intel®

ADVANCE INFORMATION

# 80386 HIGH PERFORMANCE MICROPROCESSOR WITH INTEGRATED MEMORY MANAGEMENT

- **Flexible 32-Bit Microprocessor**
  - **8, 16, 32-Bit Data Types**
  - **8 General Purpose 32-Bit Registers**
- **Very Large Address Space**
  - **4 Gigabyte Physical**
  - **64 Terabyte Virtual**
  - **4 Gigabyte Maximum Segment Size**
- **Integrated Memory Management Unit**
  - **Virtual Memory Support**
  - **Optional On-Chip Paging**
  - **4 Levels of Protection**
  - **Fully Compatible with 80286**
- **Object Code Compatible with All 8086 Family Microprocessors**
- **Virtual 8086 Mode Allows Running of 8086 Software in a Protected and Paged System**
- **Hardware Debugging Support**
- **Optimized for System Performance**
  - **Pipelined Instruction Execution**
  - **On-Chip Address Translation Caches**
  - **12.5 and 16 MHz Clock**
  - **32 Megabytes/Sec Bus Bandwidth**
- **High Speed Numerics Support via 80287 and 80387 Coprocessors**
- **Complete System Development Support**
  - **Software: C, PL/M, Assembler System Generation Tools**
  - **Debuggers: PSCOPE, ICE™-386**
- **High Speed CHMOS III Technology**
- **132 Pin Grid Array Package**

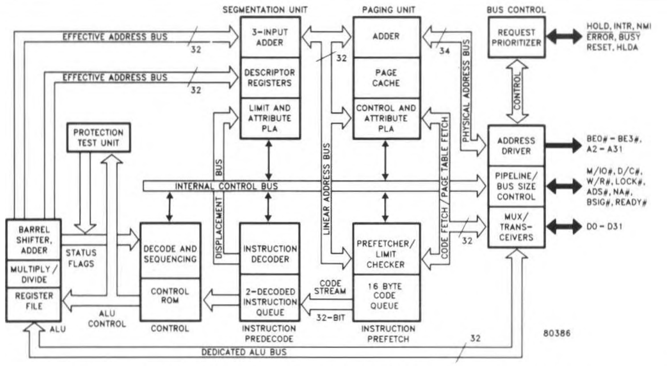
(See Packaging Specification, Order #231369)

The 80386 is an advanced 32-bit microprocessor designed for applications needing very high performance and optimized for multitasking operating systems. The 32-bit registers and data paths support 32-bit addresses and data types. The processor addresses up to four gigabytes of physical memory and 64 terabytes (2**46) of virtual memory. The integrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 80386 allows the simultaneous running of multiple operating systems.

Instruction pipelining, on-chip address translation, a a high bus bandwidth ensure short average instruction execution times and high system throughput. The 80386 processor is capable of execution at sustained rates of between 3 and 4 million instructions per second.

The 80386 offers new testability and debugging features. Testability features include a self-test and direct access to the page translation cache. Four new breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses, for powerful debugging of even ROM-based systems.

Object-code compatibility with all iAPX 86 family members (8086, 8088, 80186, 80188, 80286) means the 80386 offers immediate access to the world's largest microprocesor software base.



231630-49

**Figure 1-1. 80386 Pipelined 32-Bit Microarchitecture**

Unix™ is a Trademark of AT&T Bell Labs.
MS-DOS is a Trademark of MicroSoft Corporation.

**October 1985**
**Order Number: 231630-001**

# 2. BASE ARCHITECTURE

## 2.1 INTRODUCTION

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

The central processing unit consists of the execution unit and instruction unit. The execution unit contains, the eight 32-bit general purpose registers which are used for both address calculation and data operations, a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The multiply and divide logic uses a 1-bit per cycle. The multiply algorithm stops the iteration when the most significant bits of the multiplier are all zero. This allows typical 32-bit multiples to be executed in under one microsecond. 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. Each segment is divided into one or more 4K byte pages. To implement virtual memory system, the 80386 supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes in size. A given region of the linear address space, a segment, can have attributes associated with it. These attributes include its location, size, type (i.e. stack, code or data), and protection characteristics. Each task on an 80386 can have a maximum of 16,381 segments of up to four gigabytes each, thus providing 64 terabytes (trillion bytes) of virtual memory to each task.

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

The 80386 has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the 80386 operates as a very fast 8086, but with 32-bit extensions if desired. Real mode is required primarily to setup the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management, paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each 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 80386 operating system, by the use of paging, and emulation of I/O instructions.

Finally, to facilitate high performance system hardware designs, the 80386 bus interface offers address pipelining, dynamic data bus sizing, and direct Byte Enable signals for each byte of the data bus. These hardware features are described fully beginning in Section 5.

## 2.2 REGISTER OVERVIEW

The 80386 has 32 register resources in the following categories:

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

The registers are a superset of the 8086, 80186 and 80286 registers, so all 16-bit 8086, 80186 and 80286 registers are contained within the 32-bit 80386.

Figure 2-1 shows all of 80386 base architecture registers, which include the general address and data registers, the instruction pointer, and the flags register. The contents of these registers are task-specific, so these registers are automatically loaded with a new context upon a task switch operation.

The base architecture also includes six directly accessible segments, each up to 4 Gbytes in size. The segments are indicated by the selector values placed in 80386 segment registers of Figure 2-1. Various selector values can be loaded as a program executes, if desired.

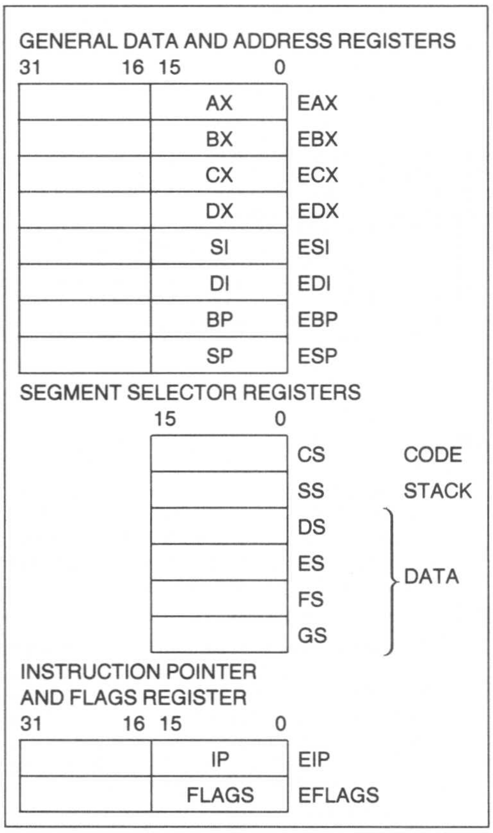


**Figure 2-1. 80386 Base Architecture Registers**

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

The other types of registers Control, System Address, Debug and Test registers are primarily used to simplify the design and debugging of operating systems.

## 2.3 REGISTER DESCRIPTIONS

### 2.3.1 General Purpose Registers

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

The least significant 16 bits of the registers can be accessed separately. This is done by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP, and SP.

Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL, respectively. The higher bytes are named AH, BH, CH and DH, respectively. The individual byte accessibility offers additional flexibility for data operations, but is not used for effective address calculation.

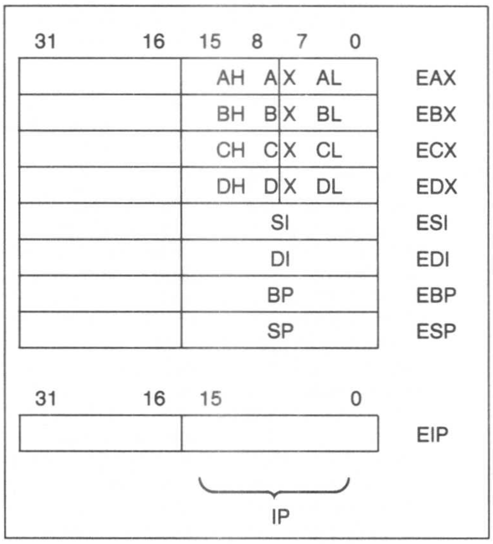


**Figure 2-2. General Registers and Instruction Pointer**

### 2.3.2 Instruction Pointer

The instruction pointer, Figure 2-2, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of EIP contain the 16-bit instruction pointer named IP, which is used by 16-bit addressing.

### 2.3.3 Flags Register

The Flags Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2-3, control certain operations and indicate status of the 80386. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit flag register named FLAGS, which is most useful when executing 8086 and 80286 code.

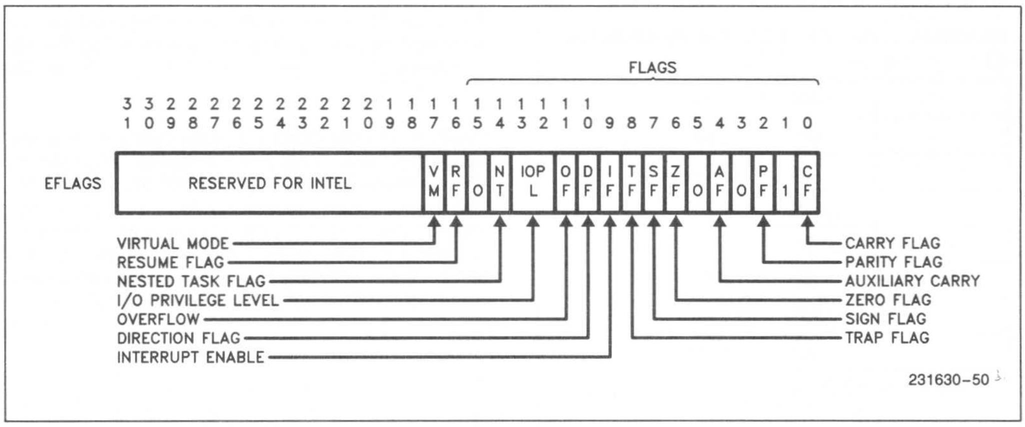


**Figure 2-3. Flags Register**

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 80386 is in Protected Mode, the 80386 will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints or single steps. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, and JMP, CALL, and INT instructions causing a task switch. These instruction set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

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

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

This two-bit field applies to Protected Mode. IOPL indicates the maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8/16/32 bit operations, OF is set according to overflow at bit 7/15/31, respectively.

DF (Direction Flag, bit 10)

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

IF (INTR Enable Flag, bit 9)

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

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 80386 generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debut registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

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

PF (Parity Flags, bit 2)

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

CF (Carry Flag, bit 0)

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

Note in these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

### 2.3.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. Segment registers are shown in Figure 2-4. In Protected Mode, each segment may range in size from one byte up to the entire linear and physi-

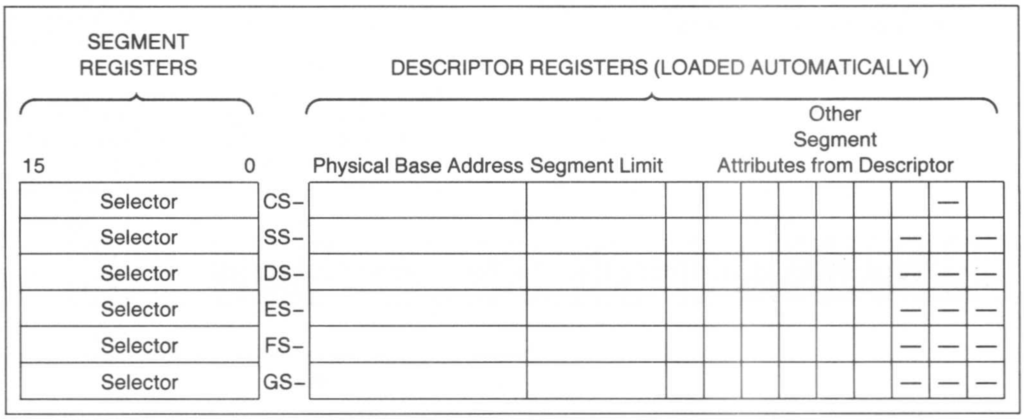

SEGMENT REGISTERS
DESCRIPTOR REGISTERS (LOADED AUTOMATICALLY)
15
0
Physical Base Address
Segment Limit
Other Segment Attributes from Descriptor
Selector
CS-
SS-
DS-
ES-
FS-
GS-

**Figure 2-4. 80386 Segment Registers, and Associated Descriptor Registers**

cal space of the machine, 4 Gbytes ($2^{32}$ bytes). In Real Address Mode, the maximum segment size is fixed at 64 Kbytes ($2^{16}$ bytes).

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

## 2.3.5 Segment Descriptor Registers

The segment descriptor registers are not-programmer visible, yet it is very useful to understand their content. Inside the 80386, a descriptor register (programmer invisible) is associated with each programmer-visible segment register, as shown by Figure 2-4. Each descriptor register holds a 32-bit segment base address, a 32-bit segment limit, and the other necessary segment attributes.

When a selector value is loaded into a segment register, the associated descriptor register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference being requested.

## 2.3.6 Control Registers

The 80386 has three control registers of 32 bits, CR0, CR2 and CR3, to hold machine state of a global nature (not specific to an individual task). These registers, along with System Address Registers described in the next section, hold machine state that affects all tasks in the system. To access the Control Registers, load and store instructions are defined.

**CR0: Machine Control Register (includes 80286 Machine Status Word)**

CR0, shown in Figure 2-5, contains 6 defined bits for control and status purposes. The low-order 16 bits of CR0 are also known as the Machine Status Word, MSW, for compatibility with 80286 Protected Mode. LMSW and SMSW instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. For compatibility with 80286 operating systems the 80386's LMSW instructions work in an identical fashion to the LMSW instruction ont he 80286. (i.e. It only operates on the low-order 16-bits of CR0 and it ignores the new bits in CR0.) New 80386 operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described ahead.

PG (Paging Enable, bit 31)

the PG bit is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

ET (Processor Extension Type, bit 4)

ET indicates the processor extension type (either 80287 or 80387) as detected by the level of the ERROR# input following 80386 reset. The ET bit may also be set or reset by loading CR0 under program control if desired. If ET is set, the 80387-compatible 32-bit protocol is used. If ET is reset, 80287-compatible 16-bit protocol is used.

Note that for strict 80286 compatibility, ET is not affected by the LMSW instruction. When the MSW or CR0 is stored, bit 4 accurately reflects the current state of the ET bit.

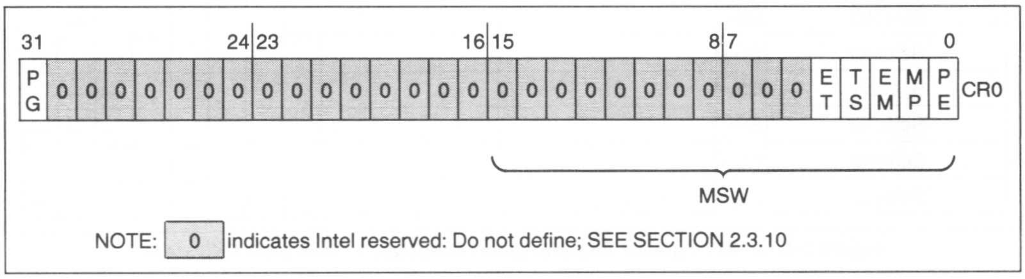

31
24 23
16 15
8 7
0
PG
0
ET
TS
EM
MP
PE
CR0
MSW
NOTE: 0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

**Figure 2-5. Control Register 0**

TS (Task Switched, bit 3)

TS is automatically set whenever a task switch operation is performed. If TS is set, a coprocessor opcode will cause a Coprocessor Not Available trap (exception 7), if the MP bit is also set. The trap handler typically saves the 80287/80387 context belonging to a previous task, loads the 80287/80387 state belonging to the current task, and clears the TS bit before returning to the faulting coprocessor opcode.

EM (Emulate Coprocessor, bit 2)

The EMulate coprocessor bit is set to cause all coprocessor opcodes to generate a Coprocessor Not Available fault (exception 7). It is reset to allow coprocessor opcodes to be executed on an actual 80287 or 80387 coprocessor (this the default case after reset). Note that the WAIT opcode is not affected by the EM bit setting.

MP (Monitor Coprocessor, bit 1)

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

PE (Protection Enable, bit 0)

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

**CR1: reserved**

CR1 is reserved for use in future Intel processors.

**CR2: Page Fault Linear Address**

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

**CR3: Page Directory Base Address**

CR3, shown in Figure 2-6, contains the physical base address of the page directory table. The 80386 page directory table is always page-aligned (4 Kbyte-aligned). Therefore the lowest twelve bits of CR3 are ignored when written and they store as undefined.

A task switch through a TSS which **changes** the value in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the paging unit cache.

## 2.3.7 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286/80386 protection model. These tables or segments are:

GDT (Global Descriptor Table),

IDT (Interrupt Descriptor Table),

LDT (Local Descriptor Table),

TSS (Task State Segment).

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers illustrated in Figure 2-7. These registers are named GDTR, IDTR, LDTR and TR, respecitvely. Section 4 **Protected Mode Architecture** describes the use of these registers.

**GDTR and IDTR**

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

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

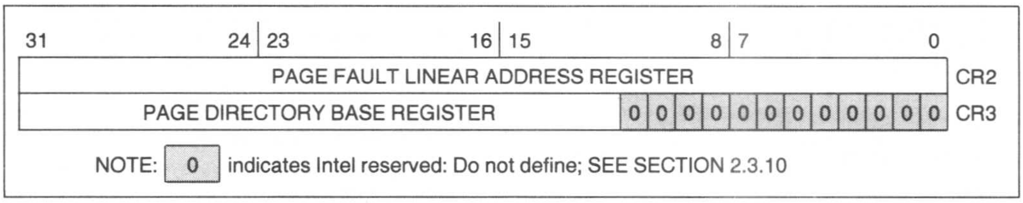

31
24
23
16
15
8
7
0
PAGE FAULT LINEAR ADDRESS REGISTER
CR2
PAGE DIRECTORY BASE REGISTER
0 0 0 0 0 0 0 0 0 0 0 0
CR3
NOTE: 0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

**Figure 2-6. Control Registers 2 and 3**

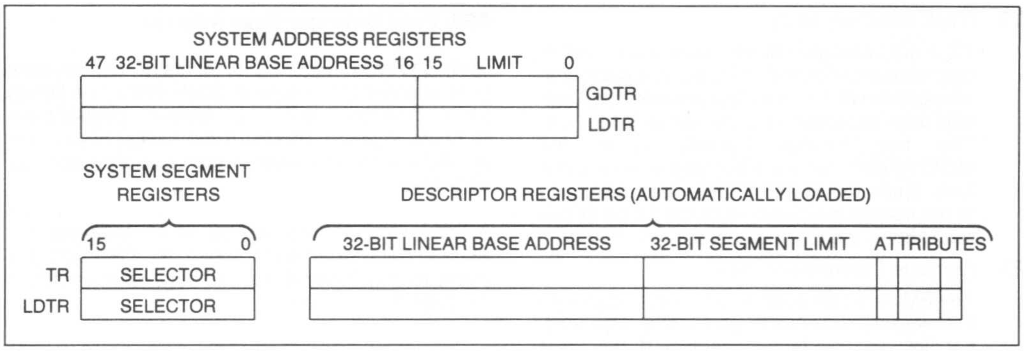

SYSTEM ADDRESS REGISTERS
47 32-BIT LINEAR BASE ADDRESS 16 15 LIMIT 0
GDTR
LDTR
SYSTEM SEGMENT REGISTERS
DESCRIPTOR REGISTERS (AUTOMATICALLY LOADED)
15 0
32-BIT LINEAR BASE ADDRESS
32-BIT SEGMENT LIMIT
ATTRIBUTES
TR
LDTR
SELECTOR
SELECTOR

**Figure 2-7. System Address and System Segment Registers**

**LDTR and TR**

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

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

## 2.3.8 Debug and Test Registers

**Debug Registers:** The six programmer accessible debug registers provide on-chip support for debugging. Debug Register CR0–3 specify the four linear breakpoints. The Debug Control Register DR6 is used to set the breakpoints and the Debug Status Register DR7, displays the current state of the breakpoints. The use of the debug registers is described in section 2.12 **Debugging support.**

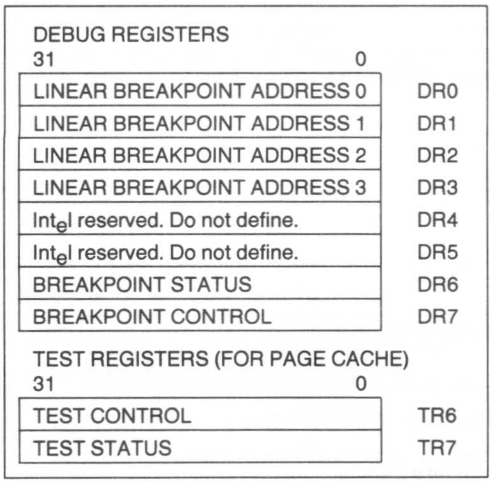

DEBUG REGISTERS
31 0
LINEAR BREAKPOINT ADDRESS 0 DR0
LINEAR BREAKPOINT ADDRESS 1 DR1
LINEAR BREAKPOINT ADDRESS 2 DR2
LINEAR BREAKPOINT ADDRESS 3 DR3
Intel reserved. Do not define. DR4
Intel reserved. Do not define. DR5
BREAKPOINT STATUS DR6
BREAKPOINT CONTROL DR7
TEST REGISTERS (FOR PAGE CACHE)
31 0
TEST CONTROL TR6
TEST STATUS TR7

**Figure 2-8. Debug and Test Registers**

**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 80386. TR6 is the command test register, and TR7 is the data register which contains the data of the Translation Lookaside buffer test. Their use is discussed in section 2.11 **Testability.**

Figure 2-8 shows the Debug and Test registers.

## 2.3.9 Register Accessibility

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2-1 summarizes these differences. See Section 4 **Protected Mode Architecture** for further details.

## 2.3.10 Compatibility

**VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS**

**In the preceding register descriptions, note certain 80386 register bits are undefined. When undefined bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:**

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**

Table 2-1. Register Usage

| Register | Use in Real Mode | | Use in Protected Mode | | Use in Virtual Mode | |
|---|---|---|---|---|---|---|
| | Load | Store | Load | Store | Load | Store |
| General Registers | Yes | Yes | Yes | Yes | Yes | Yes |
| Segment Registers | Yes | Yes | Yes | Yes | Yes | Yes |
| Flag Registers | Yes | Yes | Yes | Yes | IOPL | IOPL* |
| Control Registers | Yes | Yes | PL = 0 | PL = 0 | No | Yes |
| GDTR | Yes | Yes | PL = 0 | Yes | No | Yes |
| IDTR | Yes | Yes | PL = 0 | Yes | No | Yes |
| LDTR | No | No | PL = 0 | Yes | No | No |
| TR | No | No | PL = 0 | Yes | No | No |
| Debug Control | Yes | Yes | PL = 0 | PL = 0 | No | No |
| Test Registers | Yes | PL = 0 | PL = 0 | PL = 0 | No | No |

**NOTES:**
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 8086 Mode.

**5) However, registers which have been previously stored may be reloaded without masking.**

**Depending upon the values of undefined register bits will make your software dependent upon the unspecified 80386 handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 80386-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 80386 REGISTER BITS.**

## 2.4 INSTRUCTION SET

### 2.4.1 Instruction Set Overview

The instruction set is divided into nine categories of operations:

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

These 80386 instructions are listed in Table 2-2.

