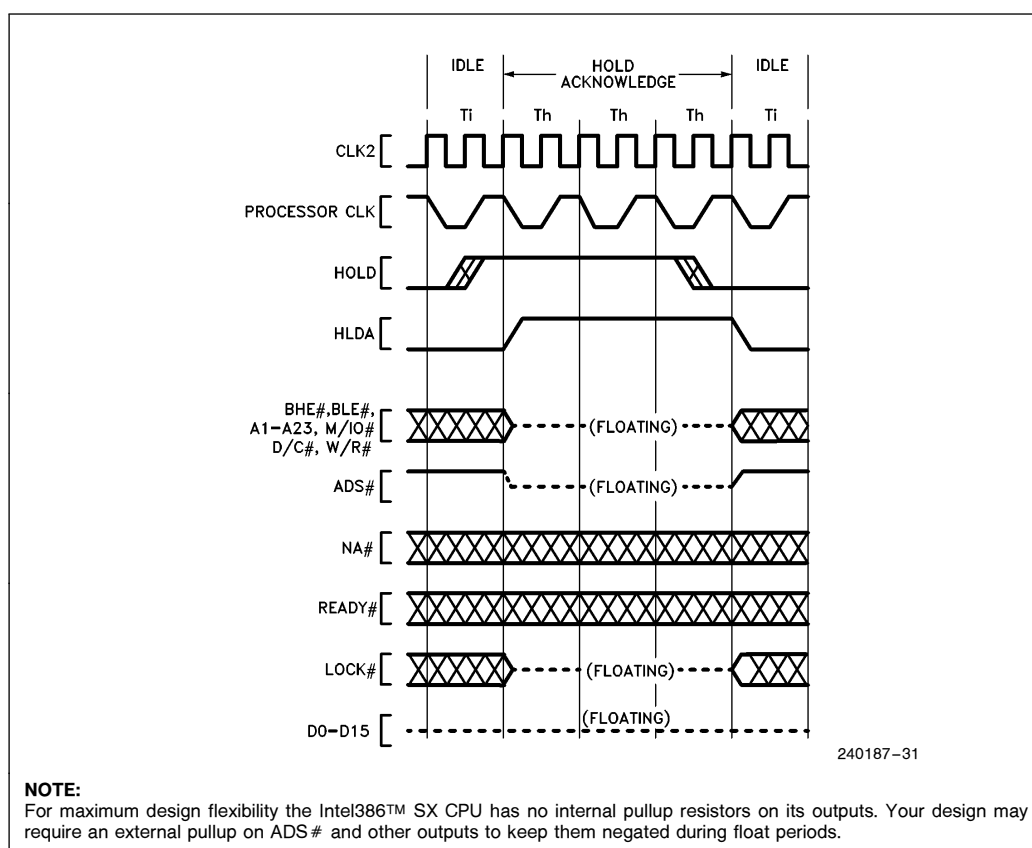


Figure 5.16. Requesting Hold from Idle Bus



FLOAT

Activating the FLT # input floats all Intel386 SX bi-directional and output signals, including HLDA. Asserting FLT # isolates the Intel386 SX from the surrounding circuitry.

As the Intel386 SX is packaged in a surface mount PQFP, it cannot be removed from the motherboard when In-Circuit Emulation (ICE) is needed. The FLT # input allows the Intel386 SX to be electrically isolated from the surrounding circuitry. This allows connection of an emulator to the Intel386 SX PQFP without removing it from the PCB. This method of emulation is referred to as ON-Circuit Emulation (ONCE).

Asserting the FLT # input unconditionally aborts the current bus cycle and forces the Intel386 SX into the FLOAT mode. Since activating FLT # unconditionally forces the Intel386 SX into FLOAT mode, the Intel386 SX is not guaranteed to enter FLOAT in a valid state. After deactivating FLT #, the Intel386 SX is not guaranteed to exit FLOAT mode in a valid state. This is not a problem as the FLT # pin is meant to be used only during ONCE. After exiting FLOAT, the Intel386 SX must be reset to return it to a valid state. Reset should be asserted before FLT # is deasserted. This will ensure that the Intel386 SX will exit float in a valid state.

FLT # has an internal pull-up resistor, and if it is not used it should be unconnected.

ENTERING AND EXITING FLOAT

FLT # is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the Intel386 SX (Figure 5.20). FLT # must be held low for a minimum of 16 CLK2 cycles. Reset should be asserted and held asserted until after FLT # is deasserted. This will ensure that the Intel386 SX will exit float in a valid state.

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.

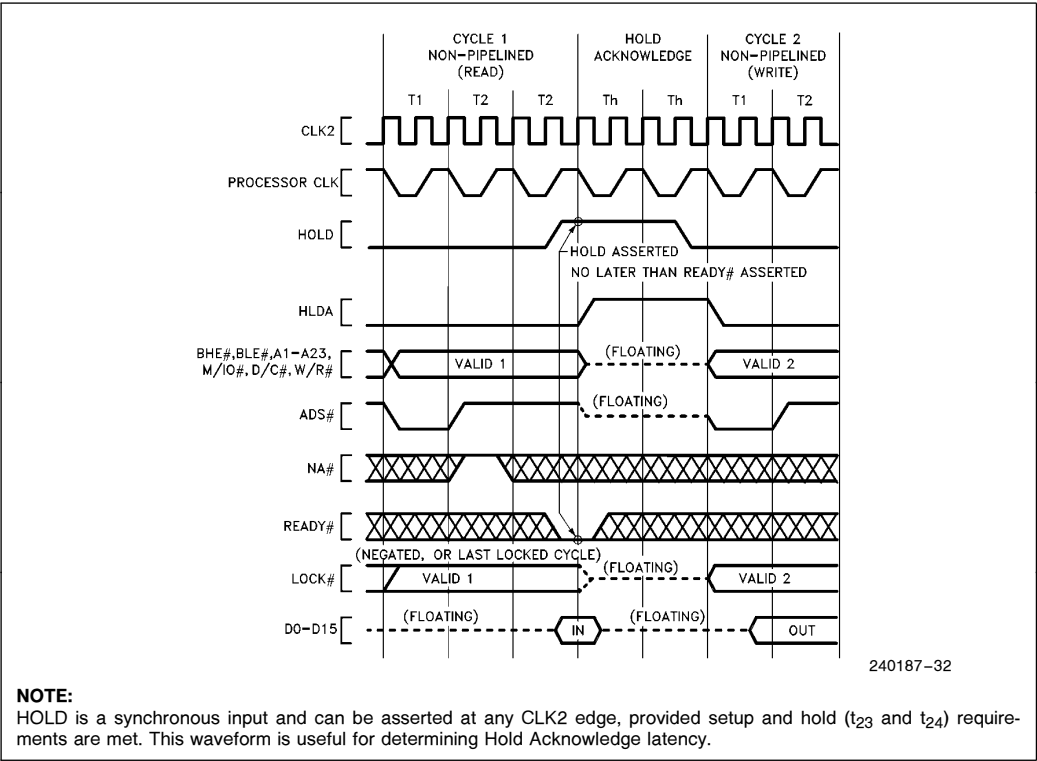


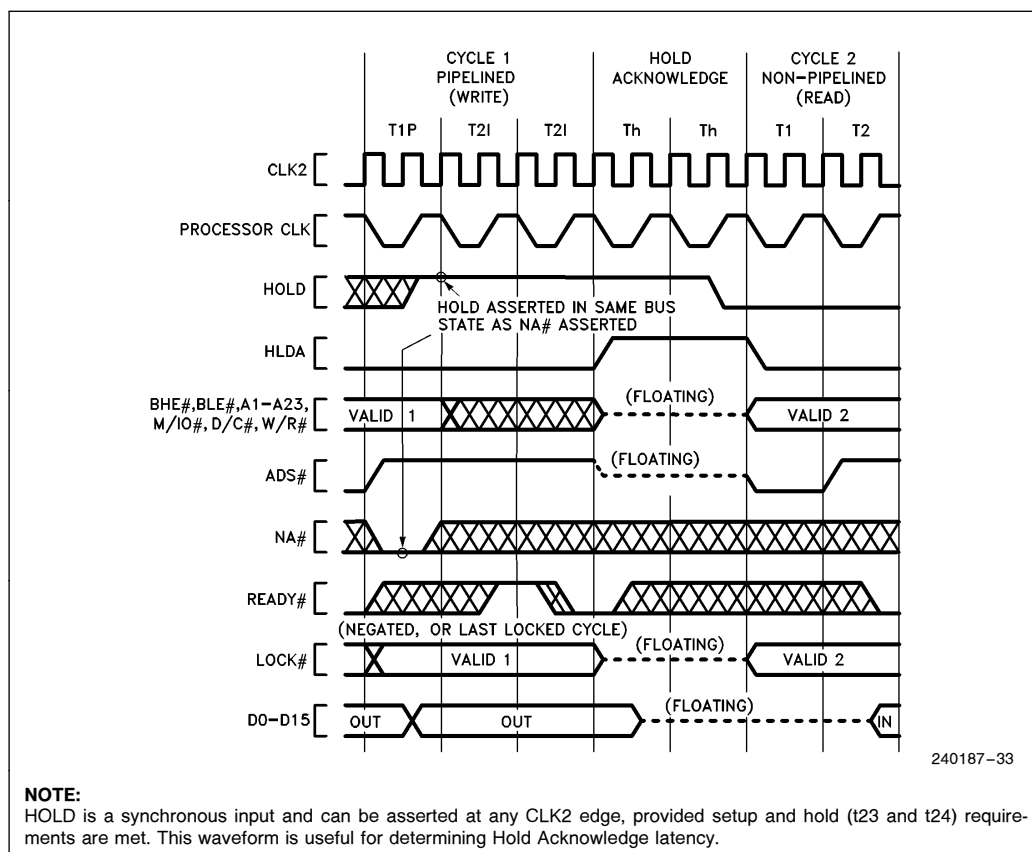
Figure 5.17. Requesting Hold from Active Bus (NA # inactive)

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the Intel386 SX Microprocessor, and at least 80 CLK2 periods if 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.

A 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 approximately  $(2^{20} + 60)$  CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the Intel386 SX Microprocessor attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the Intel386 SX Microprocessor performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.



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

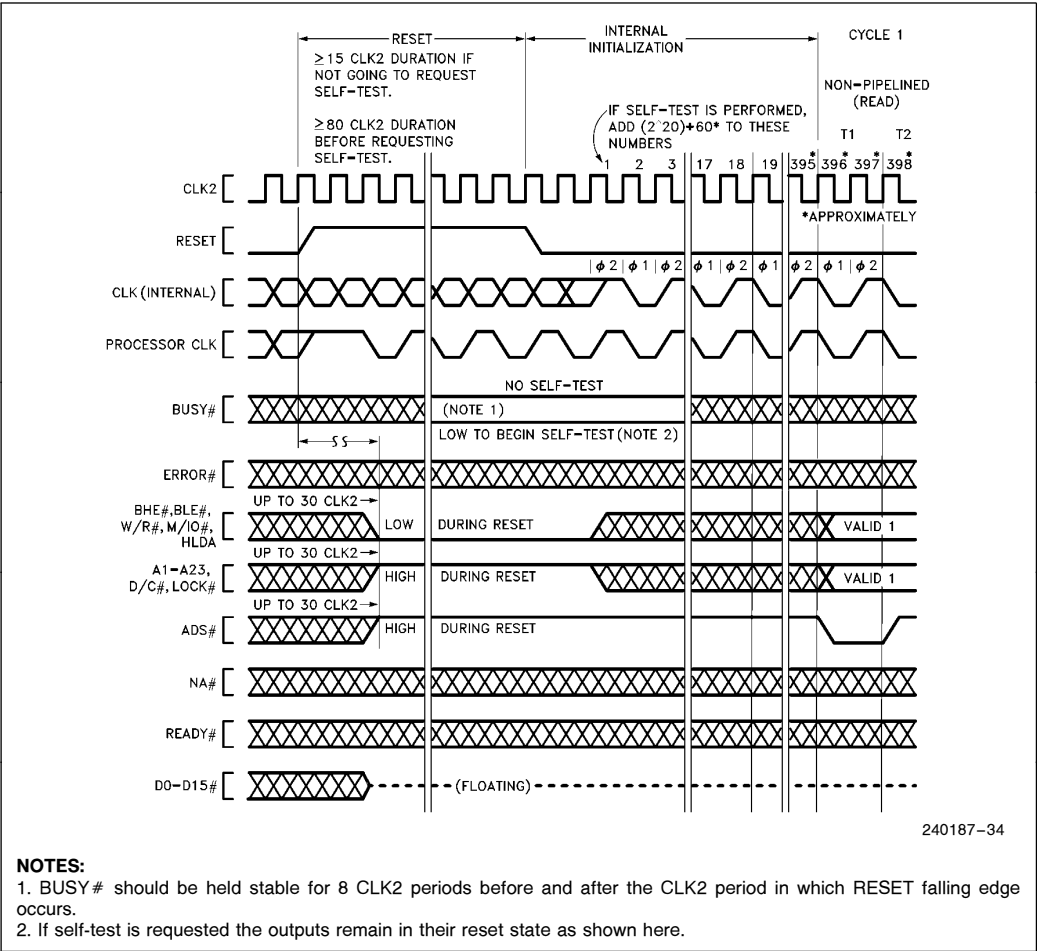


Figure 5.19. Bus Activity from Reset Until First Code Fetch

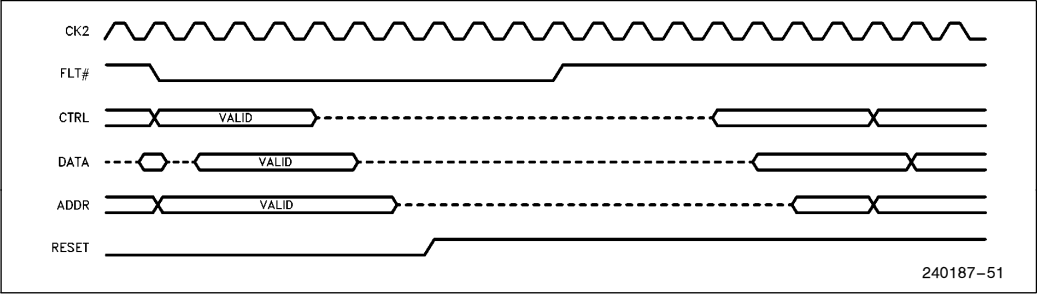


Figure 5.20. Entering and Exiting, FLT #

## 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 Intel386 SX Microprocessor passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all revision levels. Any non-zero signature indicates the unit is faulty.

## 5.6 Component and Revision Identifiers

To assist users, the Intel386 SX Microprocessor 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 Intel386 SX Microprocessor (the lower nibble, 03H, refers to the Intel386 DX Architecture. The upper nibble, 02H, refers to the second member of the Intel386 DX 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 Intel386 SX Microprocessor revision identifier will track that of the Intel386 DX CPU where possible.

The revision identifier is intended to assist 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**

Stepping	Revision Identifier
A0	04H
B	05H
C	08H
D	08H
E	08H

## 5.7 Coprocessor Interfacing

The Intel386 SX Microprocessor provides an automatic interface for the Intel387 SX numeric floating-point coprocessor. The Intel387 SX coprocessor uses an I/O mapped interface driven automatically by the Intel386 SX Microprocessor and assisted by three dedicated signals: BUSY#, ERROR# and PEREQ.

As the Intel386 SX Microprocessor 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 Intel387 SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the WAIT opcode (9BH) for Intel387 SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in Intel386 SX Microprocessor 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 instructions for high-speed coprocessor communication. The BUSY# and ERROR# inputs of the Intel386 SX Microprocessor may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 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 Intel386 SX CPU's on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the IOPL (I/O Privilege Level) mechanism.

The Intel387 SX numeric coprocessor interface is I/O mapped as shown in Table 5.8. Note that the Intel387 SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the Intel386 SX Microprocessor supports the Intel387 SX coprocessor, the Intel386 SX Microprocessor automatically generates bus cycles to the coprocessor interface addresses.

**Table 5.8. Numeric Coprocessor Port Addresses**

Address in Intel386 SX CPU I/O Space	Intel387 SX Coprocessor Register
8000F8H	Opcode Register
8000FCH/8000FEH*	Operand Register

\*Generated as 2nd bus cycle during Dword transfer.

To correctly map the Intel387 SX registers to the appropriate I/O addresses, connect the CMD0 and CMD1 lines of the Intel387 SX as listed in Table 5.9.

**Table 5.9. Connections for CMD0 and CMD1 Inputs for the Intel387 SX**

Signal	Connection
CMD0	Connect directly to Intel386 SX CPU A2 signal
CMD1	Connect to ground.

### Software Testing for Coprocessor Presence

When software is used to test for coprocessor (Intel387 SX) 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 Intel386 SX CPU's CR0 register.

## 6.0 PACKAGE THERMAL SPECIFICATIONS

The Intel386 SX Microprocessor is specified for operation when case temperature ( $T_c$ ) is within the range of 0°C–100°C. The case temperature may be measured in any environment, to determine whether the Intel386 SX Microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature ( $T_a$ ) is related to  $T_c$  and the thermal conductivity parameters  $\theta_{ja}$  and  $\theta_{jc}$  from the following equations (eqn. 3 is derived by eliminating the junction temperature ( $T_j$ ) between eqns. 1 and 2):

- 1)  $T_j = T_c + P \cdot \theta_{jc}$
- 2)  $T_a = T_j - P \cdot \theta_{ja}$
- 3)  $T_c = T_a + P \cdot [\theta_{ja} - \theta_{jc}]$

Values for  $\theta_{ja}$  and  $\theta_{jc}$  are given in Table 6.1 for the 100 lead fine pitch.  $\theta_{ja}$  is given at various airflows. The power (P) dissipated by the chip as heat is  $V_{cc} \cdot I_{cc}$ . A guaranteed maximum safe  $T_a$  can be calculated from eqn. 3 by using the maximum safe  $T_c$  of 100°C, along with the maximum power drawn by the chip in the given design, and  $\theta_{jc}$  and  $\theta_{ja}$  values from Table 6.1. (The  $\theta_{ja}$  value depends on the airflow, measured at the top of the chip, provided by the system ventilation.)

## 7.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the Intel386 SX Microprocessor, and its electrical specifications.

### 7.1 Power and Grounding

The Intel386 SX Microprocessor is implemented in CHMOS IV 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 V<sub>cc</sub> and 18 V<sub>ss</sub> pins separately feed functional units of the Intel386 SX Microprocessor.

Power and ground connections must be made to all external V<sub>cc</sub> and V<sub>ss</sub> pins of the Intel386 SX Microprocessor. On the circuit board, all V<sub>cc</sub> pins should be connected on a V<sub>cc</sub> plane and all V<sub>ss</sub> pins should be connected on a GND plane.

### POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the Intel386 SX Microprocessor. The Intel386 SX Microprocessor 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 Intel386 SX Microprocessor and decoupling capacitors as much as possible.

**Table 6.1. Thermal Resistances (°C/Watt)  $\theta_{jc}$  and  $\theta_{ja}$ .**

Package	$\theta_{jc}$	$\theta_{ja}$ 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	7.5	34.5	29.5	25.5	22.5	21.5	21



Table 7.1. Recommended Resistor Pull-ups to Vcc

Pin	Signal	Pull-up Value	Purpose
16	ADS#	20 K $\Omega$ $\pm$ 10%	Lightly pull ADS# inactive during Intel386™ SX CPU hold acknowledge states
26	LOCK#	20 K $\Omega$ $\pm$ 10%	Lightly pull LOCK# inactive during Intel386™ SX CPU hold acknowledge states

## RESISTOR RECOMMENDATIONS

The ERROR#, FLT# and BUSY# inputs have internal pull-up resistors of approximately 20 K $\Omega$  and the PEREQ input has an internal pull-down resistor of approximately 20 K $\Omega$  built into the Intel386 SX Microprocessor to keep these signals inactive when the Intel387 SX 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 Intel386 SX Microprocessor.**

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 $\Omega$  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 Intel386 SX Microprocessor 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:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.3. Intel386™ SX Microprocessor D.C. Characteristics—33 MHz, 25 MHz, 20 MHz and 16 MHz**

Symbol	Parameter	Min	Max	Unit	Test Condition
$V_{IL}$	Input LOW Voltage	-0.3	+0.8	V	
$V_{IH}$	Input HIGH Voltage	2.0	$V_{CC} + 0.3$	V	
$V_{ILC}$	CLK2 Input LOW Voltage	-0.3	+0.8	V	
$V_{IHC}$	CLK2 Input HIGH Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
$V_{OL}$	Output LOW Voltage $I_{OL} = 4$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OL} = 5$ mA: $BHE\#, BLE\#, W/R\#, D/C\#, M/IO\#, LOCK\#, ADS\#, HLDA$		0.45 0.45	V V	
$V_{OH}$	Output HIGH Voltage $I_{OH} = -1$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OH} = -0.2$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OH} = -0.9$ mA: $BHE\#, BLE\#, W/R\#, D/C\#, M/IO\#, LOCK\#, ADS\#, HLDA$ $I_{OH} = -0.18$ mA: $BHE\#, BLE\#, W/R\#, D/C\#, M/IO\#, LOCK\#, ADS\#, HLDA$	2.4 $V_{CC} - 0.5$ 2.4 $V_{CC} - 0.5$		V V V V	
$I_{LI}$	Input Leakage Current (for all pins except PEREQ, BUSY#, FLT# and ERROR#)		$\pm 15$	$\mu A$	$0V \leq V_{IN} \leq V_{CC}$
$I_{IH}$	Input Leakage Current (PEREQ pin)		200	$\mu A$	$V_{IH} = 2.4V$ , Note 1
$I_{IL}$	Input Leakage Current (BUSY#, ERROR# and FLT# pins)		-400	$\mu A$	$V_{IL} = 0.45V$ , Note 2
$I_{LO}$	Output Leakage Current		$\pm 15$	$\mu A$	$0.45V \leq V_{OUT} \leq V_{CC}$
$I_{CC}$	Supply Current CLK2 = 32 MHz: with 16 MHz Intel386 SX CPU CLK2 = 40 MHz: with 20 MHz Intel386 SX CPU CLK2 = 50 MHz: with 25 MHz Intel386 SX CPU CLK2 = 66 MHz: with 33 MHz Intel386 SX CPU		220 250 280 380	mA mA mA mA	(See Note 3) $I_{CC} \text{ typ} = 150$ mA $I_{CC} \text{ typ} = 180$ mA $I_{CC} \text{ typ} = 210$ mA $I_{CC} \text{ typ} = 290$ mA
$C_{IN}$	Input Capacitance		10	pF	$F_C = 1$ MHz, Note 4
$C_{OUT}$	Output or I/O Capacitance		12	pF	$F_C = 1$ MHz, Note 4
$C_{CLK}$	CLK2 Capacitance		20	pF	$F_C = 1$ MHz, Note 4

All values except  $I_{CC}$  tested at the minimum operating frequency of the part (CLK2 = 8 MHz).

**NOTES:**

1. PEREQ input has an internal pull-down resistor.
2. BUSY#, FLT# and ERROR# inputs each have an internal pull-up resistor.
3.  $I_{CC}$  max measurement at worst case frequency,  $V_{CC}$  and temperature, with 50 pF output load.
4. Not 100% tested.

Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.4. Low Power (LP) Intel386™ SX Microprocessor**  
**D.C. Characteristics—33 MHz, 25 MHz, 20 MHz, 16 MHz, and 12 MHz**

Symbol	Parameter	Min	Max	Unit	Test Condition
$V_{IL}$	Input LOW Voltage	-0.3	+0.8	V	
$V_{IH}$	Input HIGH Voltage	2.0	$V_{CC} + 0.3$	V	
$V_{ILC}$	CLK2 Input LOW Voltage	-0.3	+0.8	V	
$V_{IHC}$	CLK2 Input HIGH Voltage	$V_{CC} - 0.8$	$V_{CC} + 0.3$	V	
$V_{OL}$	Output LOW Voltage $I_{OL} = 4$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OL} = 5$ mA: BHE #, BLE #, W/R #, D/C #, M/IO #, LOCK #, ADS #, HLDA		0.45 0.45	V V	
$V_{OH}$	Output HIGH Voltage $I_{OH} = -1$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OH} = -0.2$ mA: $A_{23}-A_1, D_{15}-D_0$ $I_{OH} = -0.9$ mA: 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 $V_{CC} - 0.5$ 2.4  $V_{CC} - 0.5$		V V V  V	
$I_{LI}$	Input Leakage Current (for all pins except PEREQ, BUSY #, FLT # and ERROR #)		$\pm 15$	$\mu A$	$0V \leq V_{IN} \leq V_{CC}$
$I_{IH}$	Input Leakage Current (PEREQ pin)		200	$\mu A$	$V_{IH} = 2.4V$ , Note 1
$I_{IL}$	Input Leakage Current (BUSY #, ERROR # and FLT # pins)		-400	$\mu A$	$V_{IL} = 0.45V$ , Note 2
$I_{LO}$	Output Leakage Current		$\pm 15$	$\mu A$	$0.45V \leq V_{OUT} \leq V_{CC}$
$I_{CC}$	Supply Current CLK2 = 4 MHz CLK2 = 24 MHz: with 12 MHz Intel386 SX CPU CLK2 = 32 MHz: with 16 MHz Intel386 SX CPU CLK2 = 40 MHz: with 20 MHz Intel386 SX CPU CLK2 = 50 MHz: with 25 MHz Intel386 SX CPU CLK2 = 66 MHz: with 33 MHz Intel386 SX CPU		100 190 220 250 280 380	mA mA mA mA mA mA	(See Note 3) $I_{CC}$ typ = 50 mA $I_{CC}$ typ = 120 mA $I_{CC}$ typ = 150 mA $I_{CC}$ typ = 180 mA $I_{CC}$ typ = 210 mA $I_{CC}$ typ = 290 mA
$C_{IN}$	Input Capacitance		10	pF	$F_C = 1$ MHz, Note 4
$C_{OUT}$	Output or I/O Capacitance		12	pF	$F_C = 1$ MHz, Note 4
$C_{CLK}$	CLK2 Capacitance		20	pF	$F_C = 1$ MHz, Note 4

All values except  $I_{CC}$  tested at the minimum operating frequency of the part (CLK2 = 4 MHz).

**NOTES:**

- PEREQ input has an internal pull-down resistor.
- BUSY #, FLT # and ERROR # inputs each have an internal pull-up resistor.
- $I_{CC}$  max measurement at worst case frequency,  $V_{CC}$  and temperature, with 50 pF output load.
- Not 100% tested.



7.4 A.C. Specifications

The A.C. specifications given in Tables 7.5 through 7.8 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. Output delays are specified with minimum and maximum limits measured as shown. The minimum delay times are hold times provided to external circuitry. 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 operation.

Outputs ADS#, W/R#, D/C#, M/IO#, LOCK#, BHE#, BLE#, A<sub>23</sub>–A<sub>1</sub> and HLDA only change at the beginning of phase one. D<sub>15</sub>–D<sub>0</sub> (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ, FLT# and D<sub>15</sub>–D<sub>0</sub> (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.

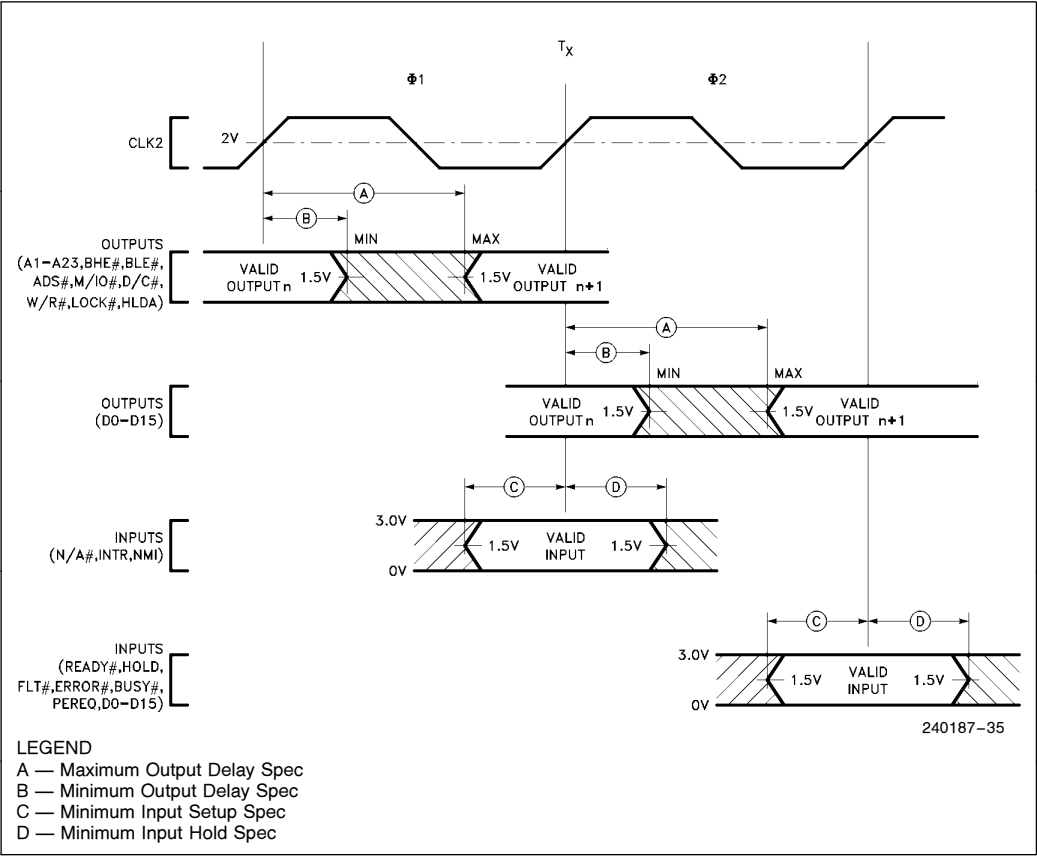


Figure 7.1. Drive Levels and Measurement Points for A.C. Specifications

**A.C. SPECIFICATIONS**Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$ **Table 7.5. Intel386™ SX Microprocessor A.C. Characteristics—33 MHz and 25 MHz**