All 80386 instructions operate on either 0, 1, 2, or 3 operands; where an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 80386 has a 16-byte prefetched 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 80386 (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands, (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

### 2.4.2 80386 Instructions

Table 2-2a. Data Transfer

| GENERAL PURPOSE | |
|---|---|
| MOV | Move operand |
| PUSH | Push operand onto stack |
| POP | Pop operand off stack |
| PUSHA | Push all registers on stack |
| POPA | Pop all registers off stack |
| XCHG | Exchange Operand, Register |
| XLAT | Translate |
| **CONVERSION** | |
| MOVZX | Move byte or Word, Dword, with zero extension |
| MOVSX | Move byte or Word, Dword, sign extended |
| CBW | Convert byte to Word, or Word to Dword |
| CDW | Convert Word to DWORD |
| CDWE | Convert Word to DWORD extended |
| CDQ | Convert DWORD to QWORD |
| **INPUT/OUTPUT** | |
| IN | Input operand from I/O space |
| OUT | Output operand to I/O space |
| **ADDRESS OBJECT** | |
| LEA | Load effective address |
| LDS | Load pointer into D segment register |
| LES | Load pointer into E segment register |
| LFS | Load pointer into F segment register |
| LGS | Load pointer into G segment register |
| LSS | Load pointer into S (Stack) segment register |
| **FLAG MANIPULATION** | |
| LAHF | Load A register from Flags |
| SAHF | Store A register in Flags |
| PUSHF | Push flags onto stack |
| POPF | Pop flags off stack |
| PUSHFD | Push EFlags onto stack |
| POPFD | Pop EFlags off stack |
| CLC | Clear Carry Flag |
| CLD | Clear Direction Flag |
| CMC | Complement Carry Flag |
| STC | Set Carry Flag |
| STD | Set Direction Flag |

Table 2-2b. Arithmetic Instructions

| ADDITION | |
|---|---|
| ADD | Add operand |
| ADC | Add with carry |
| INC | Increment operand by 1 |
| AAA | ASCII adjust for addition |
| DAA | Decimal adjust for addition |
| **SUBTRACTION** | |
| SUB | Subtract operand |
| SBB | Subtract with borrow |
| DEC | Decrement operand by 1 |
| NEG | Negate operand |
| CMP | Compare operands |
| AAS | ASCII Adjust for subtraction |
| **MULTIPLICATION** | |
| MUL | Multiply Double/Single Precision |
| IMUL | Integer multiply |
| AAM | ASCII adjust after multiply |
| **DIVISION** | |
| DIV | Divide unsigned |
| IDIV | Integer Divide |
| AAD | ASCII adjust after division |

Table 2-2c. String Instructions

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

Table 2-2d. Logical Instructions

| LOGICALS | |
|---|---|
| NOT | "NOT" operand |
| AND | "AND" operand |
| OR | "Inclusive OR" operand |
| XOR | "Exclusive OR" operand |
| TEST | "Test" operand |

**Table 2-2d. Logical Instructions** (Continued)

| SHIFTS | |
|---|---|
| SHL/SHR | Shift logical left or right |
| SAL/SAR | Shift arithmetic left or right |
| SHLD/ SHRD | Double shift left or right |
| **ROTATES** | |
| ROL/ROR | Rotate left/right |
| RCL/RCR | Rotate through carry left/right |

**Table 2-2e. Bit Manipulation Instructions**

| SINGLE BIT INSTRUCTIONS | |
|---|---|
| BT | Bit Test |
| BTS | Bit Test and Set |
| BTR | Bit Test and Reset |
| BTC | Bit Test and Complement |
| BSF | Bit Scan Forward |
| BSR | Bit Scan Reverse |
| **BIT STRING INSTRUCTIONS** | |
| IBTS | Insert Bit String |
| XBTS | Exact Bit String |

**Table 2-2f. Program Control Instructions**

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

**Table 2-2f. Program Control Instructions** (Continued)

| UNCONDITIONAL TRANSFERS | |
|---|---|
| CALL | Call procedure/task |
| RET | Return from procedure/task |
| JMP | Jump |
| **ITERATION CONTROLS** | |
| LOOP | Loop |
| LOOPE/ LOOPZ | Loop if equal/zero |
| LOOPNE/ LOOPNZ | Loop if not equal/not zero |
| JCXZ | JUMP if register CX=0 |
| **INTERRUPTS** | |
| INT | Interrupt |
| INTO | Interrupt if overflow |
| IRET | Return from Interrupt |
| CLI | Clear interrupt Enable |
| SLI | Set Interrupt Enable |

**Table 2-2g. High Level Language Instructions**

| | |
|---|---|
| BOUND | Check Array Bounds |
| ENTER | Setup Parameter Block for Entering Procedure |
| LEAVE | Leave Procedure |

**Table 2-2h. Protection Model**

| | |
|---|---|
| SGDT | Store Global Descriptor Table |
| SIDT | Store Interrupt Descriptor Table |
| STR | Store Task Register |
| SLDT | Store Local Descriptor Table |
| LGDT | Load Global Descriptor Table |
| LIDT | Load Interrupt Descriptor Table |
| LTR | Load Task Register |
| LLDT | Load Local Descriptor Table |
| ARPL | Adjust Requested Privilege Level |
| LAR | Load Access Rights |
| LSL | Load Segment Limit |
| VERR/ VERW | Verify Segment for Reading or Writing |
| LMSW | Load Machine Status Word (lower 16 bits of CR0) |
| SMSW | Store Machine Status Word |

**Table 2-2i. Processor Control Instructions**

| | |
|---|---|
| HLT | Halt |
| WAIT | Wait until BUSY# negated |
| ESC | Escape |
| LOCK | Lock Bus |

## 2.5 ADDRESSING MODES

### 2.5.1 Addressing Modes Overview

The 80386 provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

### 2.5.2 Register and Immediate Modes

Two of the addressing modes provide for instructions that operate on register or immediate operands:

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

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

### 2.5.3 Memory Addressing Modes

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following four address elements:

**DISPLACEMENT:** An 8-, or 32-bit immediate value, following the instruction. [16-bit displacements can be used by preceding the instruction with an address prefix.]

**BASE:** The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

**INDEX:** The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

**SCALE:** The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base, Index, and Displacement components which requires one additional clock.

As shown in Figure 2-9, the effective address (EA) of an operand is calculated according to the following formula.

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

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
**EXAMPLE: INC Word PTR [500]**

Register Indirect Mode: A BASE or INDEX register contains the address of the operand.
**EXAMPLE: MOV [ECX], EDX**

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operands offset.
**EXAMPLE: MOV ECX, [EAX + 24]**

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operands offset.
**EXAMPLE: ADD EAX, TABLE[ESI]**

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operands offset.
**EXAMPLE: IMUL EBX, TABLE[ESI*4],7**

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
**EXAMPLE: MOV EAX, [ESI] [EBX]**

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operands offset.
**EXAMPLE: MOV ECX, [EDX*8] [EAX]**

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
**EXAMPLE: ADD EDX, [ESI] [EBP + 00FFFFF0]**

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
**EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP + 80]**

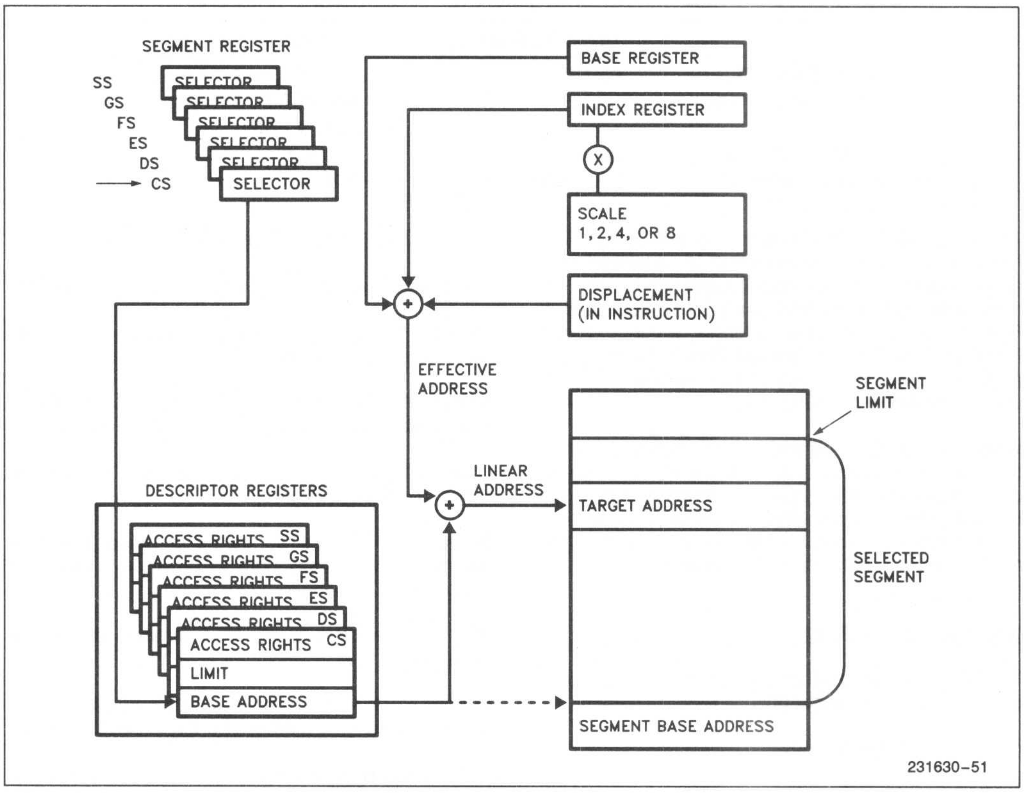


**Figure 2-9. Addressing Mode Calculations**

### 2.5.4 Differences Between 16 and 32 Bit Addresses

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

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32bitMEMORYOP, ASM 386 automatically determines that an Operand Size Prefix is needed and generates it.

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

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

Table 2-3. BASE and INDEX Registers for 16- and 32-Bit Addresses

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

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. An effective address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 80386 addressing modes.

When executing 32-bit code, the 80386 uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 286 model. Table 2-3 illustrates the differences.

## 2.6 DATA TYPES

The 80386 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 80386 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.

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

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

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

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

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 80386 is coupled with a numerics Coprocessor such as the 80287 or the 80387 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 80287 and 80387 numerics coprocessor.

Figure 2-10 illustrates the data types supported by the 80386 and the 80387/80287.

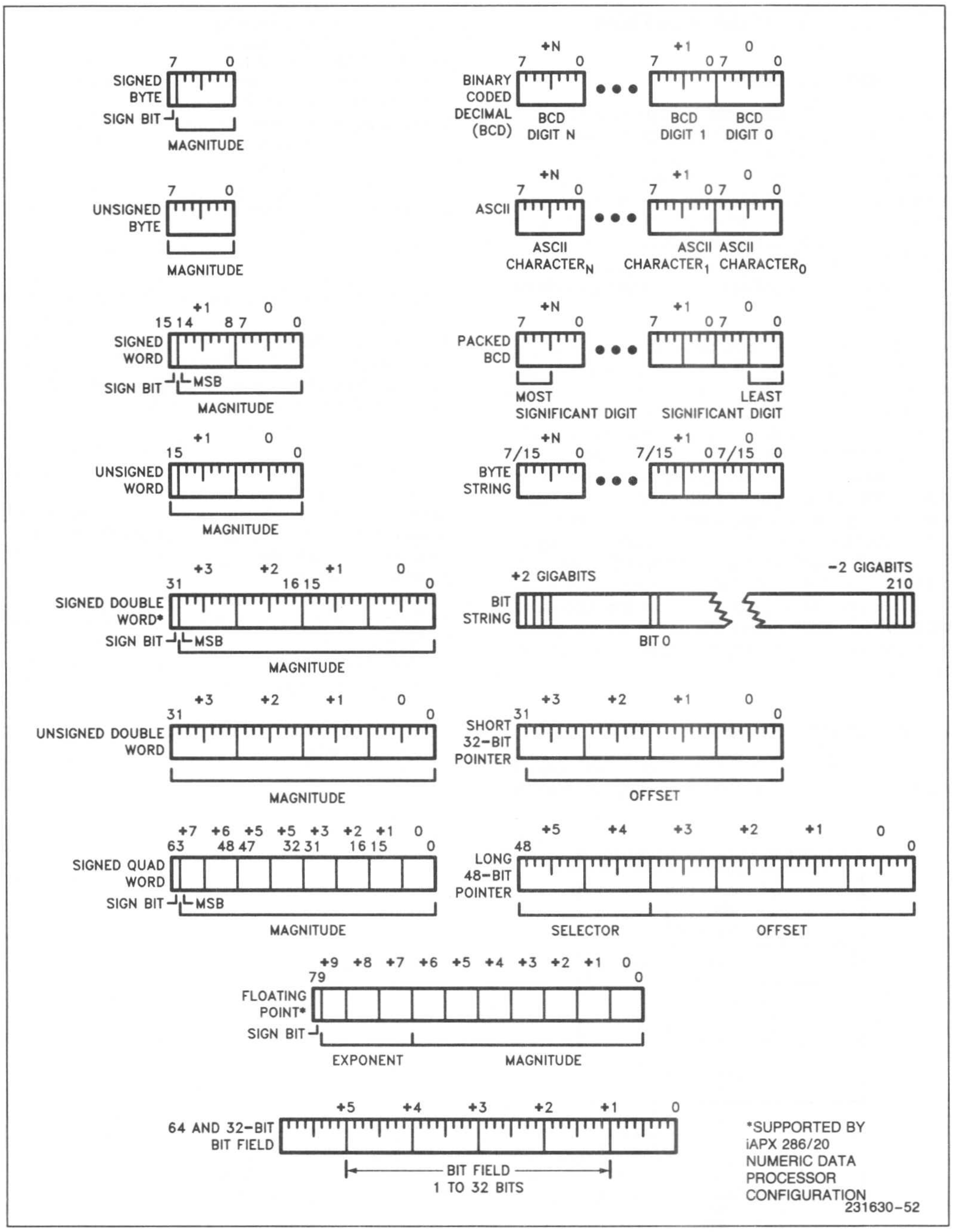


Figure 2-10. 80386 Supported Data Types

## 2.7 MEMORY ORGANIZATION

### 2.7.1 Introduction

Memory on the 80386 is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types the 386 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 80386 supports both pages and segment in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

### 2.7.2 Address Spaces

The 80386 has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on 80386 has a maximum of 16K ($2^{14} - 1$) selectors, and offsets can be 4 gigabytes, ($2^{32}$ bits) this gives a total of $2^{46}$ bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

Figure 2-11 shows the relationship between the various address spaces.

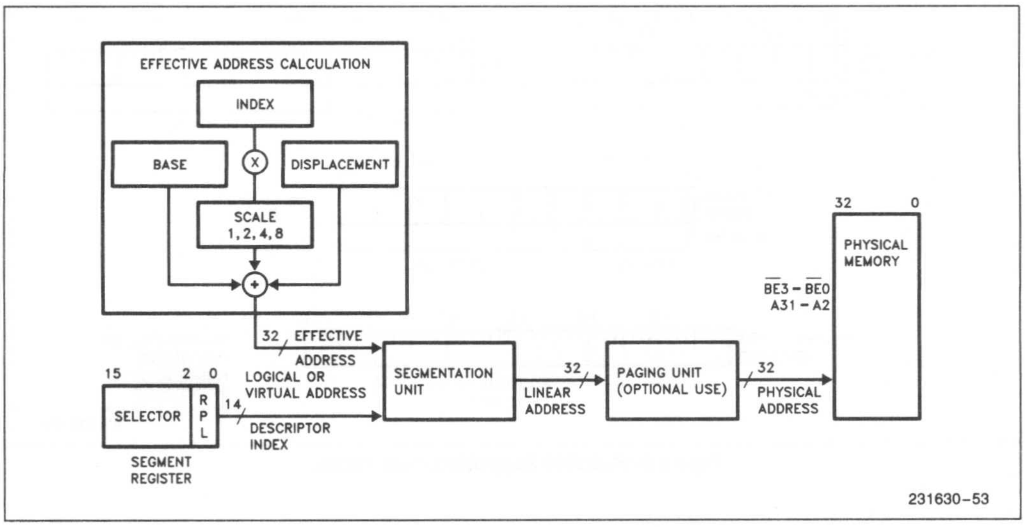


**Figure 2-11. Address Translation**

### 2.7.3 Segment Register Usage

The main data structure used to organize memory is the segment. On the 386, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are three main types of segments: code, data, and stack segments, 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. A default segment register is automatically chosen according to the rules of Table 2-4 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and the SP register as the offset; and Instruction fetches use the CS register. The contents of the Instruction Pointer provides the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2-4. The override prefixes also allow the use of the FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in section 4.1.

## 2.8 I/O SPACE

The 80386 has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 80386 also supports memory-mapped peripherals. The I/O space consists of 64K bytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64K bytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

**Table 2-4. Segment Register Selection Rules**

| Type of Memory Reference | Implied (Default) Segment Use | Segment Override Prefixes Possible |
|---|---|---|
| Code Fetch | CS | None |
| Destination of PUSH, PUSHA instructions | SS | None |
| Source of POP, POPA instructions | SS | 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 |
| [EBX] | 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 |

* Data references for the memory destination of the STOS and MOVS instructions (and REP STOS and REP MOVS) use DI as the base register and ES as the segment, with no override possible.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied in the DL, DX, or EDX registers. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel. The numerics coprocessors also reside in this I/O space at locations 800000F8H - 800000FCH (see section 5).

## 2.9 INTERRUPTS

### 2.9.1 Interrupts and Exceptions

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

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.9.3 and 2.9.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 80386 would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2-5 summarizes the possible interrupts for the 80386 and shows where the return address points to.

The 80386 has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see section 4.1). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

### 2.9.2 Interrupt Processing

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 80386 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 80386 in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

### 2.9.3 Maskable Interrupt

Maskable interrupts are the most common way used by the 80386 to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instruc-

**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 | Coprocessor Tries to Access Data Past the End of a Segment | NO | TRAP** |
| Invalid TSS | 10 | JMP, CALL, IRET, INT | YES | FAULT |
| Segment Not Present | 11 | Segment Register Instructions | YES | FAULT |
| Stack Fault | 12 | Stack References | YES | FAULT |
| General Protection Fault | 13 | Any Memory Reference | YES | FAULT |
| Page Fault | 14 | Any Memory Access or Code Fetch | YES | FAULT |
| Coprocessor Error | 16 | ESC, WAIT | YES | FAULT |
| Intel Reserved | 17–32 | | | |
| Two Byte Interrupt | 0–255 | INT n | NO | TRAP |

* Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.
** Exception 9 no longer occurs on the 80386 due to the improved interface between the 80386 and its coprocessors.

tions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in section 5.

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

### 2.9.4 Non-Maskable Interrupt

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

While executing the NMI servicing procedure, the 80386 will service neither further NMI request, nor 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.

### 2.9.5 Software Interrupts

A third type of interrupt/exception for the 80386 is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

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

A final type of software interrupt, is the single step interrupt. It is discussed in section 2.12.

### 2.9.6 Interrupt Priorities

Since interrupts are recognized only at instruction boundaries it is possible for more than one interrupt to be active at the same time. If there are simultaneous interrupts they will be processed according to the priority shown in Table 2-6. Example: A given instruction causes both a debug trap and a segment not-present exception. The 80386 will first respond to the segment not-present exception (11) by attempting to invoke the exception 11 handler. The exception 11 handler will be interrupted causing the address of the exception 11 handler to be pushed on the stack. The debug exception handler (1) will then be called. After the debug handler is finished, control will pass back to the exception 11 handler. This allows the system designer to debug his exception handlers.

**Table 2-6. Interrupt Processing Priorities**

| Processing Priority | Interrupt/Exception |
|---|---|
| 1 (highest) | Exception faults |
| 2 | TRAP instructions |
| 3 | Debug Traps for this instruction |
| 4 | Debug Faults for next instruction |
| 5 | NMI interrupt |
| 6 | INTR interrupt |

### 2.9.7 Instruction Restart

The 80386 fully supports restarting all instructions after faults. The operating system does not need to participate in the restart process, since the processor will report a page or segment fault with the machine in a state that permits restarting of the faulting instruction after the fault handler has corrected the faulting condition. (e.g. a page fault was generated, the page fault handler brings in the correct page).

Instruction restart is guaranteed except for two conditions: If the instruction causes a task switch to a TSS that is located in a not-present page. If one of the operands is located below any of the current stack pointers (i.e. at a memory address less than the top of stack) or if a floating point operand wraps around in memory.

### 2.9.8 Double Faults

A double fault results when the processor is attempting to handle an exception, and receives another exception during the handling routine. A double fault causes an exception 8. Most exceptions on the 80386 do not count toward the double fault condition (types 1, 2, 3, 4, 5, 6, 7, 9, 14, and 16). Only zero-divide errors (interrupt 0) and the segment exceptions (10, 11, 12, 13) count toward a double fault. Therefore receiving segment not-present exception (11) while responding to a debug exception would not result in a double fault. While a segment fault which occurred during a zero-divide handler would cause a double fault.

Page faults do not count toward double faults. For instance, if an instruction caused both a segment not-present exception (11) and a page not-present fault (interrupt 14) both interrupts would be processed correctly. The segment not-present handler would be invoked causing the correct segment to be loaded from the disk. The instruction would be restarted, and would then cause a page fault. The page fault handler would then bring in the correct page, and execution would proceed. This supports the concept of paging being "underneath" segmentation.

A final cause of double faults is recursive faults (e.g. the page fault handler is not present). These cause an exception 8.

## 2.10 RESET AND INITIALIZATION

When the processor is initialized or Reset the registers have the values shown in Table 2-7. The 80386 will then start executing instructions near the top of physical memory, at location FFFFFFF0H. When the first InterSegment Jump or Call is executed, address lines A20-31 will drop low, and the 80386 will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of Resets.