Symbol	Parameter	33 MHz Intel386 SX		25 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
	Operating Frequency	4	33	4	25	MHz		Half CLK2 Frequency
t <sub>1</sub>	CLK2 Period	15	125	20	125	ns	7.3	
t <sub>2a</sub>	CLK2 HIGH Time	6.25		7		ns	7.3	at 2V <sup>(3)</sup>
t <sub>2b</sub>	CLK2 HIGH Time	4.0		4		ns	7.3	at (V <sub>CC</sub> - 0.8)V <sup>(3)</sup> ; Note 3
t <sub>3a</sub>	CLK2 LOW Time	6.25		7		ns	7.3	at 2V <sup>(3)</sup>
t <sub>3b</sub>	CLK2 LOW Time	4.5		5		ns	7.3	at 0.8V <sup>(3)</sup>
t <sub>4</sub>	CLK2 Fall Time		4		7	ns	7.3	(V <sub>CC</sub> - 0.8)V to 0.8V <sup>(3)</sup>
t <sub>5</sub>	CLK2 Rise Time		4		7	ns	7.3	0.8V to (V <sub>CC</sub> - 0.8)V <sup>(3)</sup>
t <sub>6</sub>	A <sub>23</sub> -A <sub>1</sub> Valid Delay	4	15	4	17	ns	7.5	C <sub>L</sub> = 50 pF <sup>(4)</sup>
t <sub>7</sub>	A <sub>23</sub> -A <sub>1</sub> Float Delay	4	20	4	30	ns	7.6	(Note 1)
t <sub>8</sub>	BHE #, BLE #, LOCK # Valid Delay	4	15	4	17	ns	7.5	C <sub>L</sub> = 50 pF <sup>(4)</sup>
t <sub>9</sub>	BHE #, BLE #, LOCK # Float Delay	4	20	4	30	ns	7.6	(Note 1)
t <sub>10</sub>	W/R #, M/IO #, D/C #, ADS # Valid Delay	4	15	4	17	ns	7.5	C <sub>L</sub> = 50 pF <sup>(4)</sup>
t <sub>11</sub>	W/R #, M/IO #, D/C #, ADS # Float Delay	4	20	4	30	ns	7.6	(Note 1)
t <sub>12</sub>	D <sub>15</sub> -D <sub>0</sub> Write Data Valid Delay	7	23	7	23	ns	7.5	C <sub>L</sub> = 50 pF <sup>(4, 5)</sup>
t <sub>12a</sub>	D <sub>15</sub> -D <sub>0</sub> Write Data Hold Time	2		2		ns		C <sub>L</sub> = 50 pF <sup>(4)</sup>
t <sub>13</sub>	D <sub>15</sub> -D <sub>0</sub> Write Data Float Delay	4	17	4	22	ns	7.6	(Note 1)
t <sub>14</sub>	HLDA Valid Delay	4	20	4	22	ns	7.6	C <sub>L</sub> = 75 pF <sup>(4)</sup>
t <sub>15</sub>	NA # Setup Time	5		5		ns	7.4	
t <sub>16</sub>	NA # Hold Time	2		3		ns	7.4	
t <sub>19</sub>	READY # Setup Time	7		9		ns	7.4	
t <sub>20</sub>	READY # Hold Time	4		4		ns	7.4	
t <sub>21</sub>	D <sub>15</sub> -D <sub>0</sub> Read Data Setup Time	5		7		ns	7.4	
t <sub>22</sub>	D <sub>15</sub> -D <sub>0</sub> Read Data Hold Time	3		5		ns	7.4	

Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.5. Intel386™ SX Microprocessor A.C. Characteristics—33 MHz and 25 MHz (Continued)**

Symbol	Parameter	33 MHz Intel386 SX		25 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
$t_{23}$	HOLD Setup Time	9		9		ns	7.4	
$t_{24}$	HOLD Hold Time	2		3		ns	7.4	
$t_{25}$	RESET Setup Time	5		8		ns	7.7	
$t_{26}$	RESET Hold Time	2		3		ns	7.7	
$t_{27}$	NMI, INTR Setup Time	5		6		ns	7.4	(Note 2)
$t_{28}$	NMI, INTR Hold Time	5		6		ns	7.4	(Note 2)
$t_{29}$	PEREQ, ERROR #, BUSY #, FLT # Setup Time	5		6		ns	7.4	(Note 2)
$t_{30}$	PEREQ, ERROR #, BUSY #, FLT # Hold Time	4		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 CL set at 50 pF. See Figures 7 and 8 for load capacitance derating curve.
5. Minimum time not 100% tested.

**Table 7.6. Low Power (LP) Intel386™ SX Microprocessor A.C. Characteristics—33 MHz and 25 MHz**

Symbol	Parameter	33 MHz Intel386 SX		25 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
	Operating Frequency	2	33	2	25	MHz		Half CLK2 Frequency
$t_1$	CLK2 Period	15	250	20	250	ns	7.3	
$t_{2a}$	CLK2 HIGH Time	6.25		7		ns	7.3	at 2V <sup>(3)</sup>
$t_{2b}$	CLK2 HIGH Time	4.0		4		ns	7.3	at $(V_{CC} - 0.8)V^{(3)}$ ; Note 3
$t_{3a}$	CLK2 LOW Time	6.25		7		ns	7.3	at 2V <sup>(3)</sup>
$t_{3b}$	CLK2 LOW Time	4.5		5		ns	7.3	at 0.8V <sup>(3)</sup>
$t_4$	CLK2 Fall Time		4		7	ns	7.3	$(V_{CC} - 0.8)V$ to 0.8V <sup>(3)</sup>
$t_5$	CLK2 Rise Time		4		7	ns	7.3	0.8V to $(V_{CC} - 0.8)V^{(3)}$
$t_6$	A <sub>23</sub> –A <sub>1</sub> Valid Delay	4	15	4	17	ns	7.5	$C_L = 50$ pF <sup>(4)</sup>
$t_7$	A <sub>23</sub> –A <sub>1</sub> Float Delay	4	20	4	30	ns	7.6	(Note 1)
$t_8$	BHE #, BLE #, LOCK # Valid Delay	4	15	4	17	ns	7.5	$C_L = 50$ pF <sup>(4)</sup>
$t_9$	BHE #, BLE #, LOCK # Float Delay	4	20	4	30	ns	7.6	(Note 1)

Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.6. Low Power (LP) Intel386™ SX Microprocessor**  
**A.C. Characteristics—33 MHz and 25 MHz (Continued)**

Symbol	Parameter	33 MHz Intel386 SX		25 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
$t_{10}$	W/R#, M/IO#, D/C#, ADS# Valid Delay	4	15	4	17	ns	7.5	$C_L = 50$ pF(4)
$t_{11}$	W/R#, M/IO#, D/C#, ADS# Float Delay	4	20	4	30	ns	7.6	(Note 1)
$t_{12}$	D <sub>15</sub> –D <sub>0</sub> Write Data Valid Delay	7	23	7	23	ns	7.5	$C_L = 50$ pF(4, 5)
$t_{12a}$	D <sub>15</sub> –D <sub>0</sub> Write Data Hold Time	2		2		ns		$C_L = 50$ pF(4)
$t_{13}$	D <sub>15</sub> –D <sub>0</sub> Write Data Float Delay	4	17	4	22	ns	7.6	(Note 1)
$t_{14}$	HLDA Valid Delay	4	20	4	22	ns	7.6	$C_L = 50$ pF(4)
$t_{15}$	NA# Setup Time	5		5		ns	7.4	
$t_{16}$	NA# Hold Time	2		3		ns	7.4	
$t_{19}$	READY# Setup Time	7		9		ns	7.4	
$t_{20}$	READY# Hold Time	4		4		ns	7.4	
$t_{21}$	D <sub>15</sub> –D <sub>0</sub> Read Data Setup Time	5		7		ns	7.4	
$t_{22}$	D <sub>15</sub> –D <sub>0</sub> Read Data Hold Time	3		5		ns	7.4	
$t_{23}$	HOLD Setup Time	9		9		ns	7.4	
$t_{24}$	HOLD Hold Time	2		3		ns	7.4	
$t_{25}$	RESET Setup Time	5		8		ns	7.7	
$t_{26}$	RESET Hold Time	2		3		ns	7.7	
$t_{27}$	NMI, INTR Setup Time	5		6		ns	7.4	(Note 2)
$t_{28}$	NMI, INTR Hold Time	5		6		ns	7.4	(Note 2)
$t_{29}$	PEREQ, ERROR#, BUSY#, FLT# Setup Time	5		6		ns	7.4	(Note 2)
$t_{30}$	PEREQ, ERROR#, BUSY#, FLT# Hold Time	4		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. See Figures 7 and 8 for load capacitance derating curve.
5. Minimum time not 100% tested.

Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.7. Intel386™ SX A.C. Characteristics—20 MHz and 16 MHz**

Symbol	Parameter	20 MHz Intel386 SX		16 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
	Operating Frequency	4	20	4	16	MHz		Half CLK2 Frequency
$t_1$	CLK2 Period	25	125	31	125	ns	7.3	
$t_{2a}$	CLK2 HIGH Time	8		9		ns	7.3	at $2V^{(3)}$
$t_{2b}$	CLK2 HIGH Time	5		5		ns	7.3	at $(V_{CC} - 0.8)V^{(3)}$
$t_{3a}$	CLK2 LOW Time	8		9		ns	7.3	at $2V^{(3)}$
$t_{3b}$	CLK2 LOW Time	6		7		ns	7.3	at $0.8V^{(3)}$
$t_4$	CLK2 Fall Time		8		8	ns	7.3	$(V_{CC} - 0.8)V$ to $0.8V^{(3)}$
$t_5$	CLK2 Rise Time		8		8	ns	7.3	$0.8V$ to $(V_{CC} - 0.8)V^{(3)}$
$t_6$	$A_{23}-A_1$ Valid Delay	4	30	4	36	ns	7.5	$C_L = 120 \text{ pF}^{(4)}$
$t_7$	$A_{23}-A_1$ Float Delay	4	32	4	40	ns	7.6	(Note 1)
$t_8$	BHE #, BLE #, LOCK # Valid Delay	4	30	4	36	ns	7.5	$C_L = 75 \text{ pF}^{(4)}$
$t_9$	BHE #, BLE #, LOCK # Float Delay	4	32	4	40	ns	7.6	(Note 1)
$t_{10a}$	M/IO # D/C # Valid Delay	6	28	6	33	ns	7.5	$C_L = 75 \text{ pF}^{(4)}$
$t_{10b}$	W/R #, ADS # Valid Delay		26					
$t_{11}$	W/R #, M/IO #, D/C #, ADS # Float Delay	6	30	6	35	ns	7.6	(Note 1)
$t_{12}$	D <sub>15</sub> –D <sub>0</sub> Write Data Valid Delay	4	38	4	40	ns	7.5	$C_L = 120 \text{ pF}^{(4)}$
$t_{13}$	D <sub>15</sub> –D <sub>0</sub> Write Data Float Delay	4	27	4	35	ns	7.6	(Note 1)
$t_{14}$	HLDA Valid Delay	4	28	4	33	ns	7.5	$C_L = 75 \text{ pF}^{(4)}$
$t_{15}$	NA # Setup Time	5		5		ns	7.4	
$t_{16}$	NA # Hold Time	12		21		ns	7.4	
$t_{19}$	READY # Setup Time	12		19		ns	7.4	
$t_{20}$	READY # Hold Time	4		4		ns	7.4	
$t_{21}$	D <sub>15</sub> –D <sub>0</sub> Read Data Setup Time	9		9		ns	7.4	
$t_{22}$	D <sub>15</sub> –D <sub>0</sub> Read Data Hold Time	6		6		ns	7.4	
$t_{23}$	HOLD Setup Time	17		26		ns	7.4	
$t_{24}$	HOLD Hold Time	5		5		ns	7.4	
$t_{25}$	RESET Setup Time	12		13		ns	7.7	
$t_{26}$	RESET Hold Time	4		4		ns	7.7	





Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

**Table 7.7. Intel386™ SX A.C. Characteristics—20 MHz and 16 MHz (Continued)**

Symbol	Parameter	20 MHz Intel386 SX		16 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
$t_{27}$	NMI, INTR Setup Time	16		16		ns	7.4	(Note 2)
$t_{28}$	NMI, INTR Hold Time	16		16		ns	7.4	(Note 2)
$t_{29}$	PEREQ, ERROR #, BUSY #, FLT # Setup Time	14		16		ns	7.4	(Note 2)
$t_{30}$	PEREQ, ERROR #, BUSY #, FLT # Hold Time	5		5		ns	7.4	(Note 2)

**Table 7.8. Low Power (LP) Intel386™ SX A.C. Characteristics—20 MHz, 16 MHz and 12 MHz**

Symbol	Parameter	20 MHz Intel386 SX		16 MHz Intel386 SX		12 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max	Min	Max			
	Operating Frequency	2	20	2	16	2	12.5	MHz		Half CLK2 Frequency
$t_1$	CLK2 Period	25	250	31	250	40	250	ns	7.3	
$t_{2a}$	CLK2 HIGH Time	8		9		11		ns	7.3	at 2V (Note 3)
$t_{2b}$	CLK2 HIGH Time	5		5		7		ns	7.3	at $(V_{CC} - 0.8V)^{(3)}$
$t_{3a}$	CLK2 LOW Time	8		9		11		ns	7.3	at 2V <sup>(3)</sup>
$t_{3b}$	CLK2 LOW Time	6		7		9		ns	7.3	at 0.8V <sup>(3)</sup>
$t_4$	CLK2 Fall Time		8		8		8	ns	7.3	$(V_{CC} - 0.8V)$ to $0.8V^{(3)}$
$t_5$	CLK2 Rise Time		8		8		8	ns	7.3	$0.8V$ to $(V_{CC} - 0.8V)^{(3)}$
$t_6$	$A_{23}-A_1$ Valid Delay	4	30	4	36	4	42	ns	7.5	$C_L = 120$ pF <sup>(4)</sup>
$t_7$	$A_{23}-A_1$ Float Delay	4	32	4	40	4	45	ns	7.6	(Note 1)
$t_8$	BHE #, BLE #, LOCK # Valid Delay	4	30	4	36	4	36	ns	7.5	$C_L = 75$ pF
$t_9$	BHE #, BLE #, LOCK # Float Delay	4	32	4	40	4	40	ns	7.6	(Note 1)
$t_{10}$	M/IO #, D/C #, W/R #, ADS # Valid Delay	6	28	6	33	4	33	ns	7.5	$C_L = 75$ pF
$t_{11}$	M/IO #, D/C #, W/R #, ADS # Float Delay	6	30	6	35	4	35	ns	7.6	(Note 1)
$t_{12}$	D15–D0 Write Data Valid Delay	4	38	4	40	4	50	ns	7.5	$C_L = 120$ pF <sup>(4)</sup>
$t_{13}$	D15–D0 Write Data Float Delay	4	27	4	35	4	40	ns	7.6	(Note 1)
$t_{14}$	HLDA Valid Delay	4	28	6	33	4	33	ns	7.5	$C_L = 75$ pF <sup>(4)</sup>
$t_{15}$	NA # Setup Time	5		5		7		ns	7.4	
$t_{16}$	NA # Hold Time	12		21		21		ns	7.4	

Functional operating range:  $V_{CC} = 5V \pm 10\%$ ;  $T_{CASE} = 0^{\circ}C$  to  $100^{\circ}C$

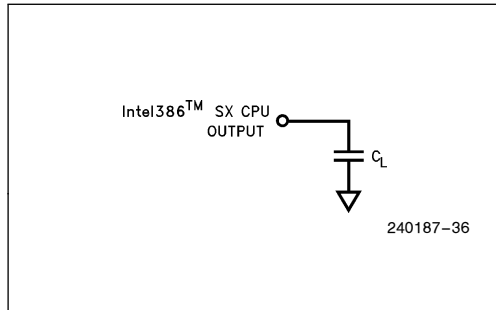
**Table 7.8. Low Power (LP) Intel386™ SX**  
**A.C. Characteristics—20 MHz, 16 MHz and 12 MHz (Continued)**

Symbol	Parameter	20 MHz Intel386 SX		16 MHz Intel386 SX		12 MHz Intel386 SX		Unit	Figure	Notes
		Min	Max	Min	Max	Min	Max			
$t_{19}$	READY# Setup Time	12		19		19		ns	7.4	
$t_{20}$	READY# Hold Time	4		4		4		ns	7.4	
$t_{21}$	D15–D0 Read Data Setup Time	9		9		9		ns	7.4	
$t_{22}$	D15–D0 Read Data Hold Time	6		6		6		ns	7.4	
$t_{23}$	HOLD Setup Time	17		26		26		ns	7.4	
$t_{24}$	HOLD Hold Time	5		5		7		ns	7.4	
$t_{25}$	RESET Setup Time	12		13		15		ns	7.7	
$t_{26}$	RESET Hold Time	4		4		6		ns	7.7	
$t_{27}$	NMI, INTR Setup Time	16		16		16		ns	7.4	(Note 2)
$t_{28}$	NMI, INTR Hold Time	16		16		16		ns	7.4	(Note 2)
$t_{29}$	PEREQ, ERROR#, BUSY#, FLT# Setup Time	14		16		16		ns	7.4	(Note 2)
$t_{30}$	PEREQ, ERROR#, BUSY#, FLT# Hold Time	5		5		5		ns	7.4	(Note 2)

**NOTES:**

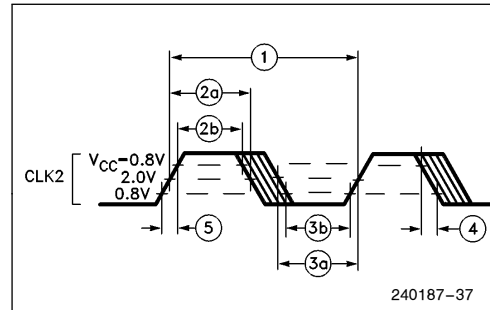
1. Float condition occurs when maximum output current becomes less than  $I_{LO}$  in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
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 Figures 7.8 though 7.10 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**

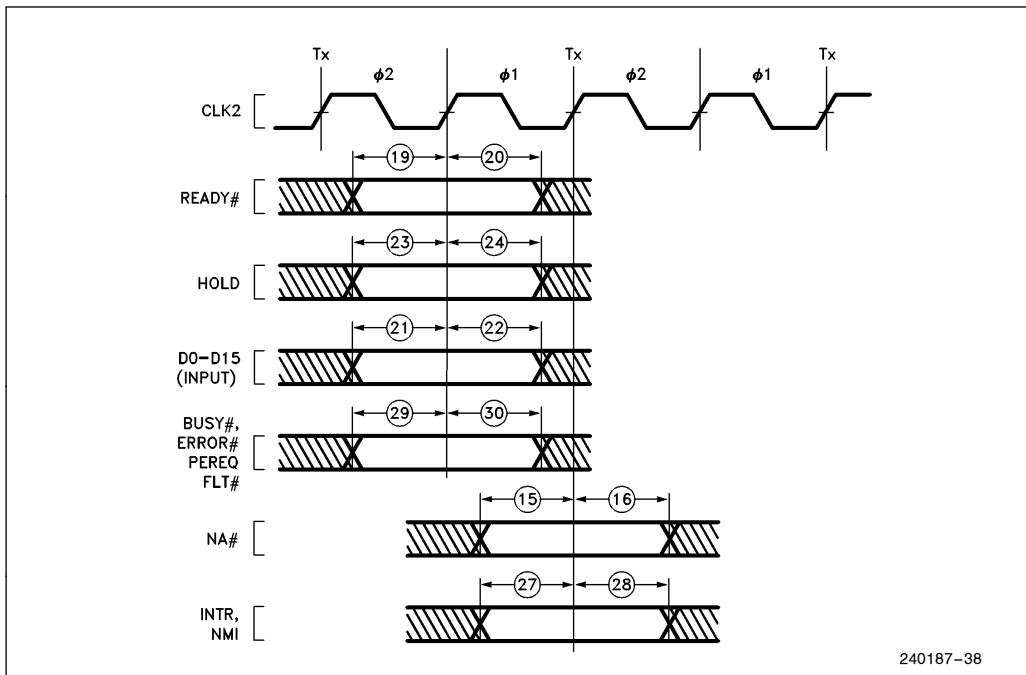


Figure 7.4. A.C. Timing Waveforms—Input Setup and Hold Timing

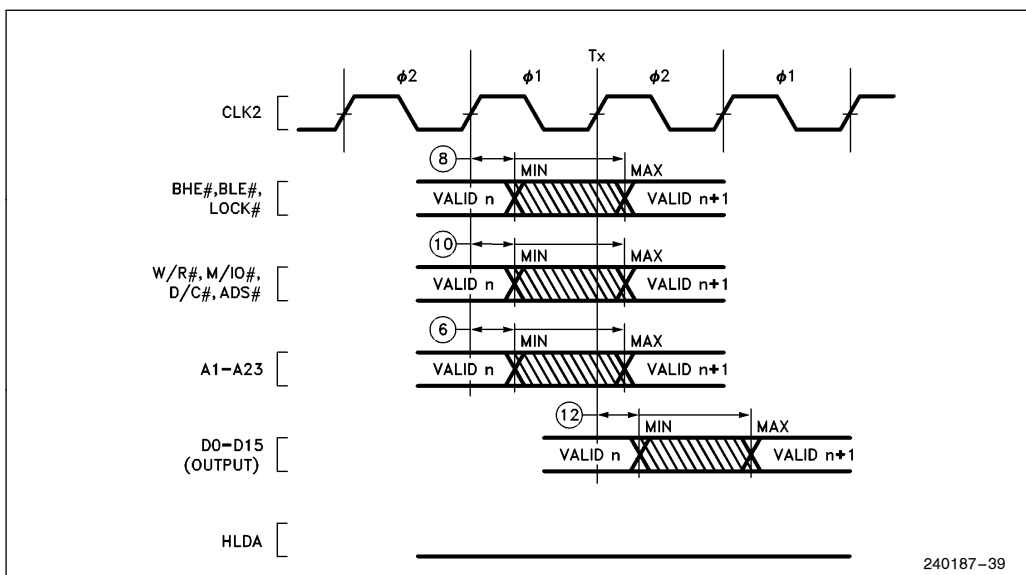


Figure 7.5. A.C. Timing Waveforms—Output Valid Delay Timing

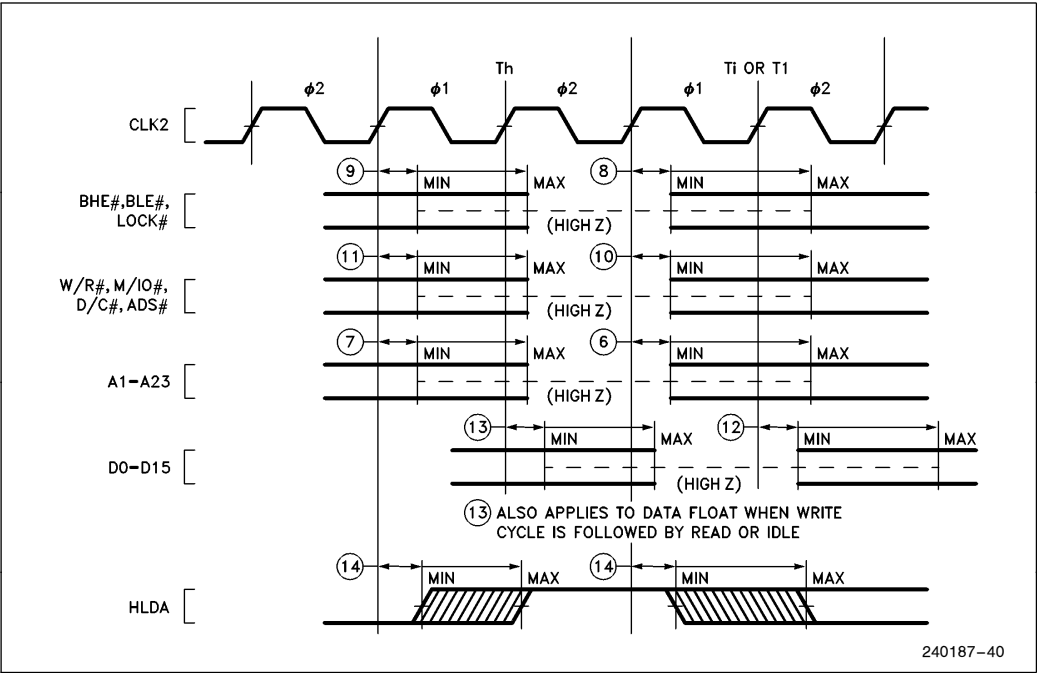


Figure 7.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing

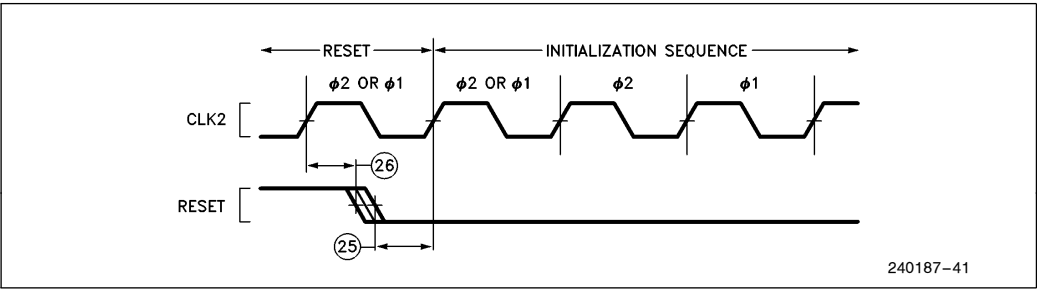
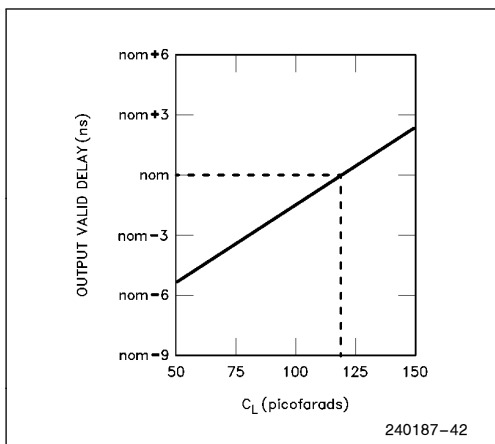
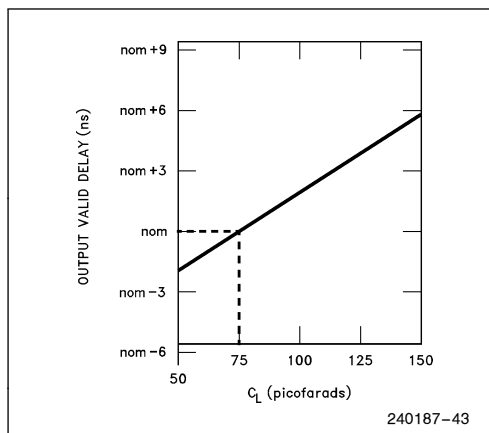


Figure 7.7. A.C. Timing Waveforms—RESET Setup and Hold Timing and Internal Phase

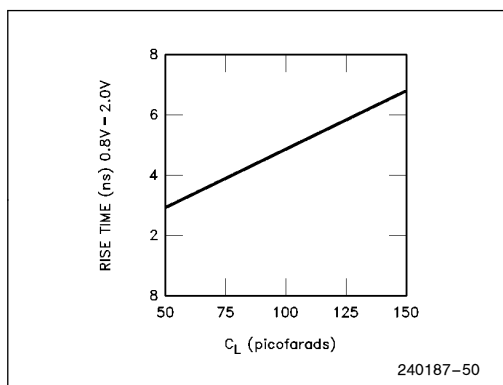




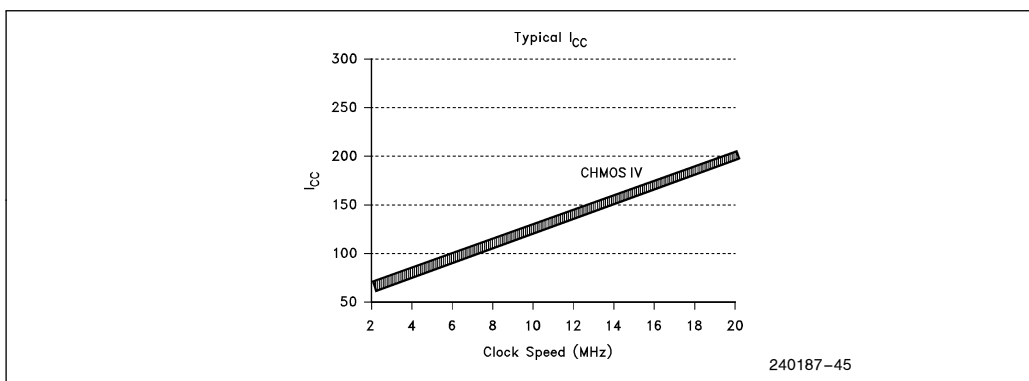
**Figure 7.8. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature ( $C_L = 120$  pF)**



**Figure 7.9. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature ( $C_L = 75$  pF)**



**Figure 7.10. Typical Output Rise Time versus Load Capacitance at Maximum Operating Temperature**



**Figure 7.11. Typical  $I_{CC}$  vs Frequency**

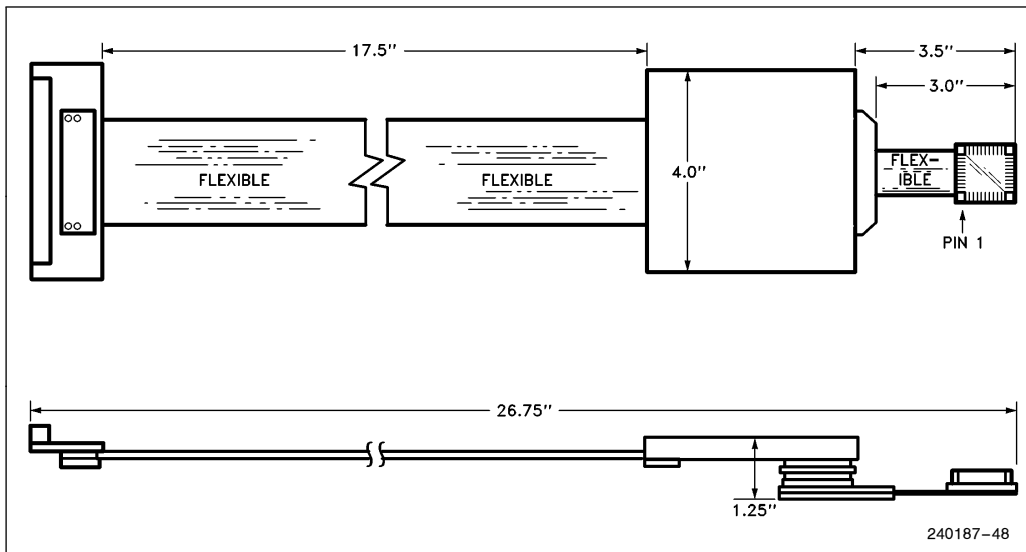


Figure 7.12. Preliminary ICE™-Intel386™ SX Emulator User Cable with PQFP Adapter

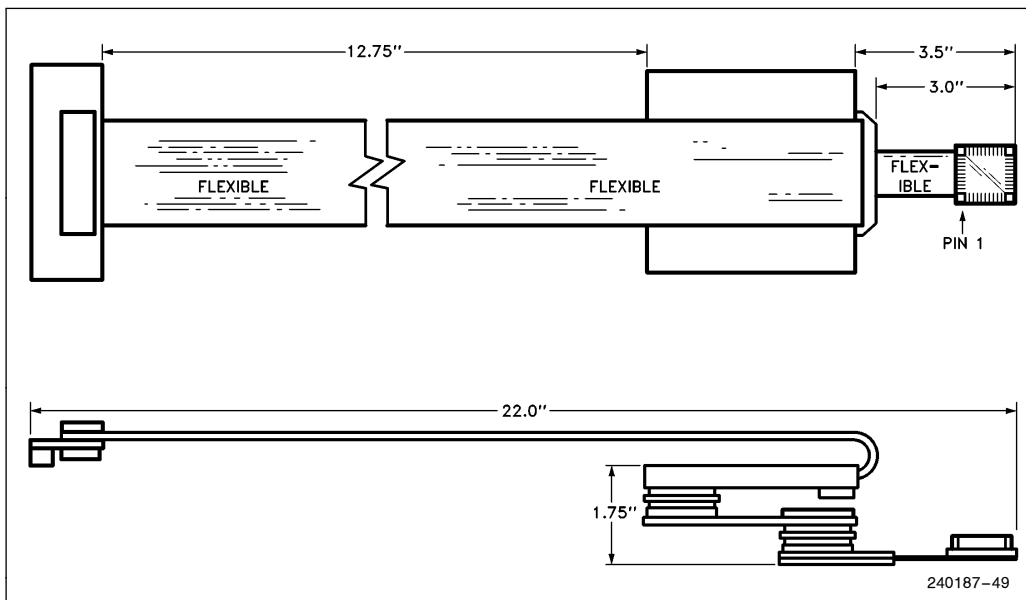


Figure 7.13. Preliminary ICE™-Intel386™ SX Emulator User Cable with OIB and PQFP Adapter

### 7.5 Designing for the ICE™-Intel386™ SX Emulator

ICE-Intel386 SX is the in-circuit emulator for the Intel386™ SX CPU. The ICE-386 SX emulator provides a 100-pin fine pitch flat-pack probe for connection to a socket located on the target system.