Driving the RESET input pin HIGH for at least 78 CLK2 periods Resets the 80386. RESET forces the 80386 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 80386 will start executing instructions at the top of physical memory.

**Table 2-7. Register Values after Reset**

| | | |
|---|---|---|
| Flag Word | UUUU0002H | Note 1 |
| Machine Status Word (CR0) | UUUUUUU0H | Note 2 |
| Instruction Pointer | 0000FFF0H | |
| Code Segment | F000H | Note 3 |
| Data Segment | 0000H | |
| Stack Segment | 0000H | |
| Extra Segment (ES) | 0000H | |
| Extra Segment (FS) | 0000H | |
| Extra Segment (GS) | 0000H | |
| All other registers | undefined | |

**NOTES:**
1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, VM (Bit 17) and RF (BIT) 16 are 0 as are all other defined flag bits.
2. CR0: (Machine Status Word). All of the defined fields in the CR0 are 0 (PG Bit 31, TS Bit 3, EM Bit 2, MP Bit 1, and PE Bit 0) except for ET Bit 4 (processor extension type). The ET Bit is set during Reset according to the type of Coprocessor in the system. If the coprocessor is an 80387 then ET will be 1, if the coprocessor is an 80287 or no coprocessor is present then ET will be 0. All other bits are undefined.
3. The Code Segment Register (CS) will have its Base Address set to FFF00000H and Limit set to 0FFFFH. All undefined bits are Intel Reserved and should not be used.

## 2.11 TESTABILITY

### 2.11.1 Self-Test

The 80386 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 80386 can be tested during self-test.

Self-Test is initiated on the 80386 when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is low. The self-test takes about 2**19 CLK2s, or approximately 30 milliseconds with a 16 MHz 80386. 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 and the EDX register are zero (0). If the results of EAX and EDX are not zero then the self-test has detected a flaw in the part.

### 2.11.2 TLB Testing

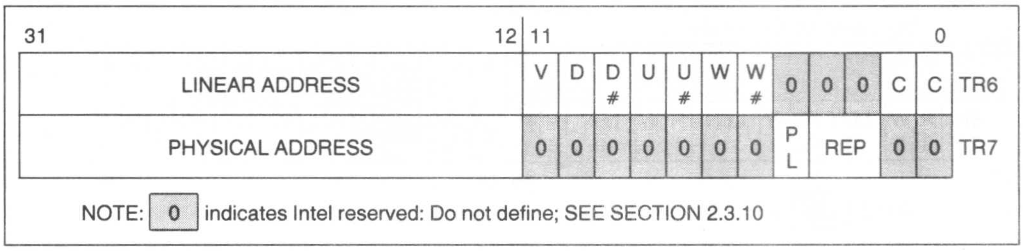
The 80386 also provides a mechanism for testing the Translation Lookaside Buffer (TLB) (see section 4.5.4 **Translation Lookaside Buffer**). This feature is primarily useful for people who wish to write test programs for the 80386. The TLB testing method is unique to the 80386 and may not be continued in future microprocessors. Testing the TLB requires the use of a tester or an assembly language program to drive a test pattern. Paging must be disabled in order to test the functionality of the TLB.

Two test registers provide a means of writing a pattern into the TLB and reading the result. TR6 is the test command register, and TR7 is the test data register. Figure 2-12 shows the two test registers.

The test registers allow two operations to be performed on the TLB: Write New TLB Entry, Perform TLB Lookup. A write to the test command register via the MOV TR6, REG instructions causes a TLB operation to be performed. If bit 0 of TR6 is a 0, a Write New TLB Entry operation occurs if bit 0 is 1 then a TLB Lookup is performed.

## 2.12 DEBUGGING SUPPORT

The 80386 provides several features which simplify the debugging process. Most of these features are designed primarily for software debugging. (Note: Intel will provide a complete set of Hardware/Software debugging tools such as ICE-386 (In Circuit Emulator) and PTM-386 (Pass Through Monitor) to complement the built in debugging features.)



31
12
11
0
LINEAR ADDRESS
V D D# U U# W W# 0 0 0 C C
TR6
PHYSICAL ADDRESS
0 0 0 0 0 0 0 PL REP 0 0
TR7
NOTE: 0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

**Figure 2-12. Test Registers**

The three major types of on-chip debugging aids are the software breakpoints, single stepping and the debug registers. A one byte interrupt INT 3 is available for use by software debuggers to implement breakpoints. The debugger should insert the INT 3 instructions in code sequences. When the INT 3 instruction is encountered execution will proceed at the interrupt handler 3.

The single step interrupt is enabled by setting the single step bit (TF) in the flag word. The TF bit is set by altering the stack image and executing a POPF or IRET instruction. After the TF bit is set, a single step interrupt will occur after the next instruction is executed. The interrupted instruction will push the current Flag register on the stack (with the TF bit set) and then will clear the TF bit, (enabling the single step interrupt handling routine to execute normally). This allows an interrupt handler to be created which can single step through a sequence of instructions. The single step interrupt uses interrupt vector 1, which is supplied internally to the processor.

After completion of the single step interrupt handling routine, the IRET will pop the flag register and then transfer control to the next instruction to be single stepped.

Debug Registers are a unique feature of the 80386. The six program accessible debug registers provide the ability to specify up to four distinct breakpoints. Unlike traditional breakpoints which only support instruction breakpointing, the 80386 debug registers allow breakpoints to be set for data accesses. Thus, if a variable is accidently being overwritten, a breakpoint can be setup to stop execution whenever that variable's contents are being changed.

Figure 2-13 shows the Debug Registers in more detail. DR0–3 contains the linear address of the breakpoint.

**NOTE:**
The linear address may not correspond to the physical address if paging is enable.

DR6 contains the status of the breakpoint registers. The bits within the register have the following meanings:

BT is set if a task switch occurs into a task where the TSS has the DEBUG TRAP bit set.

BS: Enables the debug handlers to distinguish single-step traps from the other debug conditions.

BD: Is set by the hardware if the next instruction accesses a debug register.

B0–B3: These bits are set if a qualified breakpoint has occurred. B0 is set if the Breakpoint 0 has happened etc.

DR7 is the Debug Control Register it is used to enable and qualify the various breakpoints: The bits assignment are assigned as follows:

LENi : This is a two bit field which specifies the length of the breakpoint i . All breakpoints must be aligned; 2 byte breakpoints must be aligned on Word boundaries, and 4 byte breakpoints must be aligned on Dword boundaries.

| | | |
|---|---|---|
| 00 | = | byte length |
| 01 | = | byte length |
| 10 | = | UNDEFINED |
| 11 | = | 4 byte length |

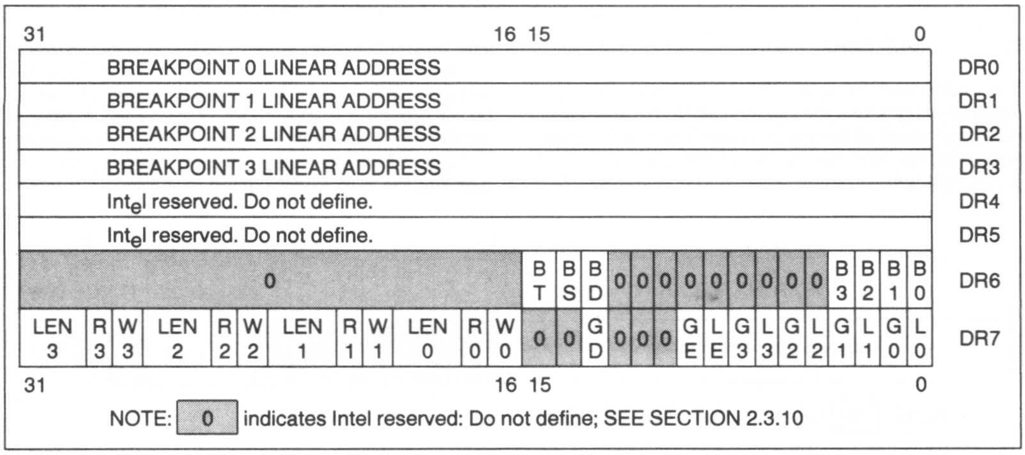

31
16 15
0
BREAKPOINT 0 LINEAR ADDRESS
DR0
BREAKPOINT 1 LINEAR ADDRESS
DR1
BREAKPOINT 2 LINEAR ADDRESS
DR2
BREAKPOINT 3 LINEAR ADDRESS
DR3
Intel reserved. Do not define.
DR4
Intel reserved. Do not define.
DR5
0 BT BS BD 0 0 0 0 0 0 0 0 0 B3 B2 B1 B0
DR6
LEN 3 R 3 W 3 LEN 2 R 2 W 2 LEN 1 R 1 W 1 LEN 0 R 0 W 0 0 0 GD 0 0 0 GE LE G3 L3 G2 L2 G1 L1 G0 L0
DR7
31
16 15
0
NOTE: 0 indicates Intel reserved: Do not define; SEE SECTION 2.3.10

**Figure 2-13. Debug Registers**

RWEi: This two-bit field specifies the type of memory access which must occur in order to activate a breakpoint:

| RWE | Break On |
|---|---|
| 00 | Instruction Execution only |
| 01 | Data Writes Only |
| 10 | UNDEFINED |
| 11 | Data Reads or Writes only (not Instruction fetches) |

GE/LE Global and Local Exact breakpoints: These bits should always be set to 1 when using breakpoints.

Gi/Li Global and Local breakpoint enables. If either Gi = 1 or Li = 1, then breakpoint i is enabled. If these bits are set then any qualified breakpoint (i.e. a breakpoint which matches the condition specified by the LWE bits) will cause the processor to execute the debug handler. The Li bits allows local breakpoints to be set for an individual task but will not affect another task. The Gi bits allow global breakpoints to be set which affect all tasks.

In order to set a breakpoint the processor must be executing at privilege level 0, or in Real Mode. Then, the breakpoint must be set by loading the breakpoint register (via a MOV DRi, REG/MEM instruction), with the address of the breakpoint. Then, the appropriate LEN and RWE must be set up. Finally, the breakpoint enable bits must be set Gi and/or Li.

**NOTE:**
The Bi bits in DR6 will always show any qualified breakpoints, but unless Gi or Li are set the processor will not execute the debug routine at interrupt 1.

## 3. REAL MODE ARCHITECTURE

### 3.1 REAL MODE INTRODUCTION

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 80386. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the 80386 instructions are available in Real Mode. The default operand size in Real Mode is 16-bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 80386 in Real Mode is 64K bytes so 32-bit addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

### 3.2 MEMORY ADDRESSING

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19, BE0–B3 are active. (Exception, the high address lines A20–A31 are high until an intersegment jump or call is executed (see section 2.10)).

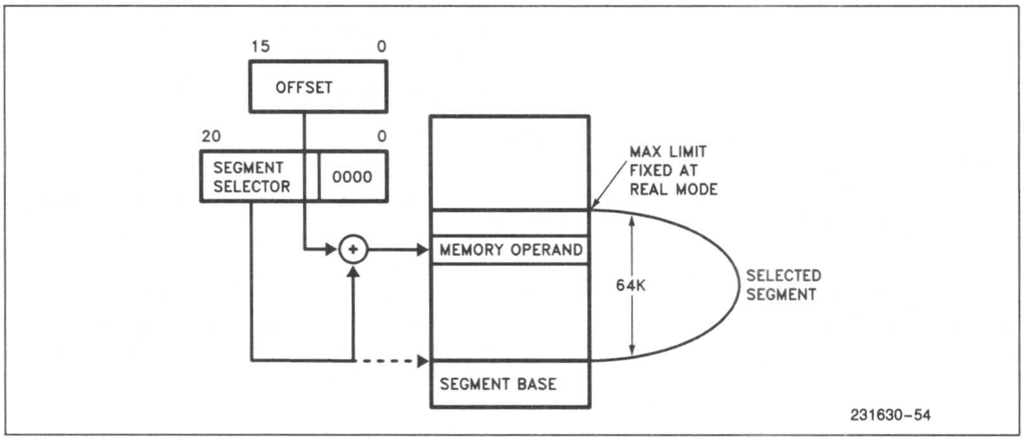


**Figure 3-1. Real Address Mode Addressing**

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

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 80386 will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment. (i.e. if an operand has an offset greater the FFFFH, example a word with a low byte at FFFFH and the high byte at 0000H)

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

## 3.3 RESERVED LOCATIONS

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

## 3.4 INTERRUPTS

Many of the exceptions shown in Table 2-5 and discussed in section 2.9 are not applicable to Real Mode operation, in particular exceptions 10, 11, 12, 14, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode Table 3-1 identifies these exceptions.

## 3.5 SHUTDOWN AND HALT

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 80386 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

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

An interrupt or an exception occur (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e. There is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even. (e.g. pushing a value on the stack when SP = 0001 resulting a stack segment greater than FFFFH)

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 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 via the RESET input.

**Table 3-1**

| Function | Interrupt Number | Related Instructions | Return Address Location |
|---|---|---|---|
| Interrupt table limit too small | 8 | INT Vector is not within table limit | Before Instruction |
| Segment overrun exception | 13 | Word memory reference With offset = FFFFH or Inst. an attempt to execute past the end of a segment | Before Instruction |

# 4. PROTECTED MODE ARCHITECTURE

## 4.1 INTRODUCTION

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

## 4.2 ADDRESSING MECHANISM

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

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4-1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 80386, as such paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4-2 shows the complete 80386 addressing mechanism with paging enabled.

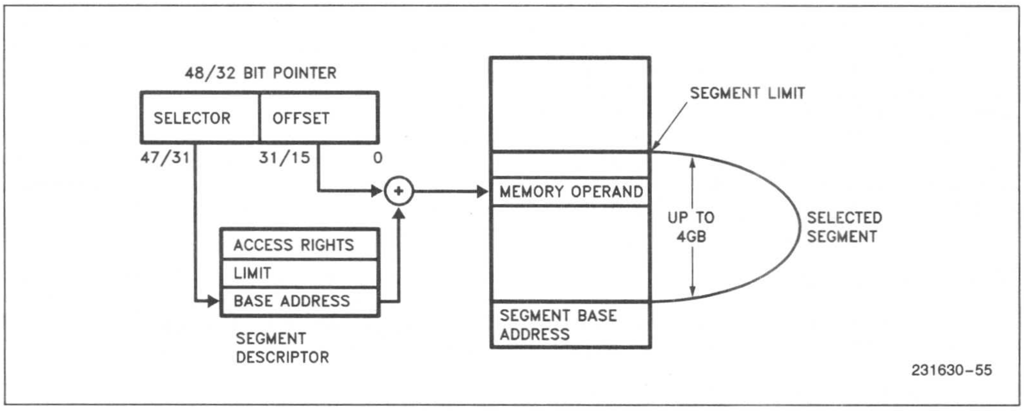


**Figure 4-1. Protected Mode Addressing**

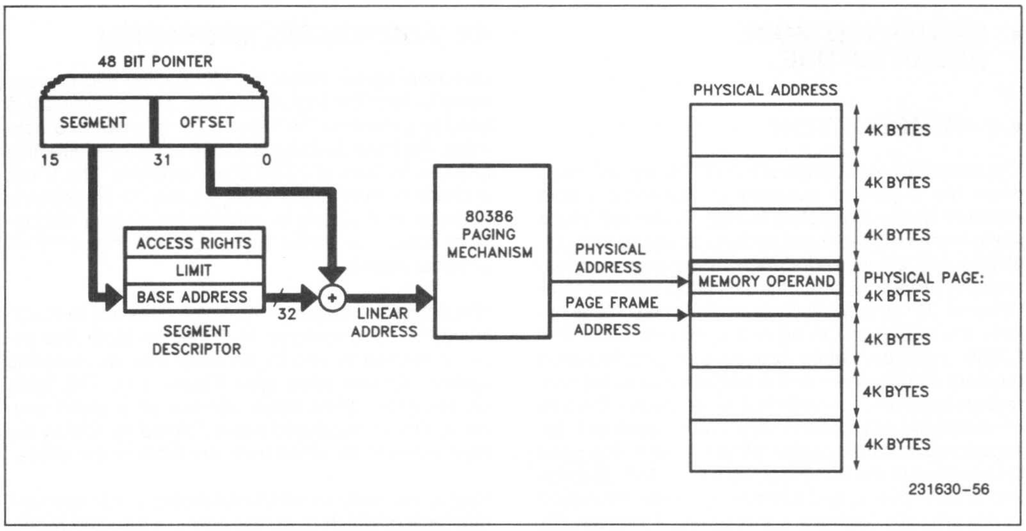


**Figure 4-2. Paging and Segmentation**

## 4.3 SEGMENTATION

### 4.3.1 Segmentation Introduction

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

### 4.3.2 Terminology

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

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

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

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

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

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

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

## 4.3.3 Descriptor Tables

### 4.3.3.1 DESCRIPTOR TABLES INTRODUCTION

The descriptor tables define all of the segments which are used in an 80386 system. There are three types of tables on the 80386 which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays, they can range in size between 8 bytes and 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 or each table.

Each of the tables has a register associated with it the GDTR, LDTR, and the IDTR; see Figure 4-3. The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

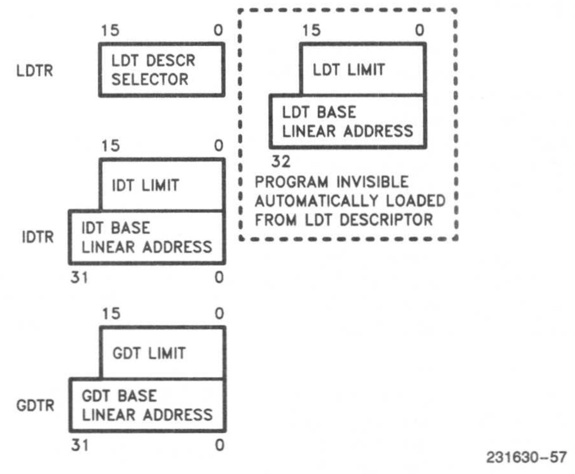


**Figure 4-3. Descriptor Table Registers**

#### 4.3.3.2 GLOBAL DESCRIPTOR TABLE

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e. interrupt and trap descriptors). Every 386 system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

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

#### 4.3.3.3 LOCAL DESCRIPTOR TABLE

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

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

#### 4.3.3.4 INTERRUPT DESCRIPTOR TABLE

The third table needed for 80386 systems is the Interrupt Descriptor Table. (See Figure 4-4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See 2.9 **Interrupts**).

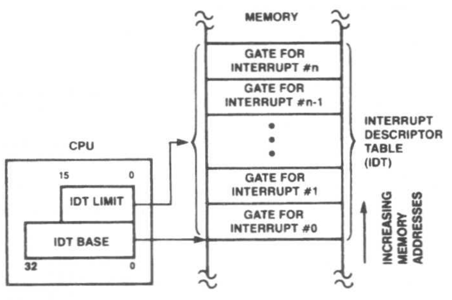


**Figure 4-4. Interrupt Descriptor Table Register Use**

### 4.3.4 Descriptors

#### 4.3.4.1 DESCRIPTOR ATTRIBUTE BITS

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

| 31 | | | | | | | | | | | 0 | BYTE ADDRESS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SEGMENT BASE 15 . . . 0 | | | | | | SEGMENT LIMIT 15 . . . 0 | | | | | | 0 |
| BASE 31 . . . 24 | G | D | 0 | 0 | LIMIT 19 . . . 16 | P | DPL | S | TYPE | A | BASE 23 . . . 16 | +4 |

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

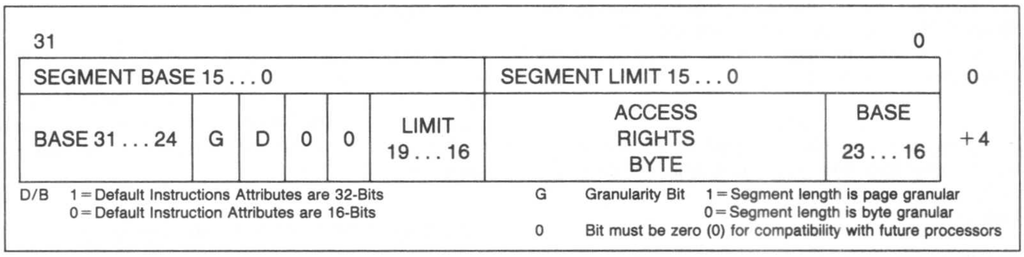
**Figure 4-5. Segment Descriptors**

attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4-5 shows the general format of a descriptor. All segments on the 80386 have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The 80386 has two main categories of segments system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

#### 4.3.4.2 386 CODE, DATA DESCRIPTORS (S=0)

Figure 4-6 shows the general format of a code and data descriptor and Table 4-1 illustrates how the bits in Access Right Byte are interpreted.



31
0
SEGMENT BASE 15...0
SEGMENT LIMIT 15...0
0
BASE 31...24
G
D
0
0
LIMIT 19...16
ACCESS RIGHTS BYTE
BASE 23...16
+4
D/B 1=Default Instructions Attributes are 32-Bits
0=Default Instruction Attributes are 16-Bits
G Granularity Bit 1=Segment length is page granular
0=Segment length is byte granular
0 Bit must be zero (0) for compatibility with future processors

**Figure 4-6. Segment Descriptors**

**Table 4-1. Access Rights Byte Definition for Code and Data Descriptions**

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

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 80386 segments can be one megabyte long with byte granularity (G = 0) or four gigabytes with page granularity (G = 1), (i.e. $2^{20}$ pages each page is 4K bytes in length). The granularity is totally unrelated to paging. A 80386 system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

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

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

The **D** bit indicates the default length for operands and effective addresses. If D = 1 then 32-bit operands and 32-bit addressing modes are assumed. If D = 0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 286 code segments will execute on the 80386 assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C = 1, can be executed and shared by programs at different privilege levels. (See section 4.4 **Protection**.)