Sockets that accept this probe are available from 3M (part #2-0100-07243-000) or from AMP (part #821959-1 and part #821949-4). The ICE-386 SX emulator probe attaches to the target system via an adapter that replaces the Intel386 SX component in the target system.

Due to the high operating frequency of Intel386 SX CPU based systems, there is no buffering between the Intel386 SX emulation processor (on the emulator probe) and the target system. A direct result of the non-buffered interconnect is that the ICE-Intel386 SX emulator shares the address and data busses with the target system.

In order to avoid problems with the shared bus and maintain signal integrity, the system designer must adhere to the following guidelines:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles (initiated by assertion of ADS#) of the Intel386 SX CPU, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an Intel386 SX CPU would, and responds to RESET later. Correct phase of the response is guaranteed.

In order to avoid problems that might arise due to the shared busses, an Optional use Isolation Board (OIB) is included with the emulator hardware. The OIB may be used to provide buffering between the emulation processor and the target system, but inserts a delay of approximately 10 ns in signal path.

In addition to the above considerations, the ICE-386 SX emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the Intel386 SX CPU system.

**Capacitive Loading:** ICE-Intel386 SX adds up to 27 pF to each Intel386 SX CPU signal.

**Drive Requirements:** ICE-Intel386 SX 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 Intel386 SX CPU emulation processor, which has standard drive and loading capability listed in Tables 7.3 and 7.4.

**Power Requirements:** For noise immunity and CMOS latch-up protection the ICE-Intel386 SX emulator processor module is powered by the user system.

The circuitry on the processor module draws up to 1.4A including the maximum Intel386 SX CPU  $I_{CC}$  from the user Intel386 SX CPU socket.

**Intel386 SX CPU Location and Orientation:** The ICE-Intel386 SX emulator processor module may require lateral clearance. Figure 7.12 shows the clearance requirements of the iMP adapter. The optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figure 7.12, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 7.13.

**Optional Isolation Board (OIB) and the CLK2 speed reduction:** Due to the unbuffered probe design, the ICE-Intel386 SX emulator is susceptible to errors on the user's bus. The OIB allows the ICE-Intel386 SX emulator 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 20 MHz.

## 8.0 DIFFERENCES BETWEEN THE Intel386™ SX CPU AND THE Intel386™ DX CPU

The following are the major differences between the Intel386 SX CPU and the Intel386 DX CPU:

1. The Intel386 SX CPU 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 Intel386 DX CPU uses four byte selects, BE0#-BE3#, to distinguish between the different bytes on its 32-bit bus.
2. The Intel386 SX CPU has no bus sizing option. The Intel386 DX CPU can select between either a 32-bit bus or a 16-bit bus by use of the BS16# input. The Intel386 SX CPU has a 16-bit bus size.
3. The NA# pin operation in the Intel386 SX CPU is identical to that of the NA# pin on the Intel386 DX CPU with one exception: the Intel386 DX CPU NA# pin cannot be activated on 16-bit bus cycles (where BS16# is LOW in the Intel386 DX CPU case), whereas NA# can be activated on any Intel386 SX CPU bus cycle.
4. The contents of all Intel386 SX CPU registers at reset are identical to the contents of the Intel386 DX CPU registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

in Intel386 DX CPU, DH = 3 indicates Intel386 DX CPU after reset

DL = revision number;

in Intel386 SX CPU, DH = 23H indicates Intel386 SX CPU after reset

DL = revision number.

5. The Intel386 DX CPU uses  $A_{31}$  and  $M/IO\#$  as selects for the numerics coprocessor. The Intel386 SX CPU uses  $A_{23}$  and  $M/IO\#$  as selects.
6. The Intel386 DX CPU prefetch unit fetches code in four-byte units. The Intel386 SX CPU prefetch unit reads two bytes as one unit (like the 80286). In BS16 mode, the Intel386 DX CPU takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the Intel386 DX CPU will fetch all four bytes before addressing the new request.
7. Both Intel386 DX CPU and Intel386 SX CPU have the same logical address space. The only difference is that the Intel386 DX CPU has a 32-bit physical address space and the Intel386 SX CPU has a 24-bit physical address space. The Intel386 SX CPU has a physical memory address space of up to 16 megabytes instead of the 4 gigabytes available to the Intel386 DX CPU. Therefore, in Intel386 SX CPU systems, the operating system must be aware of this physical memory limit and should allocate memory for applications programs within this limit. If a Intel386 DX CPU system uses only the lower 16 megabytes of physical address, then there will be no extra effort required to migrate Intel386 DX CPU software to the Intel386 SX CPU. Any application which uses more than 16 megabytes of memory can run on the Intel386 SX CPU if the operating system utilizes the Intel386 SX CPU's paging mechanism. In spite of this difference in physical address space, the Intel386 SX CPU and Intel386 DX CPU can run the same operating systems and applications within their respective physical memory constraints.
8. The Intel386 SX has an input called  $FLT\#$  which tri-states all bidirectional and output pins, including  $HLDA\#$ , when asserted. It is used with ON Circuit Emulation (ONCE). In the Intel386 DX CPU,  $FLT\#$  is found only on the plastic quad flat package version and not on the ceramic pin grid array version. For a more detailed explanation of  $FLT\#$  and testability, please refer to section 5.4.

## 9.0 INSTRUCTION SET

This section describes the 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 instructions.

### 9.1 Intel386™ SX CPU 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 Intel386 SX Microprocessor operating at 16 MHz). The actual clock count of an Intel386 SX Microprocessor 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 9-1. Instruction Set Clock Count Summary

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				
GENERAL DATA TRANSFER									
MOV = Move:									
Register to Register/Memory	<table><tr><td>1 0 0 0 1 0 0 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 0 0 w	mod reg	r/m	2/2	2/2*	b	h	
1 0 0 0 1 0 0 w	mod reg	r/m							
Register/Memory to Register	<table><tr><td>1 0 0 0 1 0 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 0 1 w	mod reg	r/m	2/4	2/4*	b	h	
1 0 0 0 1 0 1 w	mod reg	r/m							
Immediate to Register/Memory	<table><tr><td>1 1 0 0 0 1 1 w</td><td>mod 0 0 0</td><td>r/m</td></tr></table> immediate data	1 1 0 0 0 1 1 w	mod 0 0 0	r/m	2/2	2/2*	b	h	
1 1 0 0 0 1 1 w	mod 0 0 0	r/m							
Immediate to Register (short form)	<table><tr><td>1 0 1 1 w</td><td>reg</td></tr></table> immediate data	1 0 1 1 w	reg	2	2				
1 0 1 1 w	reg								
Memory to Accumulator (short form)	<table><tr><td>1 0 1 0 0 0 0 w</td><td>full displacement</td></tr></table>	1 0 1 0 0 0 0 w	full displacement	4*	4*	b	h		
1 0 1 0 0 0 0 w	full displacement								
Accumulator to Memory (short form)	<table><tr><td>1 0 1 0 0 0 1 w</td><td>full displacement</td></tr></table>	1 0 1 0 0 0 1 w	full displacement	2*	2*	b	h		
1 0 1 0 0 0 1 w	full displacement								
Register Memory to Segment Register	<table><tr><td>1 0 0 0 1 1 1 0</td><td>mod sreg3</td><td>r/m</td></tr></table>	1 0 0 0 1 1 1 0	mod sreg3	r/m	2/5	22/23	b	h, i, j	
1 0 0 0 1 1 1 0	mod sreg3	r/m							
Segment Register to Register/Memory	<table><tr><td>1 0 0 0 1 1 0 0</td><td>mod sreg3</td><td>r/m</td></tr></table>	1 0 0 0 1 1 0 0	mod sreg3	r/m	2/2	2/2	b	h	
1 0 0 0 1 1 0 0	mod sreg3	r/m							
MOVSBX = Move With Sign Extension									
Register From Register/Memory	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 1 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 w	mod reg	r/m	3/6*	3/6*	b	h
0 0 0 0 1 1 1 1	1 0 1 1 1 1 1 w	mod reg	r/m						
MOVZX = Move With Zero Extension									
Register From Register/Memory	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 0 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 w	mod reg	r/m	3/6*	3/6*	b	h
0 0 0 0 1 1 1 1	1 0 1 1 0 1 1 w	mod reg	r/m						
PUSH = Push:									
Register/Memory	<table><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 1 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 1 0	r/m	5/7*	7/9*	b	h	
1 1 1 1 1 1 1 1	mod 1 1 0	r/m							
Register (short form)	<table><tr><td>0 1 0 1 0</td><td>reg</td></tr></table>	0 1 0 1 0	reg	2	4	b	h		
0 1 0 1 0	reg								
Segment Register (ES, CS, SS or DS) (short form)	<table><tr><td>0 0 0 sreg2</td><td>1 1 0</td></tr></table>	0 0 0 sreg2	1 1 0	2	4	b	h		
0 0 0 sreg2	1 1 0								
Segment Register (ES, CS, SS, DS, FS or GS)	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 sreg3</td><td>0 0 0</td></tr></table>	0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0	2	4	b	h	
0 0 0 0 1 1 1 1	1 0 sreg3	0 0 0							
Immediate	<table><tr><td>0 1 1 0 1 0 s</td><td>0</td></tr></table> immediate data	0 1 1 0 1 0 s	0	2	4	b	h		
0 1 1 0 1 0 s	0								
PUSHA = Push All	<table><tr><td>0 1 1 0 0 0 0 0</td></tr></table>	0 1 1 0 0 0 0 0	18	34	b	h			
0 1 1 0 0 0 0 0									
POP = Pop									
Register/Memory	<table><tr><td>1 0 0 0 1 1 1 1</td><td>mod 0 0 0</td><td>r/m</td></tr></table>	1 0 0 0 1 1 1 1	mod 0 0 0	r/m	5/7	7/9	b	h	
1 0 0 0 1 1 1 1	mod 0 0 0	r/m							
Register (short form)	<table><tr><td>0 1 0 1 1</td><td>reg</td></tr></table>	0 1 0 1 1	reg	6	6	b	h		
0 1 0 1 1	reg								
Segment Register (ES, CS, SS or DS) (short form)	<table><tr><td>0 0 0 sreg</td><td>2 1 1 1</td></tr></table>	0 0 0 sreg	2 1 1 1	7	25	b	h, i, j		
0 0 0 sreg	2 1 1 1								
Segment Register (ES, CS, SS or DS), FS or GS	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 sreg</td><td>3 0 0 1</td></tr></table>	0 0 0 0 1 1 1 1	1 0 sreg	3 0 0 1	7	25	b	h, i, j	
0 0 0 0 1 1 1 1	1 0 sreg	3 0 0 1							
POPA = Pop All	<table><tr><td>0 1 1 0 0 0 0 1</td></tr></table>	0 1 1 0 0 0 0 1	24	40	b	h			
0 1 1 0 0 0 0 1									
XCHG = Exchange									
Register/Memory With Register	<table><tr><td>1 0 0 0 0 1 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 0 1 1 w	mod reg	r/m	3/5**	3/5**	b, f	f, h	
1 0 0 0 0 1 1 w	mod reg	r/m							
Register With Accumulator (short form)	<table><tr><td>1 0 0 1 0</td><td>reg</td></tr></table>	1 0 0 1 0	reg	3	3				
1 0 0 1 0	reg								
IN = Input from:									
Fixed Port	<table><tr><td>1 1 1 0 0 1 0 w</td><td>port number</td></tr></table>	1 1 1 0 0 1 0 w	port number	†26			s/t,m		
1 1 1 0 0 1 0 w	port number								
Variable Port	<table><tr><td>1 1 1 0 1 1 0 w</td></tr></table>	1 1 1 0 1 1 0 w	†27			s/t,m			
1 1 1 0 1 1 0 w									
OUT = Output to:									
Fixed Port	<table><tr><td>1 1 1 0 0 1 1 w</td><td>port number</td></tr></table>	1 1 1 0 0 1 1 w	port number	†24			s/t,m		
1 1 1 0 0 1 1 w	port number								
Variable Port	<table><tr><td>1 1 1 0 1 1 1 w</td></tr></table>	1 1 1 0 1 1 1 w	†25			s/t,m			
1 1 1 0 1 1 1 w									
LEA = Load EA to Register	<table><tr><td>1 0 0 0 1 1 0 1</td><td>mod reg</td><td>r/m</td></tr></table>	1 0 0 0 1 1 0 1	mod reg	r/m	2	2			
1 0 0 0 1 1 0 1	mod reg	r/m							

Clk Count  
Virtual  
8086 Mode

Table 9-1. 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	<table><tr><td>1 1 0 0 0 1 0 1</td><td>mod reg</td><td>r/m</td></tr></table>	1 1 0 0 0 1 0 1	mod reg	r/m	7*	26*/28*	b	h, i, j	
1 1 0 0 0 1 0 1	mod reg	r/m							
LES = Load Pointer to ES	<table><tr><td>1 1 0 0 0 1 0 0</td><td>mod reg</td><td>r/m</td></tr></table>	1 1 0 0 0 1 0 0	mod reg	r/m	7*	26*/28*	b	h, i, j	
1 1 0 0 0 1 0 0	mod reg	r/m							
LFS = Load Pointer to FS	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 0 1 0 0</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 0	mod reg	r/m	7*	29*/31*	b	h, i, j
0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 0	mod reg	r/m						
LGS = Load Pointer to GS	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 0 1 0 1</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 1	mod reg	r/m	7*	26*/28*	b	h, i, j
0 0 0 0 1 1 1 1	1 0 1 1 0 1 0 1	mod reg	r/m						
LSS = Load Pointer to SS	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 1 0 0 1 0</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 0	mod reg	r/m	7*	26*/28*	b	h, i, j
0 0 0 0 1 1 1 1	1 0 1 1 0 0 1 0	mod reg	r/m						
FLAG CONTROL									
CLC = Clear Carry Flag	<table><tr><td>1 1 1 1 1 0 0 0</td></tr></table>	1 1 1 1 1 0 0 0	2	2					
1 1 1 1 1 0 0 0									
CLD = Clear Direction Flag	<table><tr><td>1 1 1 1 1 1 0 0</td></tr></table>	1 1 1 1 1 1 0 0	2	2					
1 1 1 1 1 1 0 0									
CLI = Clear Interrupt Enable Flag	<table><tr><td>1 1 1 1 1 0 1 0</td></tr></table>	1 1 1 1 1 0 1 0	8	8		m			
1 1 1 1 1 0 1 0									
CLTS = Clear Task Switched Flag	<table><tr><td>0 0 0 0 1 1 1 1</td><td>0 0 0 0 0 1 1 0</td></tr></table>	0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 0	5	5	c	l		
0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 0								
CMC = Complement Carry Flag	<table><tr><td>1 1 1 1 0 1 0 1</td></tr></table>	1 1 1 1 0 1 0 1	2	2					
1 1 1 1 0 1 0 1									
LAHF = Load AH into Flag	<table><tr><td>1 0 0 1 1 1 1 1</td></tr></table>	1 0 0 1 1 1 1 1	2	2					
1 0 0 1 1 1 1 1									
POPF = Pop Flags	<table><tr><td>1 0 0 1 1 1 0 1</td></tr></table>	1 0 0 1 1 1 0 1	5	5	b	h, n			
1 0 0 1 1 1 0 1									
PUSHF = Push Flags	<table><tr><td>1 0 0 1 1 1 0 0</td></tr></table>	1 0 0 1 1 1 0 0	4	4	b	h			
1 0 0 1 1 1 0 0									
SAHF = Store AH into Flags	<table><tr><td>1 0 0 1 1 1 1 0</td></tr></table>	1 0 0 1 1 1 1 0	3	3					
1 0 0 1 1 1 1 0									
STC = Set Carry Flag	<table><tr><td>1 1 1 1 1 0 0 1</td></tr></table>	1 1 1 1 1 0 0 1	2	2					
1 1 1 1 1 0 0 1									
STD = Set Direction Flag	<table><tr><td>1 1 1 1 1 1 0 1</td></tr></table>	1 1 1 1 1 1 0 1							
1 1 1 1 1 1 0 1									
STI = Set Interrupt Enable Flag	<table><tr><td>1 1 1 1 1 0 1 1</td></tr></table>	1 1 1 1 1 0 1 1	8	8		m			
1 1 1 1 1 0 1 1									
ARITHMETIC									
ADD = Add									
Register to Register	<table><tr><td>0 0 0 0 0 0 d w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 0 0 d w	mod reg	r/m	2	2			
0 0 0 0 0 0 d w	mod reg	r/m							
Register to Memory	<table><tr><td>0 0 0 0 0 0 0 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 0 0 0 w	mod reg	r/m	7**	7**	b	h	
0 0 0 0 0 0 0 w	mod reg	r/m							
Memory to Register	<table><tr><td>0 0 0 0 0 0 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 0 0 0 1 w	mod reg	r/m	6*	6*	b	h	
0 0 0 0 0 0 1 w	mod reg	r/m							
Immediate to Register/Memory	<table><tr><td>1 0 0 0 0 0 s w</td><td>mod 0 0 0</td><td>r/m</td><td>immediate data</td></tr></table>	1 0 0 0 0 0 s w	mod 0 0 0	r/m	immediate data	2/7**	2/7**	b	h
1 0 0 0 0 0 s w	mod 0 0 0	r/m	immediate data						
Immediate to Accumulator (short form)	<table><tr><td>0 0 0 0 0 1 0 w</td><td colspan="2">immediate data</td></tr></table>	0 0 0 0 0 1 0 w	immediate data		2	2			
0 0 0 0 0 1 0 w	immediate data								
ADC = Add With Carry									
Register to Register	<table><tr><td>0 0 0 1 0 0 d w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 1 0 0 d w	mod reg	r/m	2	2			
0 0 0 1 0 0 d w	mod reg	r/m							
Register to Memory	<table><tr><td>0 0 0 1 0 0 0 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 1 0 0 0 w	mod reg	r/m	7**	7**	b	h	
0 0 0 1 0 0 0 w	mod reg	r/m							
Memory to Register	<table><tr><td>0 0 0 1 0 0 1 w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 0 1 0 0 1 w	mod reg	r/m	6*	6*	b	h	
0 0 0 1 0 0 1 w	mod reg	r/m							
Immediate to Register/Memory	<table><tr><td>1 0 0 0 0 0 s w</td><td>mod 0 1 0</td><td>r/m</td><td>immediate data</td></tr></table>	1 0 0 0 0 0 s w	mod 0 1 0	r/m	immediate data	2/7**	2/7**	b	h
1 0 0 0 0 0 s w	mod 0 1 0	r/m	immediate data						
Immediate to Accumulator (short form)	<table><tr><td>0 0 0 1 0 1 0 w</td><td colspan="2">immediate data</td></tr></table>	0 0 0 1 0 1 0 w	immediate data		2	2			
0 0 0 1 0 1 0 w	immediate data								
INC = Increment									
Register/Memory	<table><tr><td>1 1 1 1 1 1 1 w</td><td>mod 0 0 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 w	mod 0 0 0	r/m	2/6**	2/6**	b	h	
1 1 1 1 1 1 1 w	mod 0 0 0	r/m							
Register (short form)	<table><tr><td>0 1 0 0 0</td><td>reg</td></tr></table>	0 1 0 0 0	reg	2	2				
0 1 0 0 0	reg								
SUB = Subtract									
Register from Register	<table><tr><td>0 0 1 0 1 0 d w</td><td>mod reg</td><td>r/m</td></tr></table>	0 0 1 0 1 0 d w	mod reg	r/m	2	2			
0 0 1 0 1 0 d w	mod reg	r/m							

Table 9-1. 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
ARITHMETIC (Continued)						
Register from Memory	0 0 1 0 1 0 0 w	mod reg r/m	7**	7**	b	h
Memory from Register	0 0 1 0 1 0 1 w	mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	immediate data	2/7**	b	h
Immediate from Accumulator (short form)	0 0 1 0 1 1 0 w	immediate data	2	2		
SBB = Subtract with Borrow						
Register from Register	0 0 0 1 1 0 d w	mod reg r/m	2	2		
Register from Memory	0 0 0 1 1 0 0 w	mod reg r/m	7**	7**	b	h
Memory from Register	0 0 0 1 1 0 1 w	mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	immediate data	2/7**	b	h
Immediate from Accumulator (short form)	0 0 0 1 1 1 0 w	immediate data	2	2		
DEC = Decrement						
Register/Memory	1 1 1 1 1 1 1 w	reg 0 0 1 r/m	2/6	2/6	b	h
Register (short form)	0 1 0 0 1 reg		2	2		
CMP = Compare						
Register with Register	0 0 1 1 1 0 d w	mod reg r/m	2	2		
Memory with Register	0 0 1 1 1 0 0 w	mod reg r/m	5*	5*	b	h
Register with Memory	0 0 1 1 1 0 1 w	mod reg r/m	6*	6*	b	h
Immediate with Register/Memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	immediate data	2/5*	b	h
Immediate with Accumulator (short form)	0 0 1 1 1 1 0 w	immediate data	2	2		
NEG = Change Sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m	2/6*	2/6*	b	h
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1		4	4		
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1		4	4		
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1		4	4		
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1		4	4		
MUL = Multiply (unsigned)						
Accumulator with Register/Memory	1 1 1 1 0 1 1 w	mod 1 0 0 r/m				
Multiplier-Byte			12-17/15-20*	12-17/15-20*	b, d	d, h
-Word			12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword			12-41/17-46*	12-41/17-46*	b, d	d, h
IMUL = Integer Multiply (signed)						
Accumulator with Register/Memory	1 1 1 1 0 1 1 w	mod 1 0 1 r/m				
Multiplier-Byte			12-17/15-20*	12-17/15-20*	b, d	d, h
-Word			12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword			12-41/17-46*	12-41/17-46*	b, d	d, h
Register with Register/Memory	0 0 0 0 1 1 1 1	1 0 1 0 1 1 1 1	mod reg r/m			
Multiplier-Byte			12-17/15-20*	12-17/15-20*	b, d	d, h
-Word			12-25/15-28*	12-25/15-28*	b, d	d, h
-Doubleword			12-41/17-46*	12-41/17-46*	b, d	d, h
Register/Memory with Immediate to Register	0 1 1 0 1 0 s 1	mod reg r/m	immediate data			
-Word			13-26	13-26/14-27	b, d	d, h
-Doubleword			13-42	13-42/16-45	b, d	d, h

Table 9-1. 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			
ARITHMETIC (Continued)								
DIV = Divide (Unsigned)								
Accumulator by Register/Memory	<table><tr><td>1 1 1 1 0 1 1 w</td><td>mod 1 1 0 r/m</td></tr></table>	1 1 1 1 0 1 1 w	mod 1 1 0 r/m					
1 1 1 1 0 1 1 w	mod 1 1 0 r/m							
Divisor—Byte		14/17	14/17	b,e	e,h			
—Word		22/25	22/25	b,e	e,h			
—Doubleword		38/43	38/43	b,e	e,h			
IDIV = Integer Divide (Signed)								
Accumulator By Register/Memory	<table><tr><td>1 1 1 1 0 1 1 w</td><td>mod 1 1 1 r/m</td></tr></table>	1 1 1 1 0 1 1 w	mod 1 1 1 r/m					
1 1 1 1 0 1 1 w	mod 1 1 1 r/m							
Divisor—Byte		19/22	19/22	b,e	e,h			
—Word		27/30	27/30	b,e	e,h			
—Doubleword		43/48	43/48	b,e	e,h			
AAD = ASCII Adjust for Divide	<table><tr><td>1 1 0 1 0 1 0 1</td><td>0 0 0 0 1 0 1 0</td></tr></table>	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	19	19			
1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0							
AAM = ASCII Adjust for Multiply	<table><tr><td>1 1 0 1 0 1 0 0</td><td>0 0 0 0 1 0 1 0</td></tr></table>	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	17	17			
1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0							
CBW = Convert Byte to Word	<table><tr><td>1 0 0 1 1 0 0 0</td></tr></table>	1 0 0 1 1 0 0 0	3	3				
1 0 0 1 1 0 0 0								
CWD = Convert Word to Double Word	<table><tr><td>1 0 0 1 1 0 0 1</td></tr></table>	1 0 0 1 1 0 0 1	2	2				
1 0 0 1 1 0 0 1								
LOGIC								
Shift Rotate Instructions								
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)								
Register/Memory by 1	<table><tr><td>1 1 0 1 0 0 0 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 1 0 0 0 w	mod TTT r/m	3/7**	3/7**	b	h	
1 1 0 1 0 0 0 w	mod TTT r/m							
Register/Memory by CL	<table><tr><td>1 1 0 1 0 0 1 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 1 0 0 1 w	mod TTT r/m	3/7*	3/7*	b	h	
1 1 0 1 0 0 1 w	mod TTT r/m							
Register/Memory by Immediate Count	<table><tr><td>1 1 0 0 0 0 0 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 0 0 0 0 w	mod TTT r/m	3/7*	3/7*	b	h	
1 1 0 0 0 0 0 w	mod TTT r/m							
immed 8-bit data								
Through Carry (RCL and RCR)								
Register/Memory by 1	<table><tr><td>1 1 0 1 0 0 0 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 1 0 0 0 w	mod TTT r/m	9/10*	9/10*	b	h	
1 1 0 1 0 0 0 w	mod TTT r/m							
Register/Memory by CL	<table><tr><td>1 1 0 1 0 0 1 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 1 0 0 1 w	mod TTT r/m	9/10*	9/10*	b	h	
1 1 0 1 0 0 1 w	mod TTT r/m							
Register/Memory by Immediate Count	<table><tr><td>1 1 0 0 0 0 0 w</td><td>mod TTT r/m</td></tr></table>	1 1 0 0 0 0 0 w	mod TTT r/m	9/10*	9/10*	b	h	
1 1 0 0 0 0 0 w	mod TTT r/m							
immed 8-bit data								
TTT Instruction								
0 0 0 ROL								
0 0 1 ROR								
0 1 0 RCL								
0 1 1 RCR								
1 0 0 SHL/SAL								
1 0 1 SHR								
1 1 1 SAR								
SHLD = Shift Left Double								
Register/Memory by Immediate	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 0 0 1 0 0</td><td>mod reg r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 0	mod reg r/m	3/7**	3/7**		
0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 0	mod reg r/m						
immed 8-bit data								
Register/Memory by CL	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 0 0 1 0 1</td><td>mod reg r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 1	mod reg r/m	3/7**	3/7**		
0 0 0 0 1 1 1 1	1 0 1 0 0 1 0 1	mod reg r/m						
SHRD = Shift Right Double								
Register/Memory by Immediate	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 0 1 1 0 0</td><td>mod reg r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 0	mod reg r/m	3/7**	3/7**		
0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 0	mod reg r/m						
immed 8-bit data								
Register/Memory by CL	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 1 0 1 1 0 1</td><td>mod reg r/m</td></tr></table>	0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 1	mod reg r/m	3/7**	3/7**		
0 0 0 0 1 1 1 1	1 0 1 0 1 1 0 1	mod reg r/m						
AND = And								
Register to Register	<table><tr><td>0 0 1 0 0 0 d w</td><td>mod reg r/m</td></tr></table>	0 0 1 0 0 0 d w	mod reg r/m	2	2			
0 0 1 0 0 0 d w	mod reg r/m							

Table 9-1. 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
LOGIC (Continued)			Cik Count Virtual 8086 Mode			
Register to Memory	0 0 1 0 0 0 0 w	mod reg r/m		7**	7**	b h
Memory to Register	0 0 1 0 0 0 1 w	mod reg r/m		6*	6*	b h
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m		2/7*	2/7**	b h
Immediate to Accumulator (Short Form)	0 0 1 0 0 1 0 w	immediate data		2	2	
TEST = And Function to Flags, No Result						
Register/Memory and Register	1 0 0 0 0 1 0 w	mod reg r/m		2/5*	2/5*	b h
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m		2/5*	2/5*	b h
Immediate Data and Accumulator (Short Form)	1 0 1 0 1 0 0 w	immediate data		2	2	
OR = Or						
Register to Register	0 0 0 0 1 0 d w	mod reg r/m		2	2	
Register to Memory	0 0 0 0 1 0 0 w	mod reg r/m		7**	7**	b h
Memory to Register	0 0 0 0 1 0 1 w	mod reg r/m		6*	6*	b h
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m		2/7**	2/7**	b h
Immediate to Accumulator (Short Form)	0 0 0 0 1 1 0 w	immediate data		2	2	
XOR = Exclusive Or						
Register to Register	0 0 1 1 0 0 d w	mod reg r/m		2	2	
Register to Memory	0 0 1 1 0 0 0 w	mod reg r/m		7**	7**	b h
Memory to Register	0 0 1 1 0 0 1 w	mod reg r/m		6*	6*	b h
Immediate to Register/Memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m		2/7**	2/7**	b h
Immediate to Accumulator (Short Form)	0 0 1 1 0 1 0 w	immediate data	2	2		
NOT = Invert Register/Memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m	2/6**	2/6**	b h	
STRING MANIPULATION						
CMPS = Compare Byte Word	1 0 1 0 0 1 1 w		10*	10*	b h	
INS = Input Byte/Word from DX Port	0 1 1 0 1 1 0 w	†29	15	9*/29**	b s/t, h, m	
LODS = Load Byte/Word to AL/AX/EAX	1 0 1 0 1 1 0 w		5	5*	b h	
MOVS = Move Byte Word	1 0 1 0 0 1 0 w		7	7**	b h	
OUTS = Output Byte/Word to DX Port	0 1 1 0 1 1 1 w	†28	14	8*/28*	b s/t, h, m	
SCAS = Scan Byte Word	1 0 1 0 1 1 1 w		7*	7*	b h	
STOS = Store Byte/Word from AL/AX/EX	1 0 1 0 1 0 1 w		4*	4*	b h	
XLAT = Translate String	1 1 0 1 0 1 1 1		5*	5*	h	
REPEATED STRING MANIPULATION						
Repeated by Count in CX or ECX						
REPE CMPS = Compare String						
(Find Non-Match)	1 1 1 1 0 0 1 1	1 0 1 0 0 1 1 w	5 + 9n**	5 + 9n**	b h	