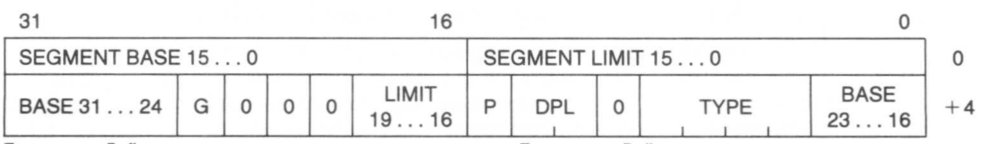
Segments identified as data segments (E = 0, S = 1) are used for two types of 80386 segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

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

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

### 4.3.4.3 SYSTEM DESCRIPTOR FORMATS

System segments describe information about operating system tables, tasks, and gates. Figure 4-7 shows the general format of system segment descriptors, and the various types of system segments. 80386 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.



| Type | Defines | Type | Defines |
|---|---|---|---|
| 0 | Invalid | 8 | Invalid |
| 1 | Available 286 TSS | 9 | Available 386 TSS |
| 2 | LDT | A | Undefined (Intel Reserved) |
| 3 | Busy 286 TSS | B | Busy 386 TSS |
| 4 | 286 Call Gate | C | 386 Call Gate |
| 5 | 286 Task Gate | D | Undefined (Intel Reserved) |
| 6 | 286 Interrupt Gate | E | 386 Interrupt Gate |
| 7 | 286 Trap Gate | F | 386 Trap Gate |

**Figure 4-7. System Segments Descriptors**

#### 4.3.4.4 LDT DESCRIPTORS (S = 0, TYPE = 2)

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

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

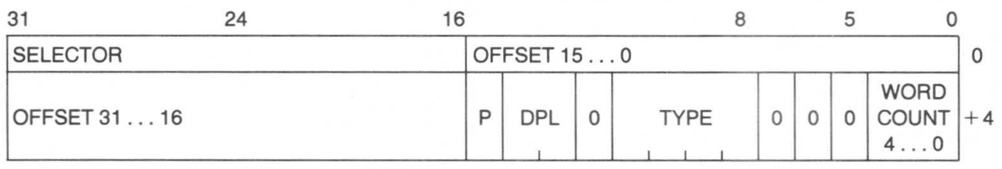
A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e. on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 286 or a 386 TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

#### 4.3.4.6 GATE DESCRIPTORS (S = 0, TYPE = 4–7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

Figure 4-8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

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



**Gate Descriptor Fields**

| Name | Value | Description |
|---|---|---|
| Type | 4 | 286 call gate |
| | 5 | Task gate |
| | 6 | 286 interrupt gate |
| | 7 | 286 trap gate |
| | C | 386 call gate |
| | E | 386 interrupt gate |
| | F | 386 trap gate |
| P | 0 | Descriptor contents are not valid |
| | 1 | Descriptor contents are valid |

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for 386 gates, and 16-bit quantities for 286 gates.

| | | |
|---|---|---|
| DESTINATION SELECTOR | 16-bit selector | Selector to the target code segment or Selector to the target task state segment for task gate |
| DESTINATION OFFSET | offset 16-bit 286 32-bit 386 | Entry point within the target code segment |

**Figure 4-8. Gate Descriptor Formats**

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

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

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see section 4.4 **Protection**). The S field bit 4 of the access rights byte must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4-8.

### 4.3.4.7 DIFFERENCES BETWEEN 386 AND 286 DESCRIPTORS

In order to provide operating system compatibility between the 80286 and 80386, the 386 supports all of the 80286 segment descriptors. Figure 4-9 shows the general format of an 80286 system segment descriptor. The only differences between 286 and 386 descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 386. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 386 system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

By supporting 80286 system segments the 80386 is able to execute 286 application programs on a 80386 operating system. This is possible because the processor automatically understands which descriptors are 286-style descriptors and which descriptors are 386-style descriptors. In particular, if the upper word of a descriptor is zero then that descriptor is a 286-style descriptor.

The only other differences between 286-style descriptors and 386 descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 286 call gates and 32-bit quantities for 386 call gates. The B bit controls the size of PUSHes when using a call gate; if B = 0 PUSHes are 16 bits, if B = 1 PUSHes are 32 bits.

### 4.3.4.8 SELECTOR FIELDS

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4-10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

### 4.3.4.9 SEGMENT DESCRIPTOR CACHE

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

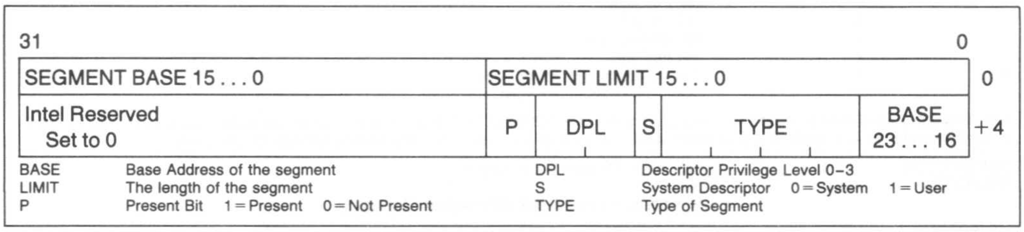

31
0
SEGMENT BASE 15 . . . 0
SEGMENT LIMIT 15 . . . 0
0
Intel Reserved Set to 0
P
DPL
S
TYPE
BASE 23 . . . 16
+4
BASE Base Address of the segment
LIMIT The length of the segment
P Present Bit 1 = Present 0 = Not Present
DPL Descriptor Privilege Level 0–3
S System Descriptor 0 = System 1 = User
TYPE Type of Segment

**Figure 4-9. 286 Code and Data Segment Descriptors**

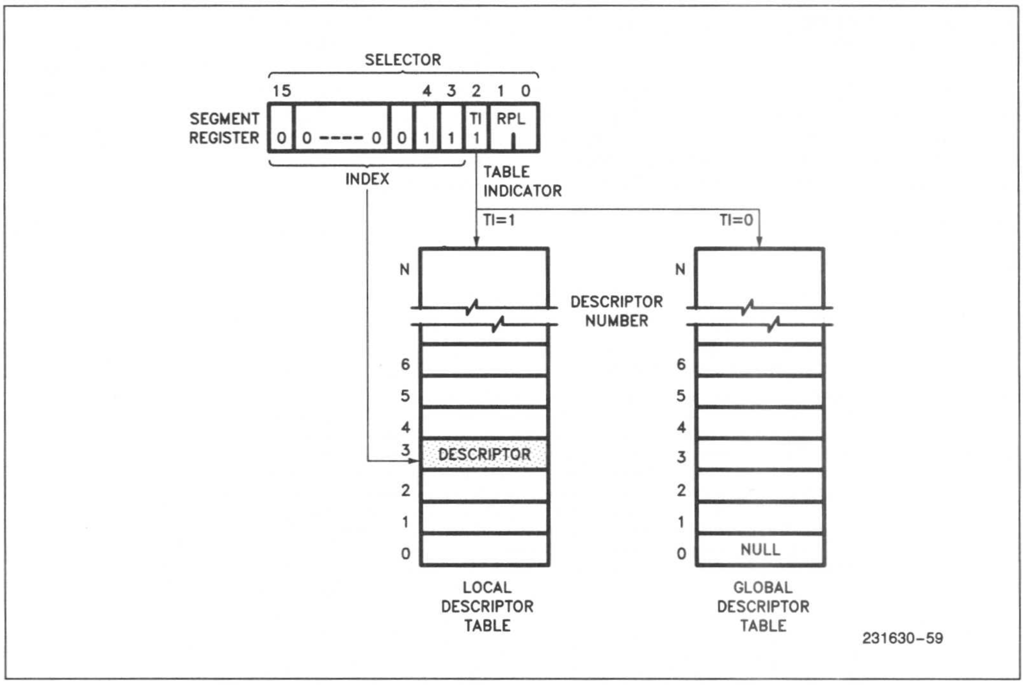
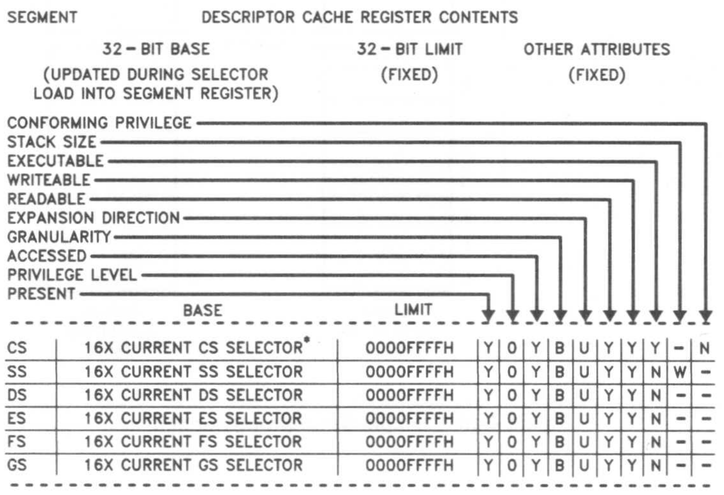


**Figure 4-10. Example Descriptor Selection**

### 4.3.4.10 SEGMENT DESCRIPTOR REGISTER SETTINGS

The contents of the segment descriptor cache vary depending on the mode the 80386 is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-11.

For compatiblity with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.



| | BASE | LIMIT | PRESENT | PRIVILEGE LEVEL | ACCESSED | GRANULARITY | EXPANSION DIRECTION | READABLE | WRITEABLE | EXECUTABLE | STACK SIZE | CONFORMING PRIVILEGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 16X CURRENT CS SELECTOR* | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | Y | – | N |
| SS | 16X CURRENT SS SELECTOR | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | N | W | – |
| DS | 16X CURRENT DS SELECTOR | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | N | – | – |
| ES | 16X CURRENT ES SELECTOR | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | N | – | – |
| FS | 16X CURRENT FS SELECTOR | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | N | – | – |
| GS | 16X CURRENT GS SELECTOR | 0000FFFFH | Y | 0 | Y | B | U | Y | Y | N | – | – |

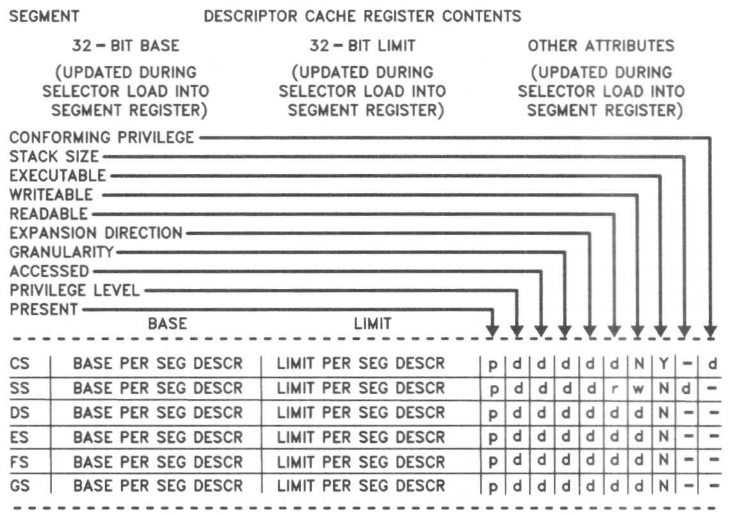
*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (e.g. intersegment CALL, or intersegment JMP, or INT). (See Figure 4-13 Example.)

Key:

| | | | |
|---|---|---|---|
| Y | = yes | D | = expand down |
| N | = no | B | = byte granularity |
| 0 | = privilege level 0 | P | = page granularity |
| 1 | = privilege level 1 | W | = push/pop 16-bit words |
| 2 | = privilege level 2 | F | = push/pop 32-bit dwords |
| 3 | = privilege level 3 | – | = does not apply to that segment cache register |
| U | = expand up | | |

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

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.


SEGMENT
DESCRIPTOR CACHE REGISTER CONTENTS
32 - BIT BASE
(UPDATED DURING SELECTOR LOAD INTO SEGMENT REGISTER)
32 - BIT LIMIT
(UPDATED DURING SELECTOR LOAD INTO SEGMENT REGISTER)
OTHER ATTRIBUTES
(UPDATED DURING SELECTOR LOAD INTO SEGMENT REGISTER)
CONFORMING PRIVILEGE
STACK SIZE
EXECUTABLE
WRITEABLE
READABLE
EXPANSION DIRECTION
GRANULARITY
ACCESSED
PRIVILEGE LEVEL
PRESENT
BASE
LIMIT

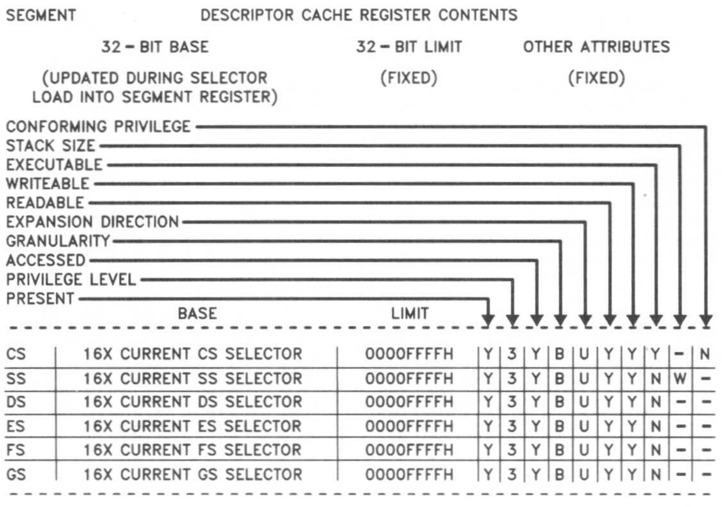
| SEGMENT | BASE | LIMIT | PRESENT | PRIVILEGE LEVEL | ACCESSED | GRANULARITY | EXPANSION DIRECTION | READABLE | WRITEABLE | EXECUTABLE | STACK SIZE | CONFORMING PRIVILEGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | d | N | Y | – | d |
| SS | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | r | w | N | d | – |
| DS | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | d | d | N | – | – |
| ES | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | d | d | N | – | – |
| FS | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | d | d | N | – | – |
| GS | BASE PER SEG DESCR | LIMIT PER SEG DESCR | p | d | d | d | d | d | d | N | – | – |

231630-61

Key:
Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

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

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.



| SEGMENT | BASE | LIMIT | PRESENT | PRIVILEGE LEVEL | ACCESSED | GRANULARITY | EXPANSION DIRECTION | READABLE | WRITEABLE | EXECUTABLE | STACK SIZE | CONFORMING PRIVILEGE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CS | 16X CURRENT CS SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | Y | – | N |
| SS | 16X CURRENT SS SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | N | W | – |
| DS | 16X CURRENT DS SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | N | – | – |
| ES | 16X CURRENT ES SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | N | – | – |
| FS | 16X CURRENT FS SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | N | – | – |
| GS | 16X CURRENT GS SELECTOR | 0000FFFFH | Y | 3 | Y | B | U | Y | Y | N | – | – |

Key:

| | | | |
|---|---|---|---|
| Y | = yes | D | = expand down |
| N | = no | B | = byte granularity |
| 0 | = privilege level 0 | P | = page granularity |
| 1 | = privilege level 1 | W | = push/pop 16-bit words |
| 2 | = privilege level 2 | F | = push/pop 32-bit dwords |
| 3 | = privilege level 3 | – | = does not apply to that segment cache register |
| U | = expand up | | |

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

## 4.4 PROTECTION

### 4.4.1 Protection Concepts

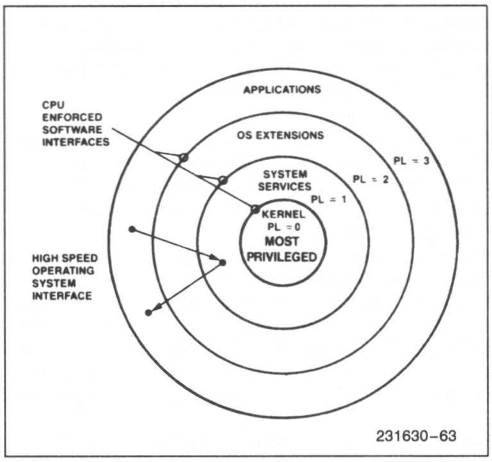


**Figure 4-14. Four-Level Hierachical Protection**

The 80386 has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 80386 provides the protection as part of its integrated Memory Management Unit. The 80386 offers an additional type of protection on a page basis, when paging is enabled (See section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the 80386 paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

### 4.4.2 Rules of Privilege

The 80386 controls access to both data and procedures between levels of a task, according to the following rules.

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

### 4.4.3 Privilege Levels

#### 4.4.3.1 TASK PRIVILEGE

At any point in time, a task on the 80386 always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

#### 4.4.3.2 SELECTOR PRIVILEGE (RPL)

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

#### 4.4.3.3 I/O PRIVILEGE

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

#### 4.4.3.4 PRIVILEGE VALIDATION

The 80386 provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4-2 summarizes the selector validation procedures available for the 80386.

**Table 4-2. Pointer Test Instructions**

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

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

#### 4.4.3.5 DESCRIPTOR ACCESS

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

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 80386 makes protection validation checks. The processor first checks to see if the segment is not a null segment, an invalid segment generates an exception 11. Then it checks to see if the selector refers to the correct type of segment. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments. The data access rules are specified in section 4.2.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

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

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

### 4.4.4 Privilege Level Transfers

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4-3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

Table 4-3. Descriptor Types Used for Control Transfer

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

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

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

**The privilege rules require that:**

— Privilege level transitions can only occur via gates.

— JMPs must be made to a non-conforming code segment with the same privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

### 4.4.5 Call Gates

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 386 call gate is activated, the following actions occur.

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

The procedure is identical for 286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

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

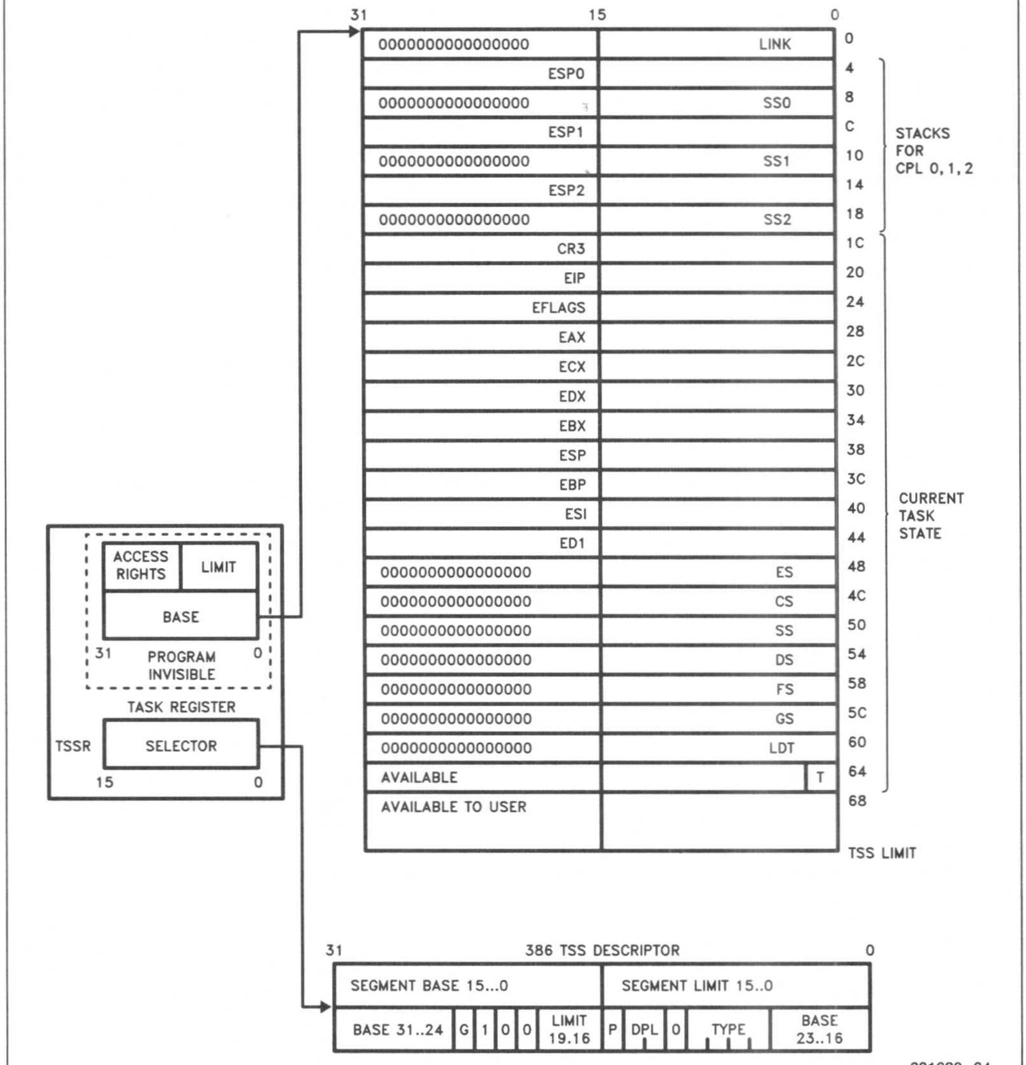
### 4.4.6 Task Switching

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 80386 directly supports this operation by providing a task switch instruction in hardware. The 80386 task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4-15) containing the entire 80386 execution state while a task gate descriptor contains a TSS selector. The 80386 supports both 286 and 386 style TSSs. Figure 4-16 shows a 286 TSS. The limit of a 386 TSS must be greater than 0064H (002BH for a 286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80386 called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:



Type = 9 Available 386 TSS,
Type = 13 Busy 386 TSS

**Figure 4-15. 386 TSS and TSS Registers**

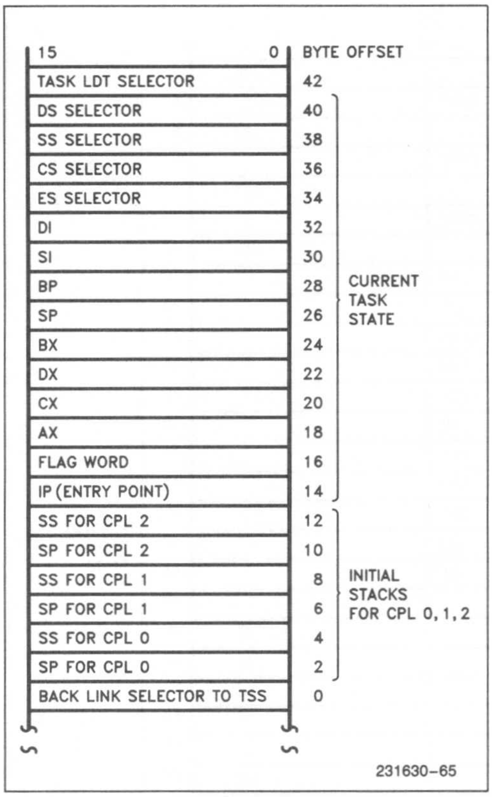


**Figure 4-16. 286 TSS**

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

The 386 task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. A 286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see section 4.6 **Virtual Mode**).

The coprocessor's state is not automatically saved when a task switch occurs, because the incoming task may not use the coprocessor. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80386 switches task, it sets the TS bit. The 80386 detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor. A processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the 386 TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

## 4.4.7 Initialization and Transition to Protected Mode

Since the 80386 begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4-17 shows the tables and Figure 4-18 the descriptors needed for a simple Protected Mode 80386 system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 80386 in Protected Mode.

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

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

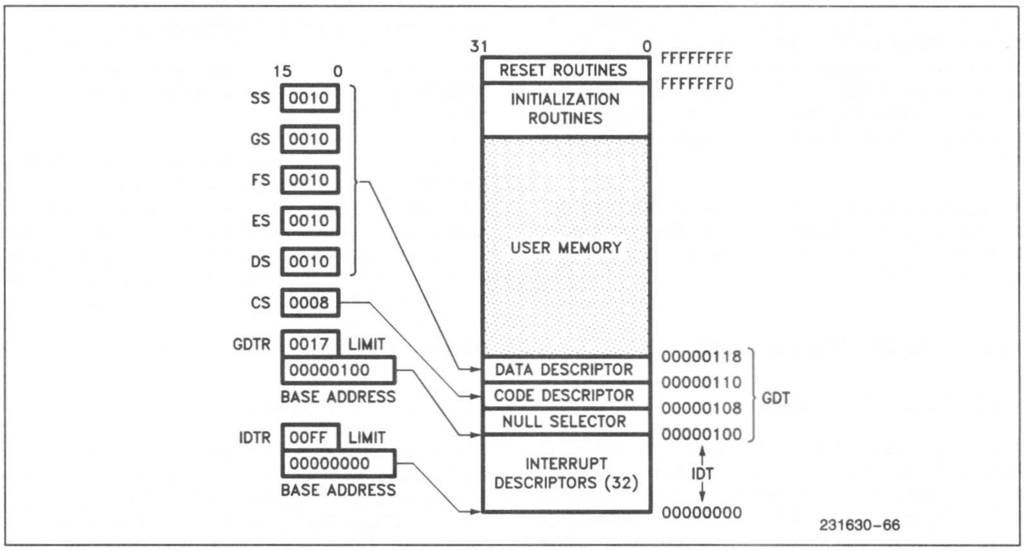


Figure 4-17. Simple Protected System

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DATA DESCRIPTOR | SEGMENT BASE 15 . . . 0<br>0118 (H) | | | | | | SEGMENT LIMIT 15 . . . 0<br>FFFF (H) | | | | | | | | |
| 2 | BASE 31 . . . 24<br>00 (H) | G<br>1 | D<br>1 | 0 | 0 | LIMIT<br>19.16<br>F (H) | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | BASE 23 . . . 16<br>00 (H) |
| CODE DESCRIPTOR | SEGMENT BASE 15 . . . 0<br>0118 (H) | | | | | | SEGMENT LIMIT 15 . . . 0<br>FFFF (H) | | | | | | | | |
| 1 | BASE 31 . . . 24<br>00 (H) | G<br>1 | D<br>1 | 0 | 0 | LIMIT<br>19.16<br>F (H) | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | BASE 23 . . . 16<br>00 (H) |
| | NULL | | | | | | DESCRIPTOR | | | | | | | | |
| 0 | | | | | | | | | | | | | | | |
| | 31 | | 24 | | | 16 | | | | 8 | | | | | 0 |

Figure 4-18. Descriptors for Simple System

### 4.4.8 Tools for Building Protected Systems

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 80386 system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSS) in a high-level language.

## 4.5 PAGING

### 4.5.1 Paging Concepts

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

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

## 4.5.2 Paging Organization

### 4.5.2.1 PAGE MECHANISM

The 80386 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 80386: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 80386 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 4-19 shows how the paging mechanism works.

### 4.5.2.2 PAGE DESCRIPTOR BASE REGISTER

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

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.4 **Translation Lookaside Buffer**).

### 4.5.2.3 PAGE DIRECTORY

The Page Directory is 4K bytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4-20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

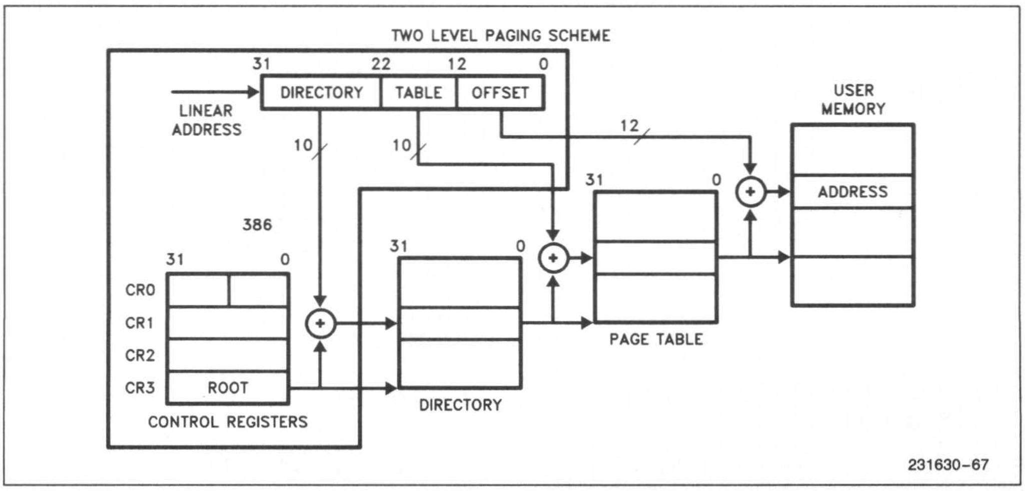


Figure 4-19. Paging Mechanism

| 31 ... 12 | 11 10 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| PAGE TABLE ADDRESS 31..12 | OS RESERVED | 0 | 0 | D | A | 0 | 0 | U — S | R — W | P |

Figure 4-20. Page Directory Entry (Points to Page Table)

| 31 … 12 | 11 10 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| PAGE FRAME ADDRESS 31..12 | OS RESERVED | 0 | 0 | D | A | 0 | 0 | U/S | R/W | P |

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

#### 4.5.2.4 PAGE TABLES

Each Page Table is 4K bytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4-21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

#### 4.5.2.5 PAGE DIRECTORY/TABLE ENTRIES

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation if P = 0 the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 80386 for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 80386, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4-20 and Figure 4-21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Use.

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

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

The 80386 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 the Page Table Entries and using these bits to address the page.

Example: If the U/S and R/W bits for the Page Directory entry was 10 and the U/S R/W bits for the Page Table Entry were 01, the access rights for the page would be 01, the numerically smaller of the two. Table 4-4 shows the affect of the U/S and R/W bits on accessing memory.

**Table 4-4. Protection Provided by R/W and U/S**

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

However a given segment can be easily made read-only for level 0, 1, or 2 via the use of segmented protection mechanisms. (Section 4.4 **Protection**).

### 4.5.4 Translation Lookaside Buffer

The 80386 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 80386 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 about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4-22 illustrates how the TLB complements the 80386's paging mechanism.

### 4.5.5 Paging Operation

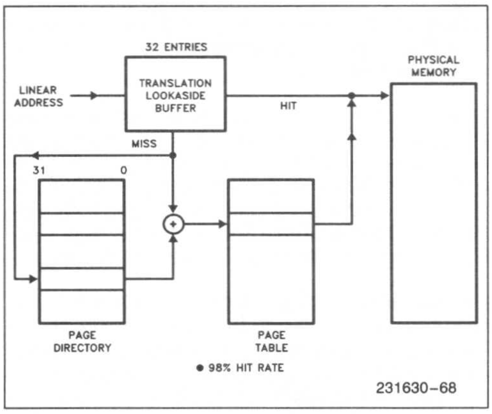


**Figure 4-22. Translation Lookaside Buffer**

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 80386 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 80386 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 80386 will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault an Exception 14.

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

The 16-bit error code is used by the operating system to determine how to handle the page fault Figure 4-23A shows the format of the page-fault error code and the interpretation of the bits.

**NOTE:**
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4-23B indicates what type of access caused the page fault.

| 15 | | | | | | | | | | | | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| U | U | U | U | U | U | U | U | U | U | U | U | U | U/S | W/R | P |

**Figure 4-23A. Page Fault Error Code Format**

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

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

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

**U**: UNDEFINED

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

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

**Figure 4-23B. Type of Access Causing Page Fault**

### 4.5.6 Operating System Responsibilities

The 80386 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 set the P present bit of page table entry to zero the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

## 4.6 VIRTUAL 8086 ENVIRONMENT

### 4.6.1 Executing 8086 Programs

The 80386 allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 80386 protection mechanism. In particular, the 80386 allows the simultaneous execution of 8086 operating systems and its applications, and an 80386 operating system and both 80286 and 80386 applications. Thus, in a multi-user 80386 computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4-24 illustrates this concept.

### 4.6.2 Virtual 8086 Mode Addressing Mechanism

One of the major differences between 80386 Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 80386 allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 80386. 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.

### 4.6.3 Paging In Virtual Mode

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 80386. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications.

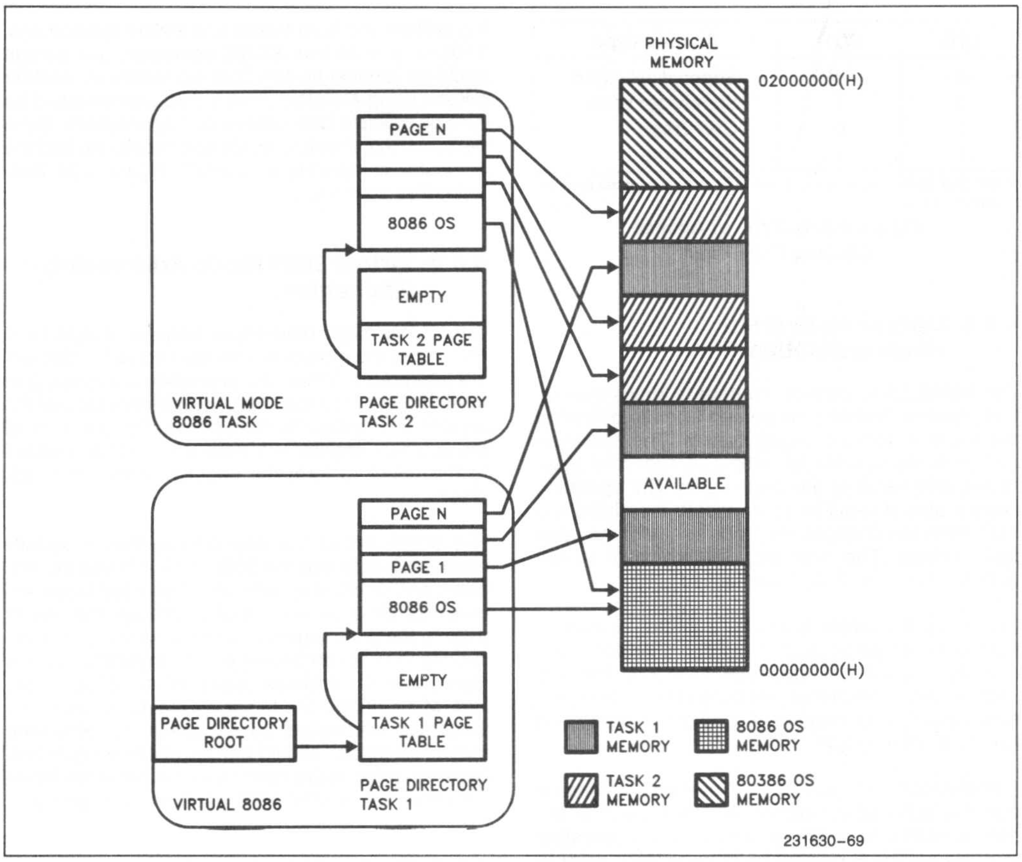


**Figure 4-24. Virtual 8086 Environment Memory Management**

Figure 4-24 shows how the 80386 paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

### 4.6.4 Protection

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 general protection fault (exception 13). Several instructions are made IOPL-sensitive, thus the operating system can decide to trap I/O instructions and emulate them by setting the IOPL = 0, or to let I/O instructions work normally by setting IOPL = 3. Since Real Mode programs are always assumed to be executing at privilege level 0 no privileged or IOPL sensitive instruction faults can be generated. However, some instructions are used only in Protected Mode, these instructions generate undefined opcodes in both Real and Virtual Modes.

The following instructions cause an exception 6 in both Real and Virtual 8086 Mode:

ARPL, LAR, LSL, VERR, VERW, STR, LTR, SLDT, and LLDT.

The following are privileged instructions. They can be used in Real Mode, but they cause a General Protection Exception (interrupt 13) in Virtual Mode or whenever the CPL > 0.

LIDT, LGDT, LMSW, CTS, HLT, MOV DRn, REG; MOV REG, DRn; MOV CRn, REG; MOV REG, CRn; MOV TRn, REG; and MOV REG, TRn.

The following instructions will generate a General Protection Exception (exception 13) when CPL > IOPL:

INS, IN, OUTS, OUT, STI, CLI, and LOCK.

The INT n, PUSHF, POPF, and IRET instructions are made IOPL sensitive only when the processor is executing in Virtual 8086 Mode. (Note that INT3 and INTO instructions are not made IOPL sensitive.)

### 4.6.5 Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 80386 operating system. The 80386 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 80386 operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 80386 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 80386 operating system. The 80386 operating system could emulate the 8086 operating system's call. Figure 4-25 shows how the 80386 operating system could intercept an 8086 operating system's call to "Open a File".

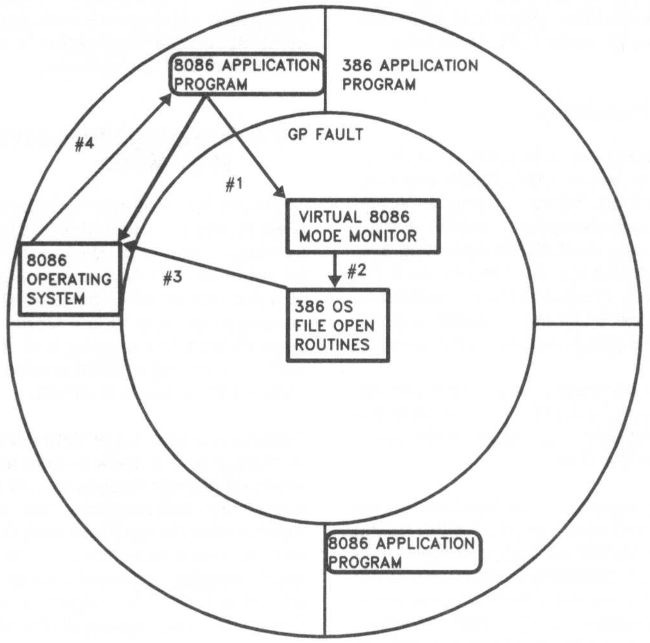
An 80386 operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

### 4.6.6 Entering and Leaving Virtual 8086 Mode

There are two methods for entering or leaving Virtual 8086 Mode. A Virtual Mode task is entered by performing a CALL or JUMP to a TSS which has the VM bit set in the EFLAGS image. Upon exiting an interrupt handler at privilege level 0 a set VM bit also causes a return to Virtual Mode. The first method is used to start the execution of a Virtual Mode task, while the second method is used to return from servicing a Virtual Mode interrupt.

Transition in and out of Virtual Mode results in a level change and a stack switch. In addition, all of the segment register images are on the stack, and then loaded with null selectors. This will permit the interrupt handlers to save and restore the segment registers as 80286 selectors, instead of 8086 style segment registers. Interrupt routines which expect values in the segment registers, will have to obtain these values by looking on the stack.

Leaving Virtual Mode is accomplished by simply JMPing to a TSS (while at privilege level 0) which does not have the VM bit set. This causes a task switch.



231630–70

8086 Application makes "Open File Call" ⟶ causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls 386 OS (Arrow #2)
386 OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

**Figure 4-25. Virtual 8086 Environment Interrupt and Call Handling**

# 5. FUNCTIONAL DATA

## 5.1 INTRODUCTION

The 80386 features a straightforward functional interface to the external hardware. The 80386 has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus outputs 32-bit address values in the most directly usable form for the high-speed local bus: 4 individual byte enable signals, and the 30 upper-order bits as a binary value. The data and address buses are interpreted and controlled with their associated control signals.

A **dynamic data bus sizing** feature allows the processor to handle a mix of 32- and 16-bit external buses on a cycle-by-cycle basis (see **5.3.4 Data Bus Sizing**). If 16-bit bus size is selected, the 80386 automatically makes any adjustment needed, even performing another 16-bit bus cycle to complete the transfer if that is necessary. 8-bit peripheral devices may be connected to 32-bit or 16-bit buses with no loss of performance. A **new address pipelining option** is provided and applies to 32-bit and 16-bit buses for substantially improved memory utilization, especially for the most heavily used memory resources.

The **address pipelining option**, when selected, typically allows a given memory interface to operate with one less wait state than would otherwise be required (see **5.4.2 Address Pipelining**). The pipelined bus is also well suited to interleaved memory designs. For 16 MHz interleaved memory designs with 100 ns access time DRAMs, zero wait states can be achieved when pipelined addressing is selected. When address pipelining is requested by the external hardware, the 80386 will output the address and bus cycle definition of the next bus cycle (if it is internally available) even while waiting for the current cycle to be acknowledged.

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

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 80386 bus cycles perform data transfer in a minimum of only two clock periods. On a 32-bit data bus, the maximum 80386 transfer bandwidth at 16 MHz is therefore 32 Mbytes/sec. Any bus cycle will be extended for more than two clock periods, however, if external hardware withholds acknowledgement of the cycle. At the appropriate time, acknowledgement is signalled by asserting the 80386 READY# input.

The 80386 can relinquish control of its local buses to allow mastership by other devices, such as direct memory access channels. When relinquished, HLDA is the only output pin driven by the 80386, providing near-complete isolation of the processor from its system. The near-complete isolation characteristic is ideal when driving the system from test equipment, and in fault-tolerant applications.

Functional data covered in this chapter describes the processor's hardware interface. First, the set of signals available at the processor pins is described (see **5.2 Signal Description**). Following that are the signal waveforms occurring during bus cycles (see **5.3 Bus Transfer Mechanism, 5.4 Bus Functional Description** and **5.5 Other Functional Descriptions**).

## 5.2 SIGNAL DESCRIPTION

### 5.2.1 Introduction

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

Example signal: M/IO# — High voltage indicates Memory selected
— Low voltage indicates I/O selected

The signal descriptions sometimes refer to AC timing parameters, such as "$t_{25}$ Reset Setup Time" and "$t_{26}$ Reset Hold Time." The values of these parameters can be found in Tables 7-4 and 7-5.

### 5.2.2 Clock (CLK2)

CLK2 provides the fundamental timing for the 80386. 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 RESET signal falling edge meets its applicable setup and hold times, $t_{25}$ and $t_{26}$.

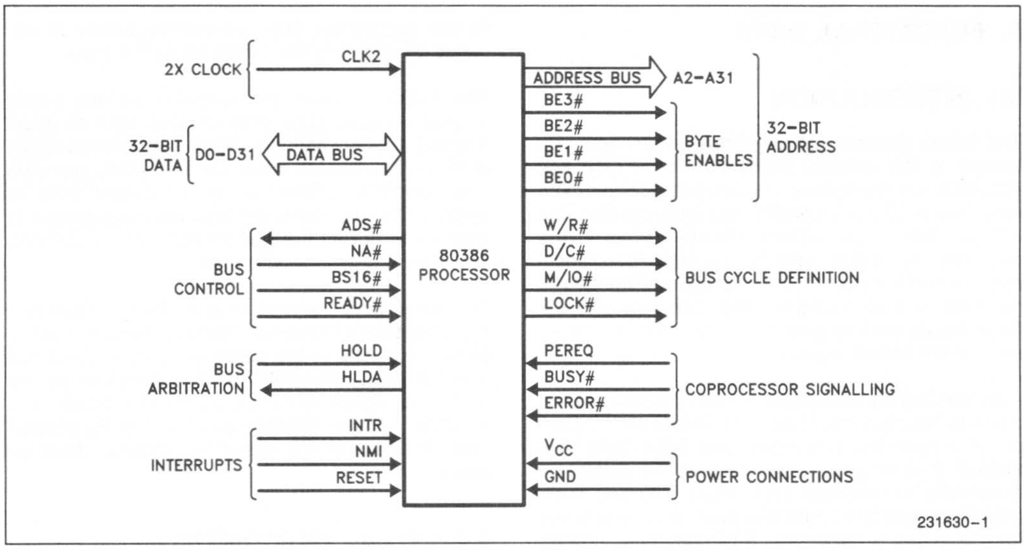


**Figure 5-1. Functional Signal Groups**

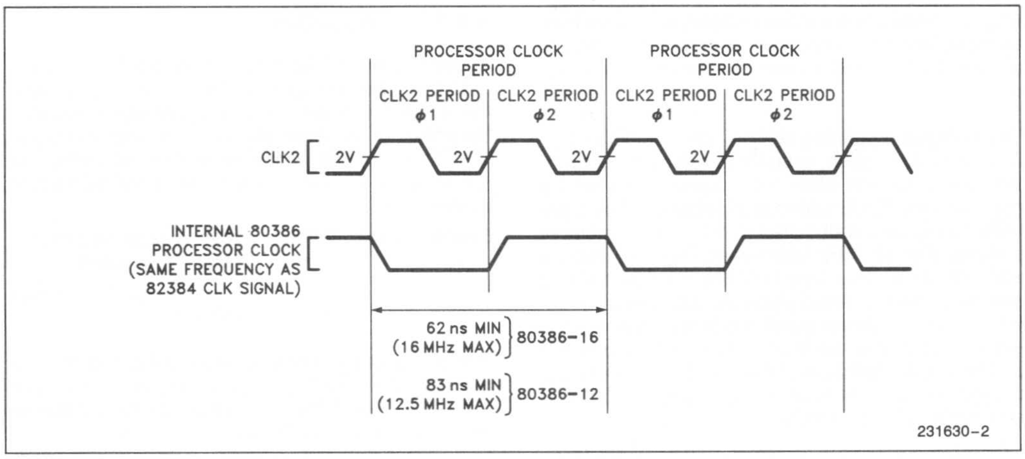


**Figure 5-2. CLK2 Signal and Internal Processor Clock**

## 5.2.3 Data Bus (D0 through D31)

These three-state bidirectional signals provide the general purpose data path between the 80386 and other devices. Data bus inputs and outputs indicate "1" when HIGH. The data bus can transfer data on 32- and 16-bit buses using a data bus sizing feature controlled by the BS16# input. See section **5.2.6 Bus Contol**. Data bus reads require that read data setup and hold times $t_{21}$ and $t_{22}$ be met for correct operation. During any write operation (and during halt cycles and shutdown cycles), the 80386 always drives all 32 signals of the data bus even if the current bus size is 16-bits.