Clk  
Count  
Virtual  
8086  
Mode

Table 9-1. 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
REPEATED STRING MANIPULATION (Continued)						
REPNE CMPS = Compare String (Find Match)	1 1 1 1 0 0 1 0    1 0 1 0 0 1 1 w	Clk Count Virtual 8086 Mode	5 + 9n**	5 + 9n**	b	h
REP INS = Input String	1 1 1 1 0 0 1 0    0 1 1 0 1 1 0 w	†	13 + 6n*	7 + 6n* / 27 + 6n*	b	s/t, h, m
REP LODS = Load String	1 1 1 1 0 0 1 0    1 0 1 0 1 1 0 w		5 + 6n*	5 + 6n*	b	h
REP MOVS = Move String	1 1 1 1 0 0 1 0    1 0 1 0 0 1 0 w		7 + 4n*	7 + 4n**	b	h
REP OUTS = Output String	1 1 1 1 0 0 1 0    0 1 1 0 1 1 1 w	†	12 + 5n*	6 + 5n* / 26 + 5n*	b	s/t, h, m
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	1 1 1 1 0 0 1 1    1 0 1 0 1 1 1 w		5 + 8n*	5 + 8n*	b	h
REPNE SCAS = Scan String (Find AL/AX/EAX)	1 1 1 1 0 0 1 0    1 0 1 0 1 1 1 w		5 + 8n*	5 + 8n*	b	h
REP STOS = Store String	1 1 1 1 0 0 1 0    1 0 1 0 1 0 1 w		5 + 5n*	5 + 5n*	b	h
BIT MANIPULATION						
BSF = Scan Bit Forward	0 0 0 0 1 1 1 1    1 0 1 1 1 1 0 0 mod reg    r/m		10 + 3n*	10 + 3n**	b	h
BSR = Scan Bit Reverse	0 0 0 0 1 1 1 1    1 0 1 1 1 1 0 1 mod reg    r/m		10 + 3n*	10 + 3n**	b	h
BT = Test Bit						
Register/Memory, Immediate	0 0 0 0 1 1 1 1    1 0 1 1 1 0 1 0 mod 1 0 0    r/m	immed 8-bit data	3/6*	3/6*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1    1 0 1 0 0 0 1 1 mod reg    r/m		3/12*	3/12*	b	h
BTC = Test Bit and Complement						
Register/Memory, Immediate	0 0 0 0 1 1 1 1    1 0 1 1 1 0 1 0 mod 1 1 1    r/m	immed 8-bit data	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1    1 0 1 1 1 0 1 1 mod reg    r/m		6/13*	6/13*	b	h
BTR = Test Bit and Reset						
Register/Memory, Immediate	0 0 0 0 1 1 1 1    1 0 1 1 1 0 1 0 mod 1 1 0    r/m	immed 8-bit data	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1    1 0 1 1 0 0 1 1 mod reg    r/m		6/13*	6/13*	b	h
BTS = Test Bit and Set						
Register/Memory, Immediate	0 0 0 0 1 1 1 1    1 0 1 1 1 0 1 0 mod 1 0 1    r/m	immed 8-bit data	6/8*	6/8*	b	h
Register/Memory, Register	0 0 0 0 1 1 1 1    1 0 1 0 1 0 1 1 mod reg    r/m		6/13*	6/13*	b	h
CONTROL TRANSFER						
CALL = Call						
Direct Within Segment	1 1 1 0 1 0 0 0	full displacement	7 + m*	9 + m*	b	r
Register/Memory						
Indirect Within Segment	1 1 1 1 1 1 1 1    mod 0 1 0    r/m		7 + m* / 10 + m*	9 + m / 12 + m*	b	h, r
Direct Intersegment	1 0 0 1 1 0 1 0	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 9-1. 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 Intel386™ SX CPU TSS				310		h,j,k,r		
From 286 Task to Virtual 8086 Task (Intel386 SX CPU TSS)				229		h,j,k,r		
From Intel386 SX CPU Task to 286 TSS				285		h,j,k,r		
From Intel386 SX CPU Task to Intel386 SX CPU TSS				392		h,j,k,r		
From Intel386 SX CPU Task to Virtual 8086 Task (Intel386 SX CPU TSS)				309		h,j,k,r		
Indirect Intersegment	<table><tr><td>1 1 1 1 1 1 1 1</td><td>mod 0 1 1</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 0 1 1	r/m	30 + m	46 + m	b	h,j,k,r
1 1 1 1 1 1 1 1	mod 0 1 1	r/m						
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 Intel386 SX CPU TSS						h,j,k,r		
From 286 Task to Virtual 8086 Task (Intel386 SX CPU TSS)						h,j,k,r		
From Intel386 SX CPU Task to 286 TSS						h,j,k,r		
From Intel386 SX CPU Task to Intel386 SX CPU TSS				399		h,j,k,r		
From Intel386 SX CPU Task to Virtual 8086 Task (Intel386 SX CPU TSS)						h,j,k,r		
JMP = Unconditional Jump								
Short	<table><tr><td>1 1 1 0 1 0 1 1</td><td>8-bit displacement</td></tr></table>	1 1 1 0 1 0 1 1	8-bit displacement	7 + m	7 + m		r	
1 1 1 0 1 0 1 1	8-bit displacement							
Direct within Segment	<table><tr><td>1 1 1 0 1 0 0 1</td><td>full displacement</td></tr></table>	1 1 1 0 1 0 0 1	full displacement	7 + m	7 + m		r	
1 1 1 0 1 0 0 1	full displacement							
Register/Memory Indirect within Segment	<table><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 0</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 0	r/m	9 + m/14 + m	9 + m/14 + m	b	h,r
1 1 1 1 1 1 1 1	mod 1 0 0	r/m						
Direct Intersegment	<table><tr><td>1 1 1 0 1 0 1 0</td><td>unsigned full offset, selector</td></tr></table>	1 1 1 0 1 0 1 0	unsigned full offset, selector	16 + m	31 + m		j,k,r	
1 1 1 0 1 0 1 0	unsigned full offset, selector							
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 Intel386 SX CPU TSS						h,j,k,r		
From 286 Task to Virtual 8086 Task (Intel386 SX CPU TSS)						h,j,k,r		
From Intel386 SX CPU Task to 286 TSS						h,j,k,r		
From Intel386 SX CPU Task to Intel386 SX CPU TSS						h,j,k,r		
From Intel386 SX CPU Task to Virtual 8086 Task (Intel386 SX CPU TSS)				395		h,j,k,r		
Indirect Intersegment	<table><tr><td>1 1 1 1 1 1 1 1</td><td>mod 1 0 1</td><td>r/m</td></tr></table>	1 1 1 1 1 1 1 1	mod 1 0 1	r/m	17 + m	31 + m	b	h,j,k,r
1 1 1 1 1 1 1 1	mod 1 0 1	r/m						
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 Intel386 SX CPU TSS						h,j,k,r		
From 286 Task to Virtual 8086 Task (Intel386 SX CPU TSS)						h,j,k,r		
From Intel386 SX CPU Task to 286 TSS						h,j,k,r		
From Intel386 SX CPU Task to Intel386 SX CPU TSS						h,j,k,r		
From Intel386 SX CPU Task to Virtual 8086 Task (Intel386 SX CPU TSS)				328		h,j,k,r		

Table 9-1. 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)								
RET = Return from CALL:								
Within Segment	<table><tr><td>1 1 0 0 0 1 1</td></tr></table>	1 1 0 0 0 1 1		12 + m	b	g, h, r		
1 1 0 0 0 1 1								
Within Segment Adding Immediate to SP	<table><tr><td>1 1 0 0 0 1 0</td><td>16-bit displ</td></tr></table>	1 1 0 0 0 1 0	16-bit displ		12 + m	b	g, h, r	
1 1 0 0 0 1 0	16-bit displ							
Intersegment	<table><tr><td>1 1 0 0 1 0 1 1</td></tr></table>	1 1 0 0 1 0 1 1		36 + m	b	g, h, j, k, r		
1 1 0 0 1 0 1 1								
Intersegment Adding Immediate to SP	<table><tr><td>1 1 0 0 1 0 1 0</td><td>16-bit displ</td></tr></table>	1 1 0 0 1 0 1 0	16-bit displ		36 + m	b	g, h, j, k, r	
1 1 0 0 1 0 1 0	16-bit displ							
Protected Mode Only (RET):								
to Different Privilege Level								
Intersegment								
Intersegment Adding Immediate to SP								
72								
72								
CONDITIONAL JUMPS								
NOTE: Times Are Jump "Taken or Not Taken"								
JO = Jump on Overflow								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 0 0 0</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 0 0 0	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 0 0 0	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 0 0 0</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 0	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 0	full displacement						
JNO = Jump on Not Overflow								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 0 0 1</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 0 0 1	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 0 0 1	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 0 0 1</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 1	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 0 0 1	full displacement						
JB/JNAE = Jump on Below/Not Above or Equal								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 0 1 0</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 0 1 0	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 0 1 0	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 0 1 0</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 0	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 0	full displacement						
JNB/JAE = Jump on Not Below/Above or Equal								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 0 1 1</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 0 1 1	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 0 1 1	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 0 1 1</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 1	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 0 1 1	full displacement						
JE/JZ = Jump on Equal/Zero								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 1 0 0</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 1 0 0	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 1 0 0	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 1 0 0</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 0	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 0	full displacement						
JNE/JNZ = Jump on Not Equal/Not Zero								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 1 0 1</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 1 0 1	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 1 0 1	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 1 0 1</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 1	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 1 0 1	full displacement						
JBE/JNA = Jump on Below or Equal/Not Above								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 1 1 0</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 1 1 0	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 1 1 0	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 1 1 0</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 0	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 0	full displacement						
JNBE/JA = Jump on Not Below or Equal/Above								
8-Bit Displacement	<table><tr><td>0 1 1 1 0 1 1 1</td><td>8-bit displ</td></tr></table>	0 1 1 1 0 1 1 1	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 0 1 1 1	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 0 1 1 1</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 1	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 0 1 1 1	full displacement						
JS = Jump on Sign								
8-Bit Displacement	<table><tr><td>0 1 1 1 1 0 0 0</td><td>8-bit displ</td></tr></table>	0 1 1 1 1 0 0 0	8-bit displ	7 + m or 3	7 + m or 3		r	
0 1 1 1 1 0 0 0	8-bit displ							
Full Displacement	<table><tr><td>0 0 0 0 1 1 1 1</td><td>1 0 0 0 1 0 0 0</td><td>full displacement</td></tr></table>	0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 0	full displacement	7 + m or 3	7 + m or 3		r
0 0 0 0 1 1 1 1	1 0 0 0 1 0 0 0	full displacement						



Table 9-1. Instruction Set Clock Count Summary (Continued)

		CLOCK COUNT		NOTES	
INSTRUCTION	FORMAT	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	0 1 1 1 1 0 0 1    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 0 0 1    full displacement	7 + m or 3	7 + m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	0 1 1 1 1 0 1 0    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 0 1 0    full displacement	7 + m or 3	7 + m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	0 1 1 1 1 0 1 1    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 0 1 1    full displacement	7 + m or 3	7 + m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	0 1 1 1 1 1 0 0    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 1 0 0    full displacement	7 + m or 3	7 + m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	0 1 1 1 1 1 0 1    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 1 0 1    full displacement	7 + m or 3	7 + m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	0 1 1 1 1 1 1 0    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 1 1 0    full displacement	7 + m or 3	7 + m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	0 1 1 1 1 1 1 1    8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	0 0 0 0 1 1 1 1    1 0 0 0 1 1 1 1    full displacement	7 + m or 3	7 + m or 3		r
JCXZ = Jump on CX Zero					
	1 1 1 0 0 0 1 1    8-bit displ	9 + m or 5	9 + m or 5		r
JECXZ = Jump on ECX Zero					
	1 1 1 0 0 0 1 1    8-bit displ	9 + m or 5	9 + m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times	1 1 1 0 0 0 1 0    8-bit displ	11 + m	11 + m		r
LOOPZ/LOOPE = Loop with Zero/Equal	1 1 1 0 0 0 0 1    8-bit displ	11 + m	11 + m		r
LOOPNZ/LOOPNE = Loop While Not Zero	1 1 1 0 0 0 0 0    8-bit displ	11 + m	11 + m		r
CONDITIONAL BYTE SET					
NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	0 0 0 0 1 1 1 1    1 0 0 1 0 0 0 0    mod 0 0 0    r/m	4/5*	4/5*		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	0 0 0 0 1 1 1 1    1 0 0 1 0 0 0 1    mod 0 0 0    r/m	4/5*	4/5*		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	0 0 0 0 1 1 1 1    1 0 0 1 0 0 1 0    mod 0 0 0    r/m	4/5*	4/5*		h

Table 9-1. 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				
CONDITIONAL BYTE SET (Continued)									
SETNB = Set Byte on Not Below/Above or Equal									
To Register/Memory	<table><tr><td>00001111</td><td>10010011</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10010011	mod 000	r/m	4/5*	4/5*		h
00001111	10010011	mod 000	r/m						
SETE/SETZ = Set Byte on Equal/Zero									
To Register/Memory	<table><tr><td>00001111</td><td>10010100</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10010100	mod 000	r/m	4/5*	4/5*		h
00001111	10010100	mod 000	r/m						
SETNE/SETNZ = Set Byte on Not Equal/Not Zero									
To Register/Memory	<table><tr><td>00001111</td><td>10010101</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10010101	mod 000	r/m	4/5*	4/5*		h
00001111	10010101	mod 000	r/m						
SETBE/SETNA = Set Byte on Below or Equal/Not Above									
To Register/Memory	<table><tr><td>00001111</td><td>10010110</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10010110	mod 000	r/m	4/5*	4/5*		h
00001111	10010110	mod 000	r/m						
SETNBE/SETA = Set Byte on Not Below or Equal/Above									
To Register/Memory	<table><tr><td>00001111</td><td>10010111</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10010111	mod 000	r/m	4/5*	4/5*		h
00001111	10010111	mod 000	r/m						
SETS = Set Byte on Sign									
To Register/Memory	<table><tr><td>00001111</td><td>10011000</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011000	mod 000	r/m	4/5*	4/5*		h
00001111	10011000	mod 000	r/m						
SETNS = Set Byte on Not Sign									
To Register/Memory	<table><tr><td>00001111</td><td>10011001</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011001	mod 000	r/m	4/5*	4/5*		h
00001111	10011001	mod 000	r/m						
SETP/SETPE = Set Byte on Parity/Parity Even									
To Register/Memory	<table><tr><td>00001111</td><td>10011010</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011010	mod 000	r/m	4/5*	4/5*		h
00001111	10011010	mod 000	r/m						
SETNP/SETPO = Set Byte on Not Parity/Parity Odd									
To Register/Memory	<table><tr><td>00001111</td><td>10011011</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011011	mod 000	r/m	4/5*	4/5*		h
00001111	10011011	mod 000	r/m						
SETL/SETNGE = Set Byte on Less/Not Greater or Equal									
To Register/Memory	<table><tr><td>00001111</td><td>10011100</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011100	mod 000	r/m	4/5*	4/5*		h
00001111	10011100	mod 000	r/m						
SETNL/SETGE = Set Byte on Not Less/Greater or Equal									
To Register/Memory	<table><tr><td>00001111</td><td>01111101</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	01111101	mod 000	r/m	4/5*	4/5*		h
00001111	01111101	mod 000	r/m						
SETLE/SETNG = Set Byte on Less or Equal/Not Greater									
To Register/Memory	<table><tr><td>00001111</td><td>10011110</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011110	mod 000	r/m	4/5*	4/5*		h
00001111	10011110	mod 000	r/m						
SETNLE/SETG = Set Byte on Not Less or Equal/Greater									
To Register/Memory	<table><tr><td>00001111</td><td>10011111</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	10011111	mod 000	r/m	4/5*	4/5*		h
00001111	10011111	mod 000	r/m						
ENTER = Enter Procedure		<table><tr><td>11001000</td><td>16-bit displacement, 8-bit level</td></tr></table>	11001000	16-bit displacement, 8-bit level					
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		<table><tr><td>11001001</td></tr></table>	11001001	4	4	b	h		
11001001									