## 5.2.4 Address Bus (BE0# through BE3#, A2 through A31)

These three-state outputs provide physical memory addresses or I/O port addresses. The address bus is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 kilobytes of I/O address space (00000000H through 0000FFFFH) for programmed I/O. I/O transfers automatically generated for 80386-to-coprocessor communication use I/O addresses 800000F8H through 800000FFH, so A31 HIGH in conjunction with M/IO# LOW allows simple generation of the coprocessor select signal.

The Byte Enable outputs, BE0#-BE3#, directly indicate which bytes of the 32-bit data bus are involved with the current transfer. This is most convenient for external hardware.

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

The number of Byte Enables asserted indicates the physical size of the operand being transferred (1, 2, 3, or 4 bytes). Refer to section **5.3.6 Operand Alignment**.

When a memory write cycle or I/O write cycle is in progress, and the operand being transferred occupies **only** the upper 16 bits of the data bus (D16-D31), duplicate data is simultaneously presented on the corresponding lower 16-bits of the data bus (D0-D15). This duplication is performed for optimum write performance on 16-bit buses. The pattern of write data duplication is a function of the Byte Enables asserted during the write cycle. Table 5-1 lists the write data present on D0-D31, as a function of the asserted Byte Enable outputs BE0#-BE3#.

### 5.2.5 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. LOCK# distinguishes between locked and unlocked bus cycles.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as the ADS# (Address Status output) is driven asserted. The LOCK# is driven valid at the same time as the bus cycle begins, which due to address pipelining, could be later than ADS# is driven asserted. See **5.4.3.4 Pipelined Address.**

Exact bus cycle definitions, as a function of W/R#, D/C#, and MI/IO#, are given in Table 5-2. Note one combination of W/R#, D/C# and M/IO# is never given when ADS# is asserted (however, that combination, which is listed as "does not occur," will occur during **idle** bus states when ADS# is **not** asserted). If M/IO#, D/C#, and W/R# are qualified by ADS# asserted, then a decoding scheme may use the non-occurring combination to its best advantage.

**Table 5-1. Write Data Duplication as a Function of BE0#-BE3#**

| 80386 Byte Enables | | | | 80386 Write Data | | | | Automatic Duplication? |
|---|---|---|---|---|---|---|---|---|
| BE3# | BE2# | BE1# | BE0# | D24-D31 | D16-D23 | D8-D15 | D0-D7 | |
| High | High | High | Low | undef | undef | undef | A | No |
| High | High | Low | High | undef | undef | B | undef | No |
| High | Low | High | High | undef | C | undef | C | Yes |
| Low | High | High | High | D | undef | D | undef | Yes |
| High | High | Low | Low | undef | undef | B | A | No |
| High | Low | Low | High | undef | C | B | undef | No |
| Low | Low | High | High | D | C | D | C | Yes |
| High | Low | Low | Low | undef | C | B | A | No |
| Low | Low | Low | High | D | C | B | undef | No |
| Low | Low | Low | Low | D | C | B | A | No |

Key:

D = logical write data d24-d31
C = logical write data d16-d23
B = logical write data d8-d15
A = logical write data d0-d7

Table 5-2. Bus Cycle Definition

| M/IO# | D/C# | W/R# | Bus Cycle Type | | Locked? |
|---|---|---|---|---|---|
| Low | Low | Low | INTERRUPT ACKNOWLEDGE | | Yes |
| Low | Low | High | does not occur | | — |
| Low | High | Low | I/O DATA READ | | No |
| Low | High | High | I/O DATA WRITE | | No |
| High | Low | Low | MEMORY CODE READ | | No |
| High | Low | High | HALT: Address = 2 (BE0# High BE1# High BE2# Low BE3# High A2–A31 Low) | SHUTDOWN: Address = 0 (BE0# Low BE1# High BE2# High BE3# High A2–A31 Low) | No |
| High | High | Low | MEMORY DATA READ | | Some Cycles |
| High | High | High | MEMORY DATA WRITE | | Some Cycles |

## 5.2.6 Bus Control Signals

### 5.2.6.1 INTRODUCTION

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

### 5.2.6.2 ADDRESS STATUS (ADS#)

This three-state output indicates that a valid bus cycle definition, and address (W/R#, D/C#, M/IO#, BE0#–BE3#, and A2–A31) is being driven at the 80386 pins. It is asserted during T1 and T2P bus states (see **5.4.3.2 Non-pipelined Address** and **5.4.3.4 Pipelined Address** for additional information on bus states).

### 5.2.6.3 TRANSFER ACKNOWLEDGE (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BE0#–BE3# and BS16# are accepted or provided. When READY# is sampled asserted during a read cycle or interrupt acknowledge cycle, the 80386 latches the input data and terminates the cycle. When READY# is sampled asserted during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY must always meet setup and hold times $t_{19}$ and $t_{20}$ for correct operation. See all sections of **5.4 Bus Functional Description**.

### 5.2.6.4 NEXT ADDRESS REQUEST (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BE0#–BE3#, A2–A31, W/R#, D/C# and M/IO# from the 80386 even if the end of the current cycle is not being acknowledged on READY#. If this input is asserted when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. See **5.4.2 Address Pipelining** and **5.4.3 Read and Write Cycles**.

### 5.2.6.5 BUS SIZE 16 (BS16#)

The BS16# feature allows the 80386 to directly connect to 32-bit and 16-bit data buses. Asserting this input constrains the current bus cycle to use only the lower-order half (D0–D15) of the data bus, corresponding to BE0# and BE1#. Asserting BS16# has no additional effect if only BE0# and/or BE1# are asserted in the current cycle. However, during bus cycles asserting BE2# or BE3#, asserting BS16# will automatically cause the 80386 to make adjustments for correct transfer of the upper bytes(s) using only physical data signals D0–D15.

If the operand spans both halves of the data bus and BS16# is asserted, the 80386 will automatically perform another 16-bit bus cycle. BS16# must always meet setup and hold times $t_{17}$ and $t_{18}$ for correct operation.

80386 I/O cycles automatically generated for coprocessor communication do not require BS16# be asserted. The coprocessor type, 80287 or 80387, is sensed on the ERROR# input shortly after the falling edge of RESET. The 80386 transfers only 16-bit quantities between itself and the 80287, but must transfer 32-bit quantities between itself and the 80387. Therefore BS16# is a don't care during 80287 cycles and must not be asserted during 80387 communication cycles.

## 5.2.7 Bus Arbitration Signals

### 5.2.7.1 INTRODUCTION

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **5.5.1 Entering and Exiting Hold Acknowledge** for additional information.

### 5.2.7.2 BUS HOLD REQUEST (HOLD)

This input indicates some device other than the 80386 requires bus mastership.

HOLD must remain asserted as long as any other device is a local bus master. HOLD is not recognized while RESET is asserted. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high impedance) state.

HOLD is level-sensitive and is a synchronous input. HOLD signals must always meet setup and hold times $t_{23}$ and $t_{24}$ for correct operation.

### 5.2.7.3 BUS HOLD ACKNOWLEDGE (HLDA)

Assertion of this output indicates the 80386 has relinquished control of its local bus in response to HOLD asserted, and is in the bus Hold Acknowledge state.

The Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 80386. The other output signals or bidirectional signals (D0–D31, BE0#–BE3#, A2–A31, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus master may control them. Pullup resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **7.2.3 Resistor Recommendations**. Also, one rising edge occuring on the NMI input during Hold Acknowledge is remembered, for processing after the HOLD input is negated.

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

## 5.2.8 Coprocessor Interface Signals

### 5.2.8.1 INTRODUCTION

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 80386 and its 80287 or 80387 processor extension.

### 5.2.8.2 COPROCESSOR REQUEST (PEREQ)

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

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

### 5.2.8.3 COPROCESSOR BUSY (BUSY#)

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

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

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

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the 80386 performs an internal self-test (see **5.5.3 Bus Activity During and Following Reset**). If BUSY# is sampled HIGH, no self-test is performed.

#### 5.2.8.4 COPROCESSOR ERROR (ERROR#)

This input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 80386 when a coprocessor instruction is encountered, and if asserted, the 80386 generates exception 7 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 80386 generating exception 7 even if ERROR# is asserted. These instructions are FNINIT, FNCLEX, FSTSW, FSTSWAX, FSTCW, FSTENV, FSAVE, FESTENV and FESAVE.

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

ERROR# serves an additional function. If ERROR# is LOW no later than 20 CLK2 periods after the falling edge of RESET and remains LOW at least until the 80386 begins its first bus cycle, an 80387 is assumed to be present (ET bit in CR0 automatically gets set to 1). Otherwise, an 80287 (or no coprocessor) is assumed to be present (ET bit in CR0 automatically is reset to 0). See **5.5.3 Bus Activity During and After Reset**. Only the ET bit is set by this ERROR# pin test. Software must set the EM and MP bits in CR0 as needed. Therefore, distinguishing 80287 presence from no coprocessor requires a software test and appropriately resetting or setting the EM bit of CR0 (set EM = 1 when no coprocessor is present). If ERROR# is sampled LOW after reset (indicating 80387) but software later sets EM = 1, the 80386 will behave as if no coprocessor is present.

### 5.2.9 Interrupt Signals

#### 5.2.9.1 INTRODUCTION

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

#### 5.2.9.2 MASKABLE INTERRUPT REQUEST (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 80386 Flag Register IF bit. When the 80386 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 D0–D7 to identify the source of the interrupt.

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

#### 5.2.9.3 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 perfomed when processing NMI.

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

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

#### 5.2.9.4 RESET (RESET)

This input signal suspends any operation in progress and places the 80386 in a known reset state. The 80386 is reset by asserting RESET for 15 or more CLK2 periods (78 or more CLK2 periods before requesting self test). When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5-3. If RESET and HOLD are both asserted at a point in time, RESET takes priority even if the 80386 was in a Hold Acknowledge state prior to RESET asserted.

RESET is level-sensitive and is allowed to be asynchronous to the CLK2 signal. If desired, the phase of the internal processor clock, and the entire 80386 state can be completely synchronized to external circuitry by ensuring the RESET signal falling edge meets its applicable setup and hold times, $t_{25}$ and $t_{26}$.

**Table 5-3. Pin State (Bus Idle) During Reset**

| Pin Name | Signal Level During Reset |
|---|---|
| ADS# | High |
| D0–D31 | High Impedance |
| BE0#–BE3# | Low |
| A2–A31 | High |
| W/R# | High |
| D/C# | High |
| M/IO# | Low |
| LOCK# | High |
| HLDA | Low |

### 5.2.10 Signal Summary

Table 5-4 summarizes the characteristics of all 80386 signals.

**Table 5-4. 80386 Signal Summary**

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

**NOTE:**
If the phase of the internal processor clock must be synchronized to external circuitry, RESET falling edge must meet setup and hold times $t_{25}$ and $t_{26}$.

## 5.3 BUS TRANSFER MECHANISM

### 5.3.1 Introduction

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and double-word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two or even three physical bus cycles are performed as required for unaligned operand transfers. See **5.3.4 Dynamic Data Bus Sizing** and **5.3.6 Operand Alignment**.

The 80386 address signals are designed to simplify external system hardware. Higher-order address bits are provided by A2–A31. Lower-order address in the form of BE0#–BE3# directly provides linear selects for the four bytes of the 32-bit data bus. Physical operand size information is thereby implicitly provided each bus cycle in the most usable form.

Byte Enable outputs BE0#–BE3# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5-5. During a bus cycle, any possible pattern of contiguous, asserted Byte Enable outputs can occur, but never patterns having a negated Byte Enable separating two or three asserted Enables.

Address bits A0 and A1 of the physical operand's base address can be created when necessary (for instance, for Multibus I or Multibus II interface), as a function of the lowest-order asserted Byte Enable. This is shown by Table 5-6. Logic to generate A0 and A1 is given by Figure 5-3.

**Table 5-5. Byte Enables and Associated Data and Operand Bytes**

| Byte Enable Signal | Associated Data Bus Signals |
|---|---|
| BE0# | D0–D7 (byte 0—least significant) |
| BE1# | D8–D15 (byte 1) |
| BE2# | D16–D23 (byte 2) |
| BE3# | D24–D31 (byte 3—most significant) |

**Table 5-6. Generating A0–A31 from BE0#–BE3# and A2–A31**

| 80386 Address Signals | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **A31** | ......... | **A2** | | | **BE3#** | **BE2#** | **BE1#** | **BE0#** |
| | **Physical Base Address** | | | | | | | |
| A31 | ......... | A2 | A1 | A0 | | | | |
| A31 | ......... | A2 | 0 | 0 | X | X | X | Low |
| A31 | ......... | A2 | 0 | 1 | X | X | Low | High |
| A31 | ......... | A2 | 1 | 0 | X | Low | High | High |
| A31 | ......... | A2 | 1 | 1 | Low | High | High | High |

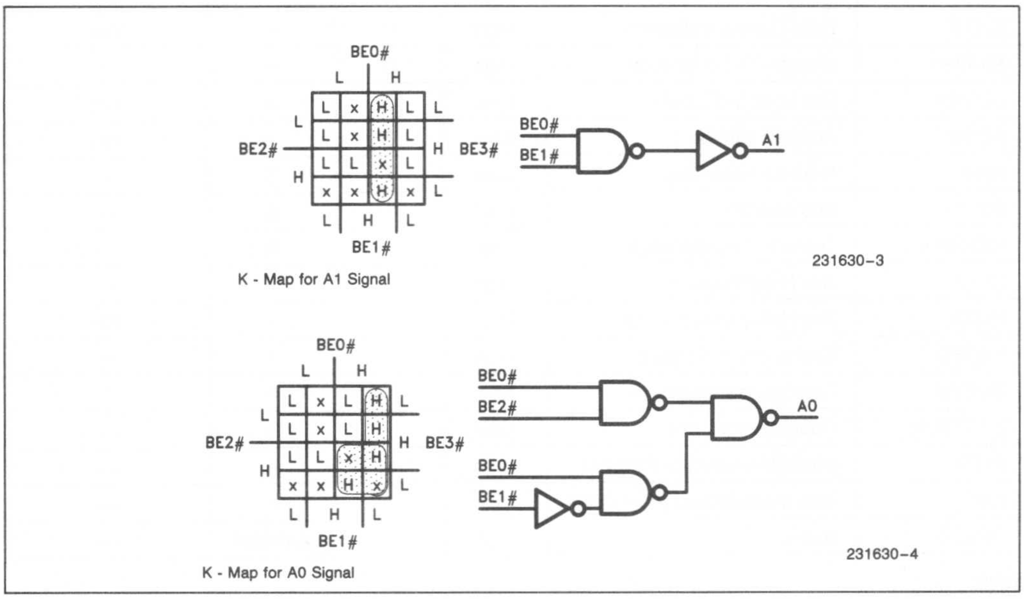


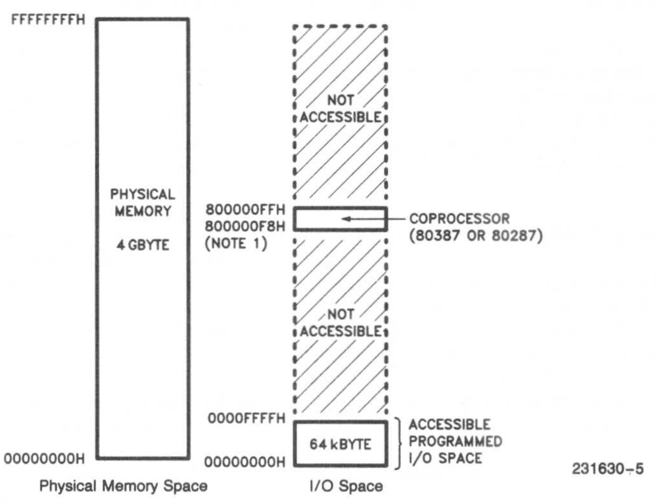
**Figure 5-3. Logic to Generate A0, A1 from BE0#–BE3#**

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 **5.4 Bus Functional Description**.

Since a bus cycle requires a minimum of two bus states (equal to two processor clock periods), data can be transferred between external devices and the 80386 at a maximum rate of one 4-byte Dword every two processor clock periods, for a maximum bus bandwidth of 32 megabytes/second (80386-16 operating at 16 MHz processor clock rate).

## 5.3.2 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 5-4, physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes) and I/O addresses from 00000000H to 0000FFFFH (64 kilobytes) for programmed I/O. Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 800000F8H to 800000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A31 and M/IO# signals.



**NOTE:**
Since A31 is HIGH during automatic communication with coprocessor, A31 HIGH and M/IO# LOW can be used to easily generate a coprocessor select signal.

**Figure 5-4. Physical Memory and I/O Spaces**

## 5.3.3 Memory and I/O Organization

The 80386 datapath to memory and I/O spaces can be 32 bits wide or 16 bits wide. When 32-bits wide, memory and I/O spaces are organized naturally as arrays of physical 32-bit Dwords. Each memory or I/O Dword has four individually addressable bytes at consecutive byte addresses. The lowest-addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31.

The 80386 includes a bus control input, BS16#, that also allows direct connection to 16-bit memory or I/O spaces organized as a sequence of 16-bit words. Cycles to 32-bit and 16-bit memory or I/O devices may occur in any sequence, since the BS16# control is sampled during each bus cycle. See **5.3.4 Dynamic Data Bus Sizing**. The Byte Enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure, whether 32 or 16 bits wide.

## 5.3.4 Dynamic Data Bus Sizing

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

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

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

There are two types of situations where the processor is affected by asserting BS16#, depending on which Byte Enables are asserted during the current bus cycle:

Upper Half Only:
Only BE2# and/or BE3# asserted.

Upper and Lower Half:
At least BE1#, BE2# asserted (and perhaps also BE0# and/or BE3#).

Effect of asserting BS16# during "upper half only" read cycles:

Asserting BS16# during "upper half only" reads causes the 80386 to read data on the lower 16 bits of the data bus and ignore data on the upper 16 bits of the data bus. Data that would have been read from D16–D31 (as indicated by BE2# and BE3#) will instead be read from D0–D15 respectively.

Effect of asserting BS16# during "upper half only" write cycles:

Asserting BS16# during "upper half only" writes does not affect the 80386. When only BE2# and/or BE3# are asserted during a write cycle the 80386 always duplicates data signals D16–D31 onto D0–D15 (see Table 5-1). Therefore, no further 80386 action is required to perform these writes on 32-bit or 16-bit buses.

Effect of asserting BS16# during "upper and lower half" read cycles:

Asserting BS16# during "upper and lower half" reads causes the processor to perform two 16-bit read cycles for complete physical operand transfer. Bytes 0 and 1 (as indicated by BE0# and BE1#) are read on the first cycle using D0–D15. Bytes 2 and 3 (as indicated by BE2# and BE3#) are read during the second cycle, again using D0–D15. D16–D31 are ignored during both 16-bit cycles. BE0# and BE1# are always negated during the second 16-bit cycle. BS16# does not have to be asserted during the second 16-bit cycle. See **Figure 5-14, cycles 2 and 2a**.

Effect of asserting BS16# during "upper and lower half" write cycles:

Asserting BS16# during "upper and lower half" writes causes the 80386 to perform two 16-bit write cycles for complete physical operand transfer. All bytes are available the first write cycle allowing external hardware to receive Bytes 0 and 1 (as indicated by BE0# and BE1#) using D0–D15. On the second cycle the 80386 duplicates Bytes 2 and 3 on D0–D15 and Bytes 2 and 3 (as indicated by BE2# and BE3#) are written using D0–D15. BE0# and BE1# are always negated during the second 16-bit cycle. BS16# does not have to be asserted during the second 16-bit cycle. See **Figure 5-14, cycles 1 and 1a**.

### 5.3.5 Interfacing with 32- and 16-Bit Memories

In 32-bit-wide physical memories such as Figure 5-5, each physical Dword begins at a byte address that is a multiple of 4. A2–A31 are directly used as a Dword select and BE0#–BE3# as byte selects. BS16# is negated for all bus cycles involving the 32-bit array.

When 16-bit-wide physical arrays are included in the system, as in Figure 5-6, each 16-bit physical word begins at a address that is a multiple of 2. Note the address is decoded, to assert BS16# only during bus cycles involving the 16-bit array. (If desiring to use

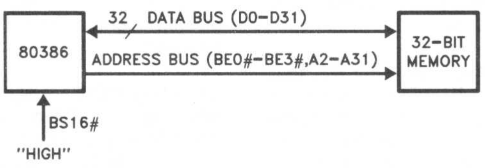


**Figure 5-5. 80386 with 32-Bit Memory**

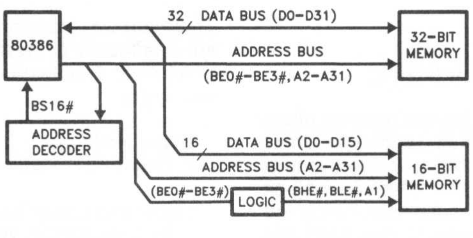


**Figure 5-6. 80386 with 32-Bit and 16-Bit Memory**

pipelined address with 16-bit memories then BE0#–BE3# and W/R# are also decoded to determine when BS16# should be asserted. See **5.4.3.7 Maximum Pipelined Address Usage with 16-Bit Bus Size.**)

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

To generate an A1 signal and two Byte Enable signals for 16-bit access, BE0#–BE3# should be decoded as in Table 5-7. Note certain combinations of BE0#–BE3# are never generated by the 80386, leading to "don't care" conditions in the decoder. Any BE0#–BE3# decoder, such as Figure 5-7, may use the non-occurring BE0#–BE3# combinations to its best advantage.

### 5.3.6 Operand Alignment

With the flexibility of memory addressing on the 80386, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dword operands beginning at addresses not evenly divisible by 4, or a 16-bit word operand split between two physical Dwords of the memory array.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 5-8 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple bus cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first (but if BS16# asserted requires two 16-bit cycles be performed, that part of the transfer is low-order first).

## 5.4 BUS FUNCTIONAL DESCRIPTION

### 5.4.1 Introduction

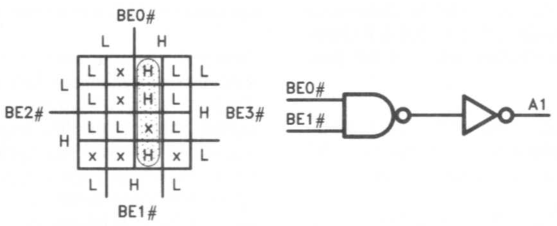
The 80386 has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus provides a 32-bit value using 30 signals for the 30 upper-order address bits and 4 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled via several associated definition or control signals.

**Table 5-7. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices**

| 80386 Signals | | | | 16-Bit Bus Signals | | | Comments |
|---|---|---|---|---|---|---|---|
| BE3# | BE2# | BE1# | BE0# | A1 | BHE# | BLE# (A0) | |
| H* | H* | H* | H* | x | x | x | x—no active bytes |
| H | H | H | L | L | H | L | |
| H | H | L | H | L | L | H | |
| H | H | L | L | L | L | L | |
| H | L | H | H | H | H | L | |
| H* | L* | H* | L* | x | x | x | x—not contiguous bytes |
| H | L | L | H | L | L | H | |
| H | L | L | L | L | L | L | |
| L | H | H | H | H | L | H | |
| L* | H* | H* | L* | x | x | x | x—not contiguous bytes |
| L* | H* | L* | H* | x | x | x | x—not contiguous bytes |
| L* | H* | L* | L* | x | x | x | x—not contiguous bytes |
| L | L | H | H | H | L | L | |
| L* | L* | H* | L* | x | x | x | x—not continguous bytes |
| L | L | L | H | L | L | H | |
| L | L | L | L | L | L | L | |

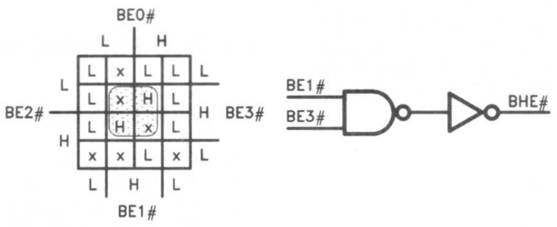
BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes



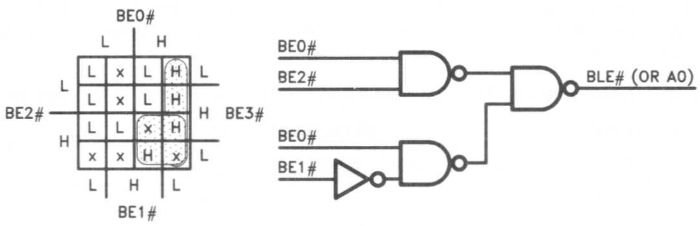
231630-8

K-map for A1 signal (same as Figure 5-3)



231630-9

K-map for 16-bit BHE# signal



231630-10

K-map for 16-bit BLE# signal (same as A0 signal in Figure 5-3)

**Figure 5-7. Logic to Generate A1, BHE# and BLE# for 16-Bit Buses**

**Table 5-8. Transfer Bus Cycles for Bytes, Words and Dwords**

| | **Byte-Length of Logical Operand** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | | | | 4 | | | |
| Physical Byte Address in Memory (low-order bits) | xx | 00 | 01 | 10 | 11 | 00 | 01 | 10 | 11 |
| Transfer Cycles over 32-Bit Data Bus | b | w | w | w | hb, lb | d | hb l3 | hw, lw | h3, lb |
| Transfer Cycles over 16-Bit Data Bus | b | w | ▨ lb, hb | w | ▨ hb, lb | ▨ lw, hw | hb, ▨ lb, mw | hw, lw | ▨ mw, hb, lb |

| | | | |
|---|---|---|---|
| Key: b = byte transfer | | 3 = 3-byte transfer | |
| w = word transfer | | d = Dword transfer | |
| l = low-order portion | | h = high-order portion | |
| m = mid-order portion | | | |
| x = don't care | | | |
| ▨ = BS16# asserted causes second bus cycle | | | |

The definition of each bus cycle is given by three definition signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals BE0#–BE3# and other address signals A2–A31. A status signal, ADS#, indicates when the 80386 issues a new bus cycle definition and address.

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

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

1) read from memory space
2) locked read from memory space
3) write to memory space
4) locked write to memory space
5) read from I/O space (or coprocessor)
6) write to I/O space (or coprocessor)
7) interrupt acknowledge
8) indicate halt, or indicate shutdown

Table 5-1 shows the encoding of the bus cycle definition signals for each bus cycle. See section **5.2.5 Bus Cycle Definition**.

The data bus has a dynamic sizing feature supporting 32- and 16-bit bus size. Data bus size is indicated to the 80386 using its Bus Size 16 (BS16#) input. All bus functions can be performed with either data bus size.

When the 80386 bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The idle state can be identified by the 80386 giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle has been terminated. The hold acknowledge state is identified by the 80386 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.

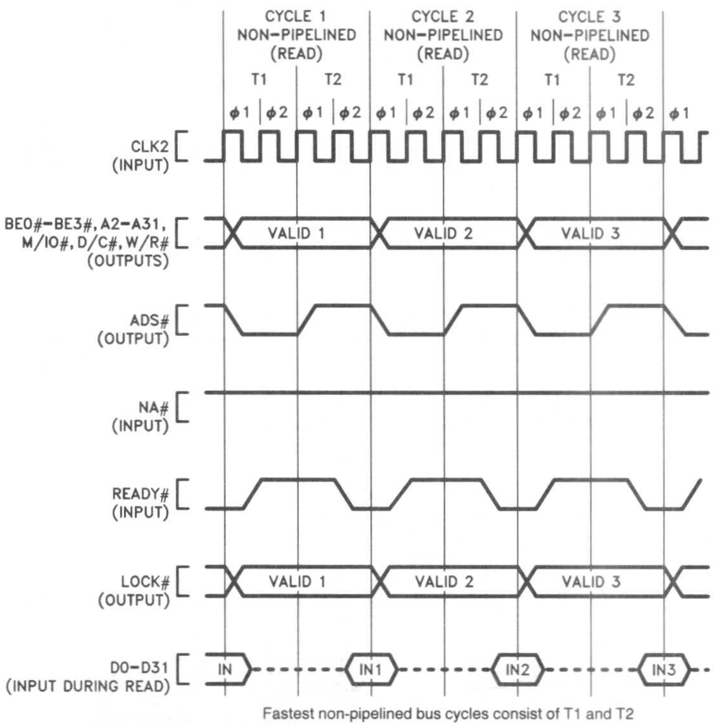

CYCLE 1 NON-PIPELINED (READ)
CYCLE 2 NON-PIPELINED (READ)
CYCLE 3 NON-PIPELINED (READ)
T1
T2
φ1
φ2
CLK2 (INPUT)
BE0#–BE3#, A2–A31, M/IO#, D/C#, W/R# (OUTPUTS)
VALID 1
VALID 2
VALID 3
ADS# (OUTPUT)
NA# (INPUT)
READY# (INPUT)
LOCK# (OUTPUT)
D0–D31 (INPUT DURING READ)
IN
IN1
IN2
IN3
Fastest non-pipelined bus cycles consist of T1 and T2
231630–11

**Figure 5-8. Fastest Read Cycles with Non-Pipelined Address Timing**

The fastest 80386 bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5-8. The bus states in each cycle are named **T1** and **T2**. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough. The high-bandwidth, two-clock bus cycle realizes the full potential of fast main memory, or cache memory.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 80386 READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If READY# is not immediately asserted however, T2 states are repeated indefinitely until the READY# input is sampled asserted.

### 5.4.2 Address Pipelining

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

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

When address pipelining is selected, the address (BE0#–BE3#, A2–A31) and definition (W/R#, D/C# and M/IO#) of the next cycle are available before the end of the current cycle. To signal their availability, the 80386 address status output (ADS#) is also asserted. Figure 5-9 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5-9 the fastest bus cycles using pipelined address require only two bus states, named **T1P** and **T2P**. Therefore cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased compared to that of a non-pipelined cycle.

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

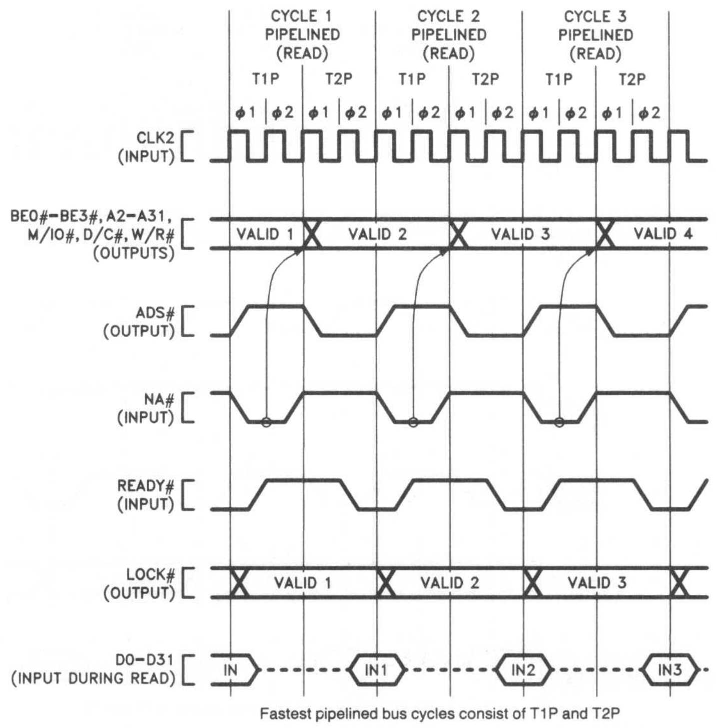


**Figure 5-9. Fastest Read Cycles with Pipelined Address Timing**

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

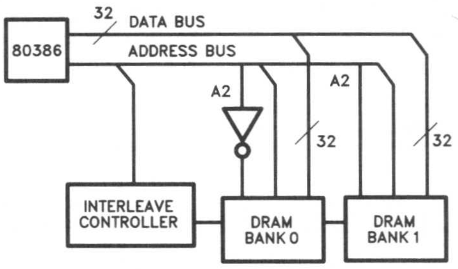
If a system contains a memory structure of two or more interleaved memory banks, pipelined address timing potentially allows even more overlap of activity. This is true when the interleaved memory controller is designed to allow the next memory operation to begin in one memory bank while the current bus cycle is still activating another memory bank. Figure 5-10 shows the general structure of the 80386 with 2-bank and 4-bank interleaved memory. Note each memory bank of the interleaved memory has full data bus width (32-bit data width typically, unless 16-bit bus size is selected).

Further details of pipelined address timing are given in **5.4.3.4 Pipelined Address, 5.4.3.5 Initiating and Maintaining Pipelined Address, 5.4.3.6 Pipelined Address with Dynamic Bus Sizing,** and **5.4.3.7 Maximum Pipelined Address Usage with 16-Bit Bus Size.**

**TWO-BANK INTERLEAVED MEMORY**

a) Address signal A2 selects bank

b) 32-bit datapath to each bank

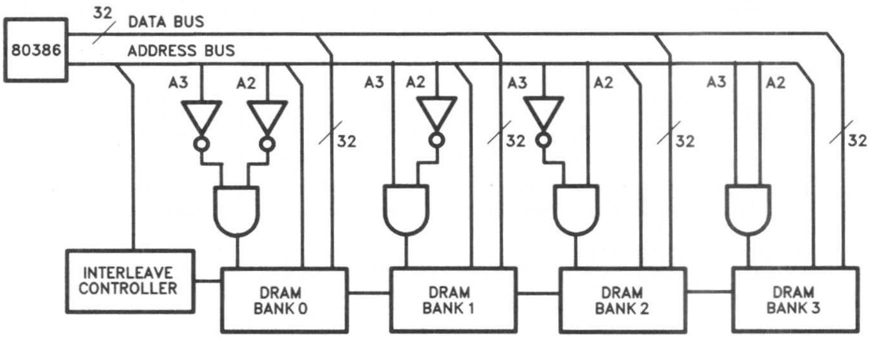


231630–13

**FOUR-BANK INTERLEAVED MEMORY**

a) Address signals A3 and A2 select bank

b) 32-bit datapath to each bank



231630–14

**Figure 5-10. 2-Bank and 4-Bank Interleaved Memory Structure**

## 5.4.3 Read and Write Cycles

### 5.4.3.1 INTRODUCTION

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

Two choices of address timing are dynamically selectable: non-pipelined, or pipelined. After a bus idle state, the processor always uses non-pipelined address timing. However, the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the 80386 has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#. Generally, the NA# input is sampled each bus cycle to select the desired address timing for the next bus cycle.

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

If 16-bit bus size is indicated, the 80386 automatically responds as required to complete the transfer on a 16-bit data bus. Depending on the size and alignment of the operand, another 16-bit bus cycle may be required. Table 5-7 provides all details. When necessary, the 80386 performs an additional 16-bit bus cycle, using D0–D15 in place of D16–D31.

Terminating a read cycle 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.

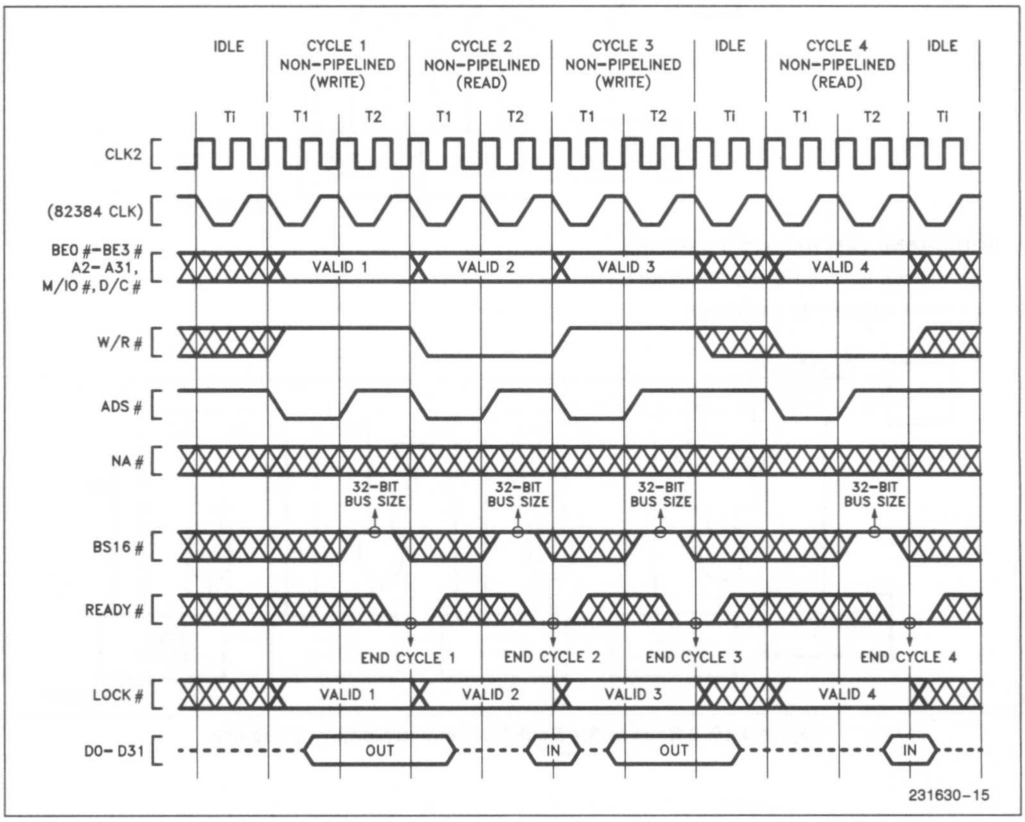


Figure 5-11. Bus Cycles with Non-Pipelined Address (zero wait states)

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 5-11. If READY# is negated as in Figure 5-12, the cycle continues 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 80386 terminates it. When a read cycle is acknowledged, the 80386 latches the information present at its data pins. When a write cycle is acknowledged, the 80386 write data remains valid throughout phase one of the next bus state, to provide write data hold time.

#### 5.4.3.2 NON-PIPELINED ADDRESS

Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5-11 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5-11 shows the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of the T1, the address signals and bus cycle definition signals are driven valid, and to signal their availability, address strobe (ADS#) is simultaneously asserted.

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

Figure 5-12 illustrates non-pipelined bus cycles with one wait added to cycles 2 and 3. READY# is sampled negated at the end of the first T2 in cycles 2 and 3. Therefore cycles 2 and 3 have T2 repeated. At the end of the second T2, READY# is sampled asserted.

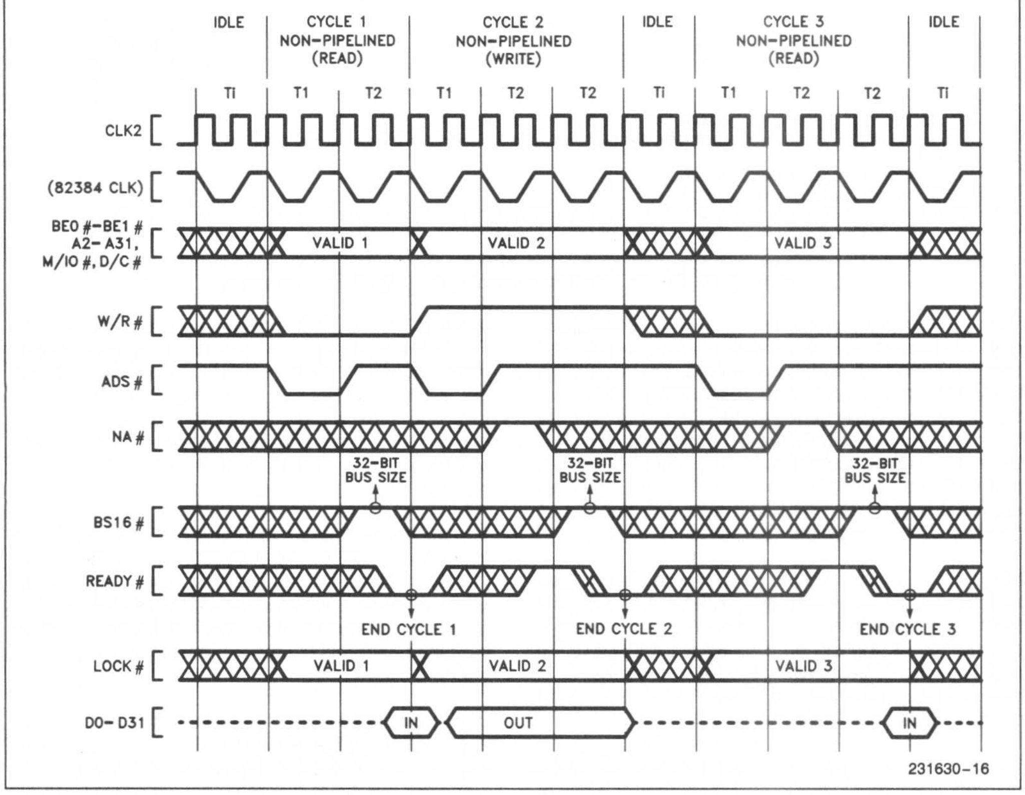
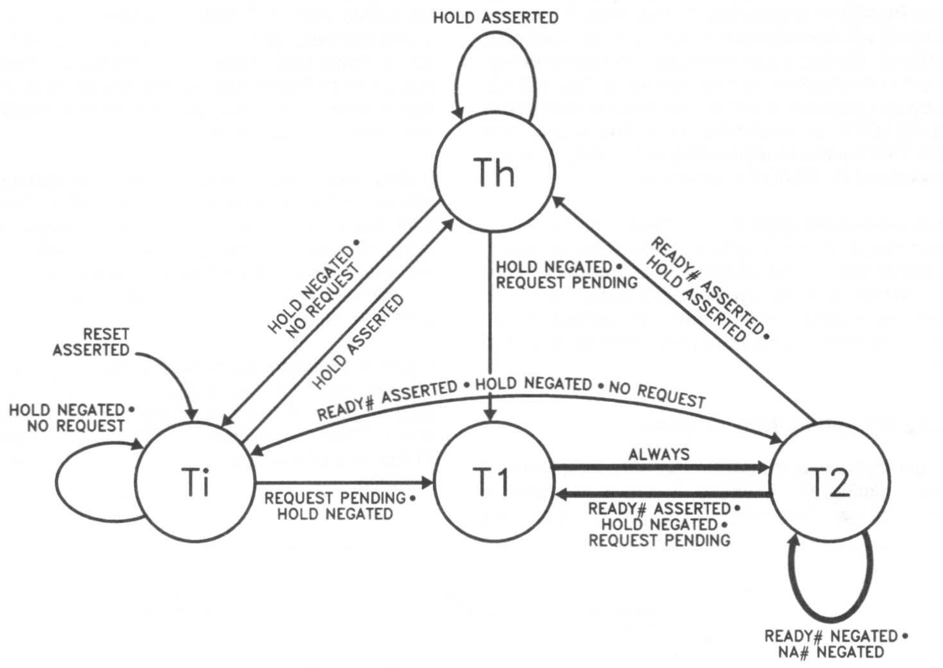


**Figure 5-12. Bus Cycles with Non-Pipelined Address (various number of wait states)**


HOLD ASSERTED
Th
HOLD NEGATED • NO REQUEST
HOLD ASSERTED
HOLD NEGATED • REQUEST PENDING
READY# ASSERTED • HOLD ASSERTED •
RESET ASSERTED
HOLD NEGATED • NO REQUEST
READY# ASSERTED • HOLD NEGATED • NO REQUEST
Ti
T1
T2
ALWAYS
REQUEST PENDING • HOLD NEGATED
READY# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# NEGATED • NA# NEGATED

231630-17

**Bus States:**
T1—first clock of a non-pipelined bus cycle (80386 drives new address and asserts ADS#)
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
Ti— idle state
Th—hold acknowledge state (80386 asserts HLDA)
The fastest bus cycle consists of two states: T1 and T2.
Four basic bus states describe bus operation when not using pipelined address. These states do include BS16# usage for 32-bit and 16-bit bus size. If asserting BS16# requires a second 16-bit bus cycle to be performed, it is performed before HOLD asserted is acknowledged.

**Figure 5-13. 80386 Bus States (not using pipelined address)**

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and you desire to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5-12 cycles 2 and 3. If NA# is sampled asserted during a T2 other than the last one, the next state would be T2I (for pipelined address) or T2P (for pipelined address) instead of another T2 (for non-pipelined address).

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5-13. The bus transitions between four possible states: T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise, the bus may be idle, in the Ti state, or in hold acknowledge, the Th state.

When address pipelining is not used, the bus state diagram is as shown in Figure 5-13. When the bus is idle it is in state Ti. Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is negated, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or Ti if there is no bus request pending, or Th if the HOLD input is being asserted.

The bus state diagram in Figure 5-13 also applies to the use of BS16#. If the 80386 makes internal adjustments for 16-bit bus size, the adjustments do not affect the external bus states. If an additional 16-bit bus cycle is required to complete a transfer on a 16-bit bus, it also follows the state transitions shown in Figure 5-13.

Use of pipelined address allows the 80386 to enter three additional bus states not shown in Figure 5-13. Figure 5-20 in **5.4.3.4 Pipelined Address** is the complete bus state diagram, including pipelined address cycles.