Table 9-1. 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		
INTERRUPT INSTRUCTIONS							
INT = Interrupt:							
Type Specified	<table><tr><td>1 1 0 0 1 1 0 1</td><td>type</td></tr></table>	1 1 0 0 1 1 0 1	type	37		b	
1 1 0 0 1 1 0 1	type						
Type 3	<table><tr><td>1 1 0 0 1 1 0 0</td></tr></table>	1 1 0 0 1 1 0 0	33		b		
1 1 0 0 1 1 0 0							
INTO = Interrupt 4 if Overflow Flag Set							
	<table><tr><td>1 1 0 0 1 1 1 0</td></tr></table>	1 1 0 0 1 1 1 0					
1 1 0 0 1 1 1 0							
If OF = 1		35		b, e			
If OF = 0		3	3	b, e			
Bound = Interrupt 5 if Detect Value Out of Range							
	<table><tr><td>0 1 1 0 0 0 1 0</td><td>mod reg</td><td>r/m</td></tr></table>	0 1 1 0 0 0 1 0	mod reg	r/m			
0 1 1 0 0 0 1 0	mod reg	r/m					
If Out of Range		44		b, e	e, g, h, j, k, r		
If In Range		10	10	b, e	e, g, h, j, k, r		
Protected Mode Only (INT)							
INT: Type Specified							
Via Interrupt or Trap Gate							
Via Interrupt or Trap Gate							
to Same Privilege Level							
to Different Privilege Level							
From 286 Task to 286 TSS via Task Gate							
From 286 Task to Intel386™ SX CPU TSS via Task Gate							
From 286 Task to virt 8086 md via Task Gate							
From Intel386™ SX CPU Task to 286 TSS via Task Gate							
From Intel386™ SX CPU Task to Intel386™ SX CPU TSS via Task Gate							
From Intel386™ SX CPU Task to virt 8086 md via Task Gate							
From virt 8086 md to 286 TSS via Task Gate							
From virt 8086 md to Intel386™ SX CPU TSS via Task Gate							
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate							
INT: TYPE 3							
Via Interrupt or Trap Gate							
to Same Privilege Level							
Via Interrupt or Trap Gate							
to Different Privilege Level							
From 286 Task to 286 TSS via Task Gate							
From 286 Task to Intel386™ SX CPU TSS via Task Gate							
From 286 Task to Virt 8086 md via Task Gate							
From Intel386™ SX CPU Task to 286 TSS via Task Gate							
From Intel386™ SX CPU Task to Intel386™ SX CPU TSS via Task Gate							
From Intel386™ SX CPU Task to Virt 8086 md via Task Gate							
From virt 8086 md to 286 TSS via Task Gate							
From virt 8086 md to Intel386™ SX CPU TSS via Task Gate							
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate							
INTO:							
Via Interrupt or Trap Gate							
to Same Privilege Level							
Via Interrupt or Trap Gate							
to Different Privilege Level							
From 286 Task to 286 TSS via Task Gate							
From 286 Task to Intel386™ SX CPU TSS via Task Gate							
From 286 Task to virt 8086 md via Task Gate							
From Intel386™ SX CPU Task to 286 TSS via Task Gate							
From Intel386™ SX CPU Task to Intel386™ SX CPU TSS via Task Gate							
From Intel386™ SX CPU Task to virt 8086 md via Task Gate							
From virt 8086 md to 286 TSS via Task Gate							
From virt 8086 md to Intel386™ SX CPU TSS via Task Gate							
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate							

Table 9-1. 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
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 Intel386™ SX CPU TSS via Task Gate			388		g, j, k, r
From 286 Task to virt 8086 Mode via Task Gate			335		g, j, k, r
From Intel386 SX CPU Task to 286 TSS via Task Gate			368		g, j, k, r
From Intel386 SX CPU Task to Intel386 SX CPU TSS via Task Gate			398		g, j, k, r
From Intel386 SX CPU 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 Intel386 SX CPU 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	1 1 0 0 1 1 1 1	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 Intel386 SX CPU 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 Intel386 SX CPU Task to 286 TSS			324		h, j, k, r
From Intel386 SX CPU Task to Intel386 SX CPU TSS			328		h, j, k, r
From Intel386 SX CPU Task to Virtual 8086 Task			377		h, j, k, r
From Intel386 SX CPU Task to Virtual 8086 Mode (within task)			113		
PROCESSOR CONTROL					
HLT = HALT	1 1 1 1 0 1 0 0	5	5		I
MOV = Move to and From Control/Debug/Test Registers					
CR0/CR2/CR3 from register	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 1 eee reg	10/4/5	10/4/5		I
Register From CR0-3	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 eee reg	6	6		I
DR0-3 From Register	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 eee reg	22	22		I
DR6-7 From Register	0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 1 1 1 eee reg	16	16		I
Register from DR6-7	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 eee reg	14	14		I
Register from DR0-3	0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 1 1 1 eee reg	22	22		I
TR6-7 from Register	0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 0 1 1 eee reg	12	12		I
Register from TR6-7	0 0 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 1 eee reg	12	12		I
NOP = No Operation	1 0 0 1 0 0 0 0	3	3		
WAIT = Wait until BUSY # pin is negated	1 0 0 1 1 0 1 1	6	6		

Table 9-1. Instruction Set Clock Count Summary (Continued)

		CLOCK COUNT		NOTES					
INSTRUCTION	FORMAT	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode				
PROCESSOR EXTENSION INSTRUCTIONS									
Processor Extension Escape	<table><tr><td>11011TTT</td><td>mod LLL</td><td>r/m</td></tr></table> TTT and LLL bits are opcode information for coprocessor.	11011TTT	mod LLL	r/m	See Intel387SX data sheet for clock counts			h	
11011TTT	mod LLL	r/m							
PREFIX BYTES									
Address Size Prefix	<table><tr><td>01100111</td></tr></table>	01100111	0	0					
01100111									
LOCK = Bus Lock Prefix	<table><tr><td>11110000</td></tr></table>	11110000	0	0			m		
11110000									
Operand Size Prefix	<table><tr><td>01100110</td></tr></table>	01100110	0	0					
01100110									
Segment Override Prefix									
CS:	<table><tr><td>00101110</td></tr></table>	00101110	0	0					
00101110									
DS:	<table><tr><td>00111110</td></tr></table>	00111110	0	0					
00111110									
ES:	<table><tr><td>00100110</td></tr></table>	00100110	0	0					
00100110									
FS:	<table><tr><td>01100100</td></tr></table>	01100100	0	0					
01100100									
GS:	<table><tr><td>01100101</td></tr></table>	01100101	0	0					
01100101									
SS:	<table><tr><td>00110110</td></tr></table>	00110110	0	0					
00110110									
PROTECTION CONTROL									
ARPL = Adjust Requested Privilege Level									
From Register/Memory	<table><tr><td>01100011</td><td>mod reg</td><td>r/m</td></tr></table>	01100011	mod reg	r/m	N/A	20/21**	a	h	
01100011	mod reg	r/m							
LAR = Load Access Rights									
From Register/Memory	<table><tr><td>00001111</td><td>00000010</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	00000010	mod reg	r/m	N/A	15/16*	a	g, h, j, p
00001111	00000010	mod reg	r/m						
LGDT = Load Global Descriptor									
Table Register	<table><tr><td>00001111</td><td>00000001</td><td>mod 010</td><td>r/m</td></tr></table>	00001111	00000001	mod 010	r/m	11*	11*	b, c	h, l
00001111	00000001	mod 010	r/m						
LIDT = Load Interrupt Descriptor									
Table Register	<table><tr><td>00001111</td><td>00000001</td><td>mod 011</td><td>r/m</td></tr></table>	00001111	00000001	mod 011	r/m	11*	11*	b, c	h, l
00001111	00000001	mod 011	r/m						
LLDT = Load Local Descriptor									
Table Register to Register/Memory	<table><tr><td>00001111</td><td>00000000</td><td>mod 010</td><td>r/m</td></tr></table>	00001111	00000000	mod 010	r/m	N/A	20/24*	a	g, h, j, l
00001111	00000000	mod 010	r/m						
LMSW = Load Machine Status Word									
From Register/Memory	<table><tr><td>00001111</td><td>00000001</td><td>mod 110</td><td>r/m</td></tr></table>	00001111	00000001	mod 110	r/m	10/13	10/13*	b, c	h, l
00001111	00000001	mod 110	r/m						
LSL = Load Segment Limit									
From Register/Memory	<table><tr><td>00001111</td><td>00000011</td><td>mod reg</td><td>r/m</td></tr></table>	00001111	00000011	mod reg	r/m				
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	<table><tr><td>00001111</td><td>00000000</td><td>mod 001</td><td>r/m</td></tr></table>	00001111	00000000	mod 001	r/m	N/A	23/27*	a	g, h, j, l
00001111	00000000	mod 001	r/m						
SGDT = Store Global Descriptor									
Table Register	<table><tr><td>00001111</td><td>00000001</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	00000001	mod 000	r/m	9*	9*	b, c	h
00001111	00000001	mod 000	r/m						
SIDT = Store Interrupt Descriptor									
Table Register	<table><tr><td>00001111</td><td>00000001</td><td>mod 001</td><td>r/m</td></tr></table>	00001111	00000001	mod 001	r/m	9*	9*	b, c	h
00001111	00000001	mod 001	r/m						
SLDT = Store Local Descriptor Table Register									
To Register/Memory	<table><tr><td>00001111</td><td>00000000</td><td>mod 000</td><td>r/m</td></tr></table>	00001111	00000000	mod 000	r/m	N/A	2/2*	a	h
00001111	00000000	mod 000	r/m						

Table 9-1. 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
PROTECTION CONTROL (Continued)									
SMSW	= Store Machine Status Word	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 0 0	r/m	2/2*	2/2*	b, c	h, l
STR	= Store Task Register To Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1	r/m	N/A	2/2*	a	h
VERR	= Verify Read Access Register/Memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 0	r/m	N/A	10/11*	a	g, h, j, p
VERW	= Verify Write Access	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 1	r/m	N/A	15/16*	a	g, h, j, p

**INSTRUCTION NOTES FOR TABLE 9-1****Notes a through c apply to 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 Real Address Mode and Protected Virtual Address Mode:**

- d. The Intel386 SX CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).  
Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

Actual Clock = if  $m < > 0$  then  $\max(\lceil \log_2 |m| \rceil, 3) + b$  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 s/t apply to 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.

## 9.2 INSTRUCTION ENCODING

### 9.2.1 Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the “mod r/m” byte and “scaled index” byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain

encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 9-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 9-2 is a complete list of all fields appearing in the instruction set. Further ahead, following Table 9-2, are detailed tables for each field.

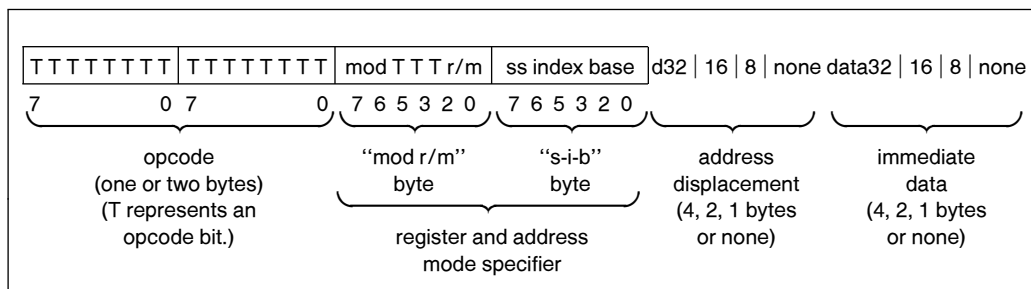


Figure 9-1. General Instruction Format

Table 9-2. Fields within Instructions

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

**Note:** Table 9-1 shows encoding of individual instructions.

### 9.2.2 32-Bit Extensions of the Instruction Set

With the Intel386 SX CPU, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the Intel386 SX CPU when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

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

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

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

### 9.2.3 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

#### 9.2.3.1 ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

#### 9.2.3.2 ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the “mod r/m” byte, or as the r/m field of the “mod r/m” byte.

##### Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
101	SI	ESI
101	DI	EDI

##### Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI



Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

### 9.2.3.3 ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the Intel386 SX CPU FS and GS segment registers to be specified.

**2-Bit sreg2 Field**

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

**3-Bit sreg3 Field**

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

### 9.2.3.4 ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the “mod r/m” byte, and a second byte of addressing information, the “s-i-b” (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the “mod r/m” byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the “mod r/m” byte, also contains three bits (shown as TTT in Figure 8-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the “mod r/m” byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the “mod r/m” byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.



Encoding of 16-bit Address Mode with “mod r/m” Byte

mod r/m	Effective Address	mod r/m	Effective Address
00 000	DS:[BX + SI]	10 000	DS:[BX + SI + d16]
00 001	DS:[BX + DI]	10 001	DS:[BX + DI + d16]
00 010	SS:[BP + SI]	10 010	SS:[BP + SI + d16]
00 011	SS:[BP + DI]	10 011	SS:[BP + DI + d16]
00 100	DS:[SI]	10 100	DS:[SI + d16]
00 101	DS:[DI]	10 101	DS:[DI + d16]
00 110	DS:d16	10 110	SS:[BP + d16]
00 111	DS:[BX]	10 111	DS:[BX + d16]
01 000	DS:[BX + SI + d8]	11 000	register—see below
01 001	DS:[BX + DI + d8]	11 001	register—see below
01 010	SS:[BP + SI + d8]	11 010	register—see below
01 011	SS:[BP + DI + d8]	11 011	register—see below
01 100	DS:[SI + d8]	11 100	register—see below
01 101	DS:[DI + d8]	11 101	register—see below
01 110	SS:[BP + d8]	11 110	register—see below
01 111	DS:[BX + d8]	11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI



**Encoding of 32-bit Address Mode with “mod r/m” byte (no “s-i-b” byte present):**

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

**Register Specified by reg or r/m during 16-Bit Data Operations:**

mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

**Register Specified by reg or r/m during 32-Bit Data Operations:**

mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI



Encoding of 32-bit Address Mode (“mod r/m” byte and “s-i-b” byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

**\*\*IMPORTANT NOTE:**  
When index field is 100, indicating “no index register,” then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

**NOTE:**  
Mod field in “mod r/m” byte; ss, index, base fields in “s-i-b” byte.



### 9.2.3.5 ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory < - - Register “reg” Field Indicates Source Operand; “mod r/m” or “mod ss index base” Indicates Destination Operand
1	Register < - - Register/Memory “reg” Field Indicates Destination Operand; “mod r/m” or “mod ss index base” Indicates Source Operand

### 9.2.3.6 ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16 32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

### 9.2.3.7 ENCODING OF CONDITIONAL TEST (tttn) FIELD

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

### 9.2.3.8 ENCODING OF CONTROL OR DEBUG OR TEST REGISTER (eee) FIELD

For the loading and storing of the Control, Debug and Test registers.

#### When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

#### When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

#### When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	



## DATA SHEET REVISION REVIEW

The following list represents key differences between this data sheet and the -007 version of the Intel386™ SX microprocessor data sheet. Please review the summary carefully.

1. Table 5.7, E-Step revision identifier is added.
2. Table 7.3,  $I_{CC}$  supply current for CLK2 = 40 MHz with 20 MHz Intel386 SX has a typical  $I_{CC}$  of 180 mA.
3. Table 7.5,  $t_4$  CLK2 fall time and  $t_5$  CLK2 rise time have no minimum time for all speeds but maximum time for all speeds is 8 ns.
4. Figure 7.11, CHMOS III characteristics for typical  $I_{CC}$  has been taken out.



This datasheet has been download from:

[www.datasheetcatalog.com](http://www.datasheetcatalog.com)

Datasheets for electronics components.