### 5.4.3.3 NON-PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

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

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

When a second 16-bit bus cycle is required to complete the transfer over a 16-bit bus, the addresses

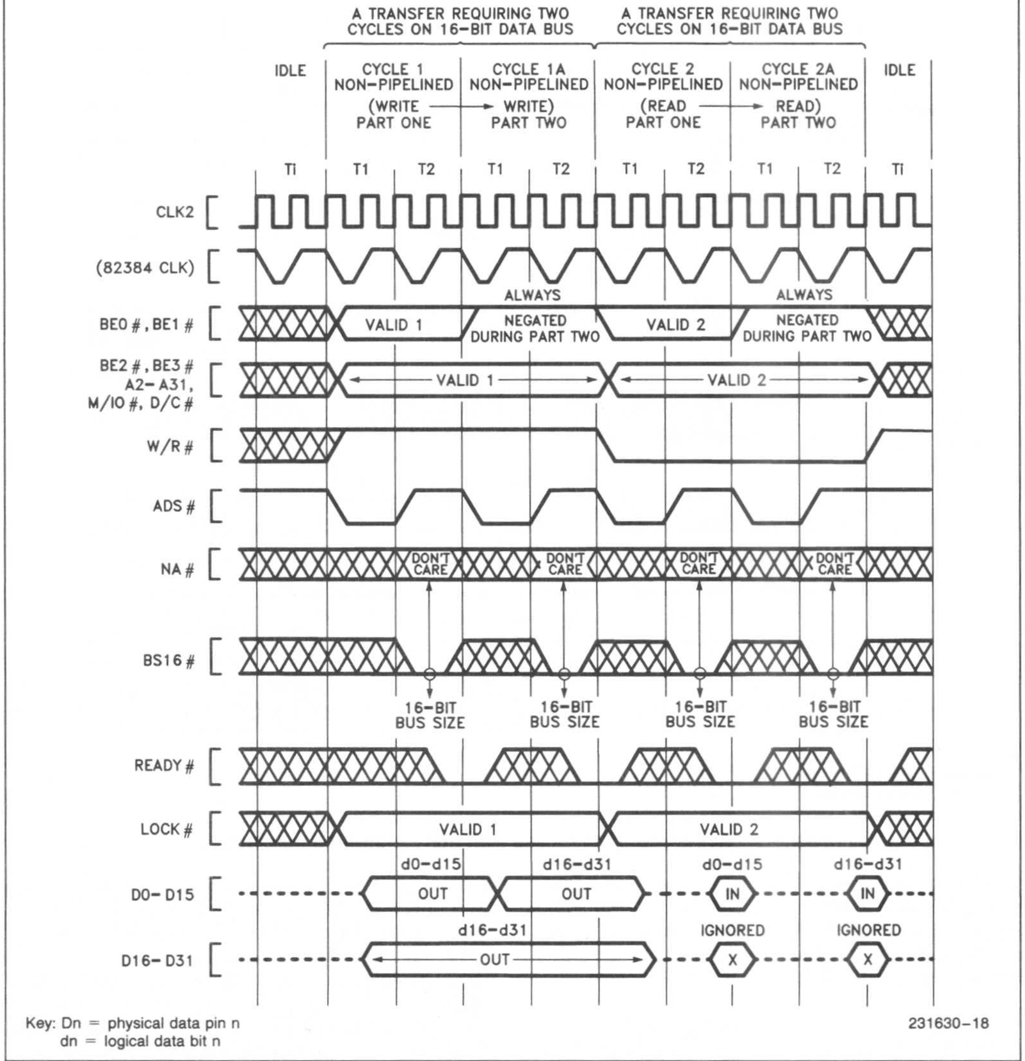
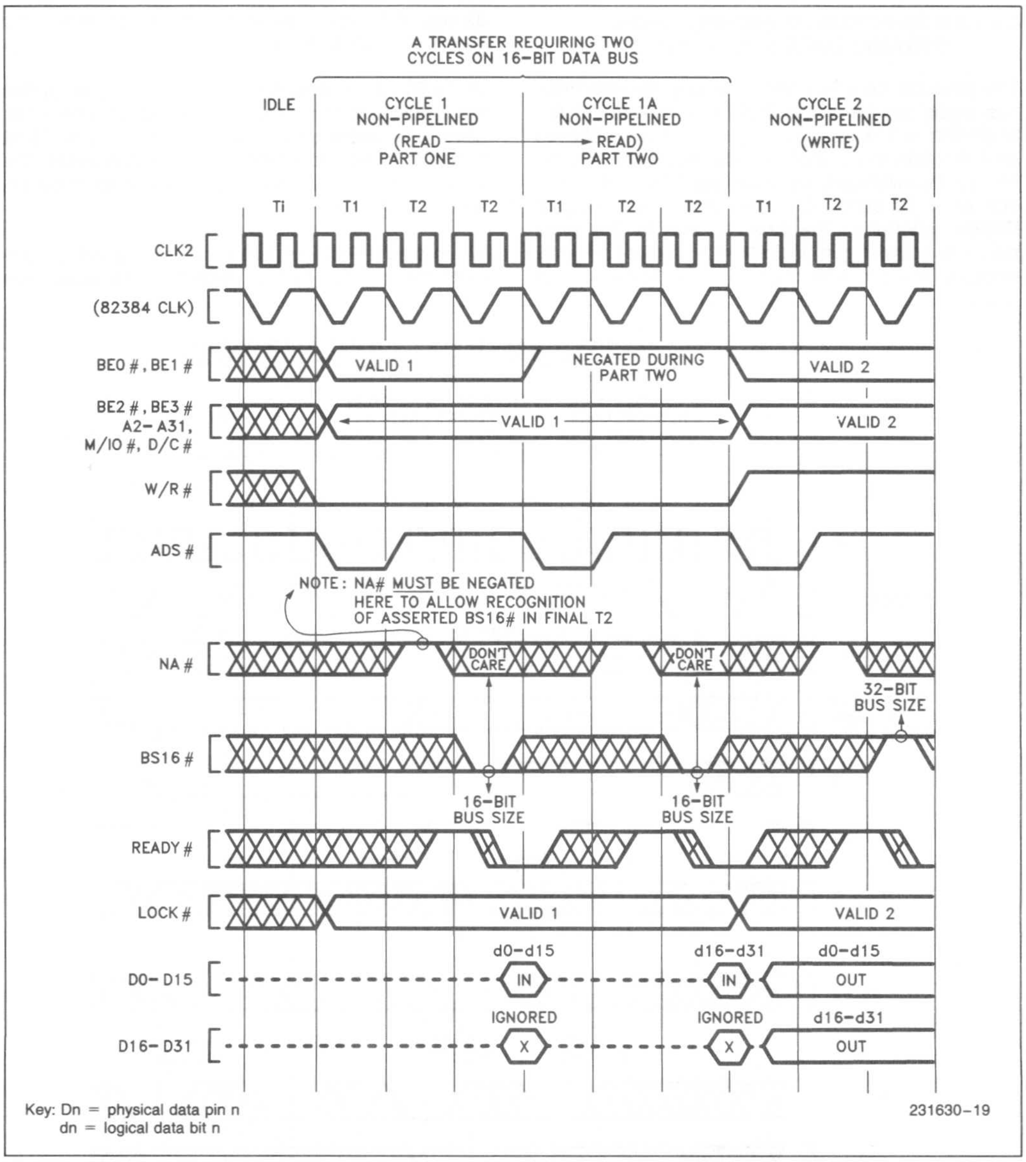

A TRANSFER REQUIRING TWO CYCLES ON 16-BIT DATA BUS
A TRANSFER REQUIRING TWO CYCLES ON 16-BIT DATA BUS
IDLE
CYCLE 1 NON-PIPELINED (WRITE PART ONE
CYCLE 1A NON-PIPELINED WRITE) PART TWO
CYCLE 2 NON-PIPELINED (READ PART ONE
CYCLE 2A NON-PIPELINED READ) PART TWO
IDLE
Ti T1 T2 T1 T2 T1 T2 T1 T2 Ti
CLK2
(82384 CLK)
BE0#, BE1#
VALID 1
ALWAYS NEGATED DURING PART TWO
VALID 2
ALWAYS NEGATED DURING PART TWO
BE2#, BE3# A2-A31, M/IO#, D/C#
VALID 1
VALID 2
W/R#
ADS#
NA#
DON'T CARE
BS16#
16-BIT BUS SIZE
READY#
LOCK#
VALID 1
VALID 2
D0-D15
d0-d15
OUT
d16-d31
OUT
d0-d15
IN
d16-d31
IN
D16-D31
d16-d31
OUT
IGNORED
X
Key: Dn = physical data pin n
dn = logical data bit n
231630-18

**Figure 5-14. Asserting BS16# (zero wait states, non-pipelined address)**



A TRANSFER REQUIRING TWO CYCLES ON 16-BIT DATA BUS
IDLE
CYCLE 1 NON-PIPELINED (READ PART ONE
CYCLE 1A NON-PIPELINED READ) PART TWO
CYCLE 2 NON-PIPELINED (WRITE)
Ti T1 T2 T2 T1 T2 T2 T1 T2 T2
CLK2
(82384 CLK)
BE0#, BE1#
VALID 1
NEGATED DURING PART TWO
VALID 2
BE2#, BE3# A2-A31, M/IO#, D/C#
VALID 1
VALID 2
W/R#
ADS#
NOTE: NA# MUST BE NEGATED HERE TO ALLOW RECOGNITION OF ASSERTED BS16# IN FINAL T2
NA#
DON'T CARE
DON'T CARE
32-BIT BUS SIZE
BS16#
16-BIT BUS SIZE
16-BIT BUS SIZE
READY#
LOCK#
VALID 1
VALID 2
D0-D15
d0-d15
IN
d16-d31
IN
d0-d15
OUT
D16-D31
IGNORED
X
IGNORED
X
d16-d31
OUT
Key: Dn = physical data pin n
dn = logical data bit n
231630-19

**Figure 5-15. Asserting BS16# (one wait state, non-pipelined address)**

generated for the two 16-bit bus cycles are closely related to each other. The addresses are the same except BE0# and BE1# are always negated for the second cycle. This is because data on D0–D15 was already transferred during the first 16-bit cycle.

Figures 5-14 and 5-15 show cases where assertion of BS16# requires a second 16-bit cycle for complete operand transfer. Figure 5-14 illustrates cycles without wait states. Figure 5-15 illustrates cycles with one wait state. In Figure 5-15 cycle 1, the bus cycle during which BS16# is asserted, note that NA# must be negated in the T2 state(s) prior to the last T2 state. This is to allow the recognition of BS16# asserted in the final T2 state. The relation of NA# and BS16# is given fully in **5.4.3.4 Pipelined Address**, but Figure 5-15 illustrates this only precaution you need to know when using BS16# with non-pipelined address.

### 5.4.3.4 PIPELINED ADDRESS

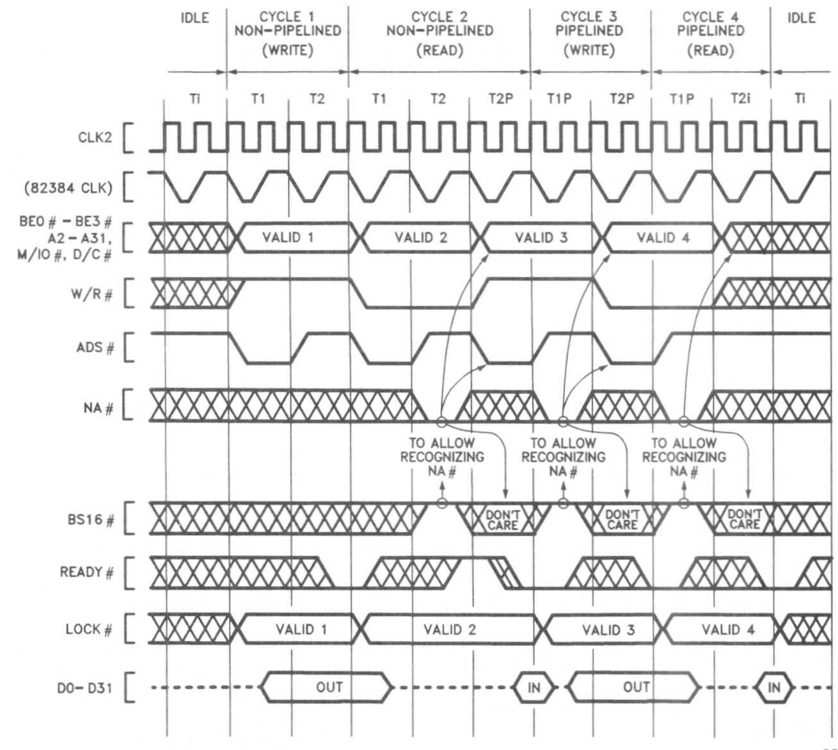
Address pipelining is the option of requesting the address and the bus cycle definition of the next, internally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the 80386 when the next address is issued. The address pipelining option is controlled 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, therefore, NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5-16, 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).

If NA# is sampled asserted, the 80386 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 80386 has the following characteristics:

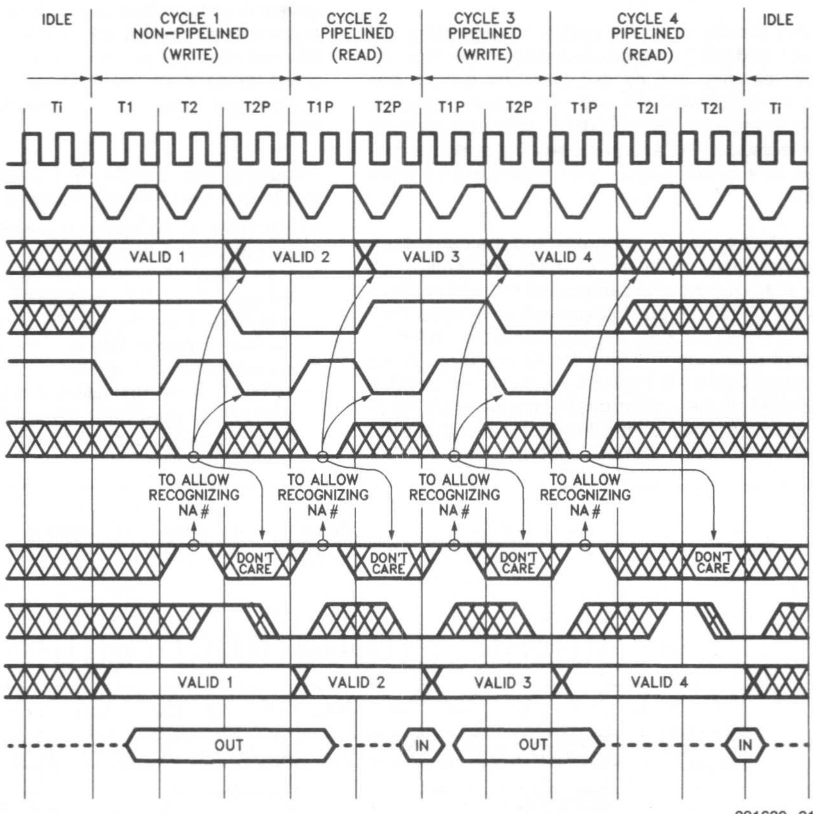
1) For NA# to be sampled asserted, BS16# must be negated at that sampling window (see Figure 5-16 Cycles 3 and 4, and Figure 5-17 Cycles 2 through 4). If NA# and BS16# are both sampled asserted during the last T2 period of a bus cycle, BS16# asserted has priority. Therefore, if both are asserted, the current bus size is taken to be 16 bits and the next address is not pipelined. Conceptually, Figure 5-18 shows the internal 80386 logic providing these characteristics.



231630–20

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

**Figure 5-16. Transitioning to Pipelined Address During Burst of Bus Cycles**



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

**Figure 5-17. Fastest Transition to Pipelined Address Following Idle Bus State**

2) The next address may appear as early as the bus state after NA# was sampled asserted (see Figures 5-16 or 5-17). 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 5-19 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 80386 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.

3) Once NA# is sampled asserted, the 80386 commits itself to the highest priority bus request that is pending internally. It can no longer perform another 16-bit transfer to the same address should BS16# be asserted externally, so thereafter must assume the current bus size is 32 bits. Therefore if NA# is sampled asserted within a bus cycle, BS16# is ignored thereafter in that bus cycle (see Figures 5-16, 5-17, 5-19). Consequently, do not assert NA# during bus cycles which must have BS16# driven asserted. See **5.4.3.6 Dynamic Bus Sizing with Pipelined Address.**

4) Any address which is validated by a pulse on the 80386 ADS# output will remain stable on the address pins for at least two processor clock periods. The 80386 cannot produce a new address more frequently than every two processor clock periods (see Figures 5-16, 5-17, 5-19).

5) Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 5-19 Cycle 1).

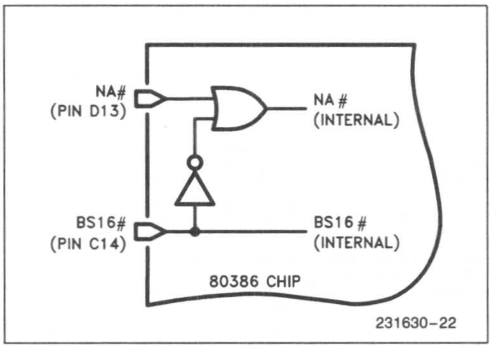


**Figure 5-18. 80386 Internal Logic on NA# and BS16#**

The complete bus state transition diagram, including operation with pipelined address is given by 5-20. 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.

#### 5.4.3.5 INITIATING AND MAINTAINING PIPELINED ADDRESS

Using the state diagram Figure 5-20, observe the transitions from an idle state, Ti, to the beginning of a pipelined bus cycle, T1P. From an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided 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:

$$\underbrace{\text{Ti, Ti, Ti,}}_{\substack{\text{idle}\\\text{states}}}\ \underbrace{\text{T1 - T2 - T2P,}}_{\substack{\text{non-pipelined}\\\text{cycle}}}\ \underbrace{\text{T1P - T2P,}}_{\substack{\text{pipelined}\\\text{cycle}}}$$

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

$$\underbrace{\text{Th, Th, Th,}}_{\substack{\text{hold}\\\text{acknowledge}\\\text{states}}}\ \underbrace{\text{T1 - T2 - T2P,}}_{\substack{\text{non-pipelined}\\\text{cycle}}}\ \underbrace{\text{T1P - T2P,}}_{\substack{\text{pipelined}\\\text{cycle}}}$$

The transition to pipelined address is shown functionally by Figure 5-17 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. During Figure 5-17 Cycle 1 therefore, sampling begins in T2. Once NA# is sampled asserted during the current cycle, the 80386 is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5-16 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.

Example transition bus cycles are Figure 5-17 Cycle 1 and Figure 5-16 Cycle 2. Figure 5-17 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5-16 Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (you assert NA# at that time), and T2P (provided the 80386 has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note three states (T1, T2 and T2P) are only required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5-17 Cycle 1. Figure 5-17 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 80386 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 5-16 and 5-17 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 80386 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.

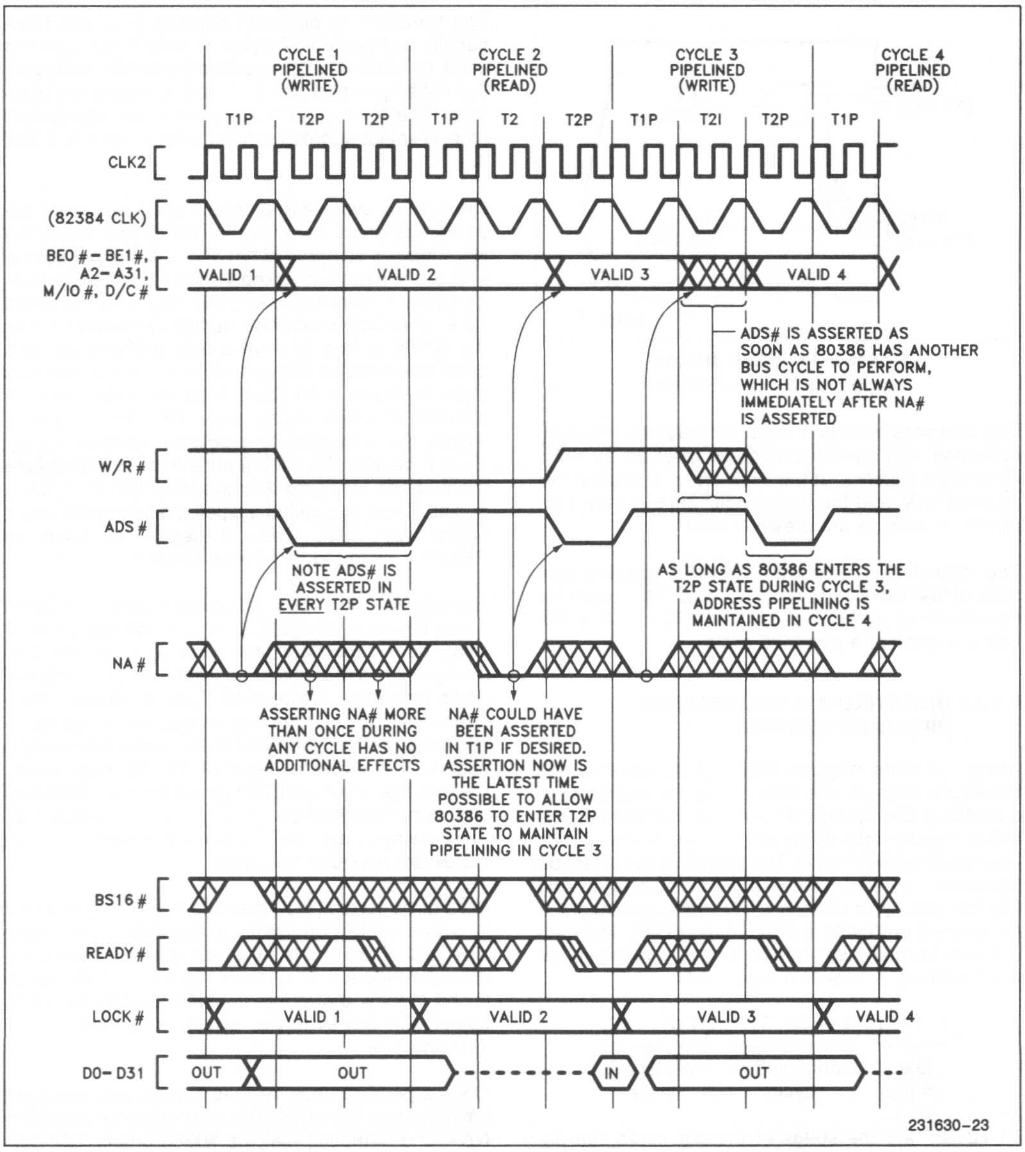
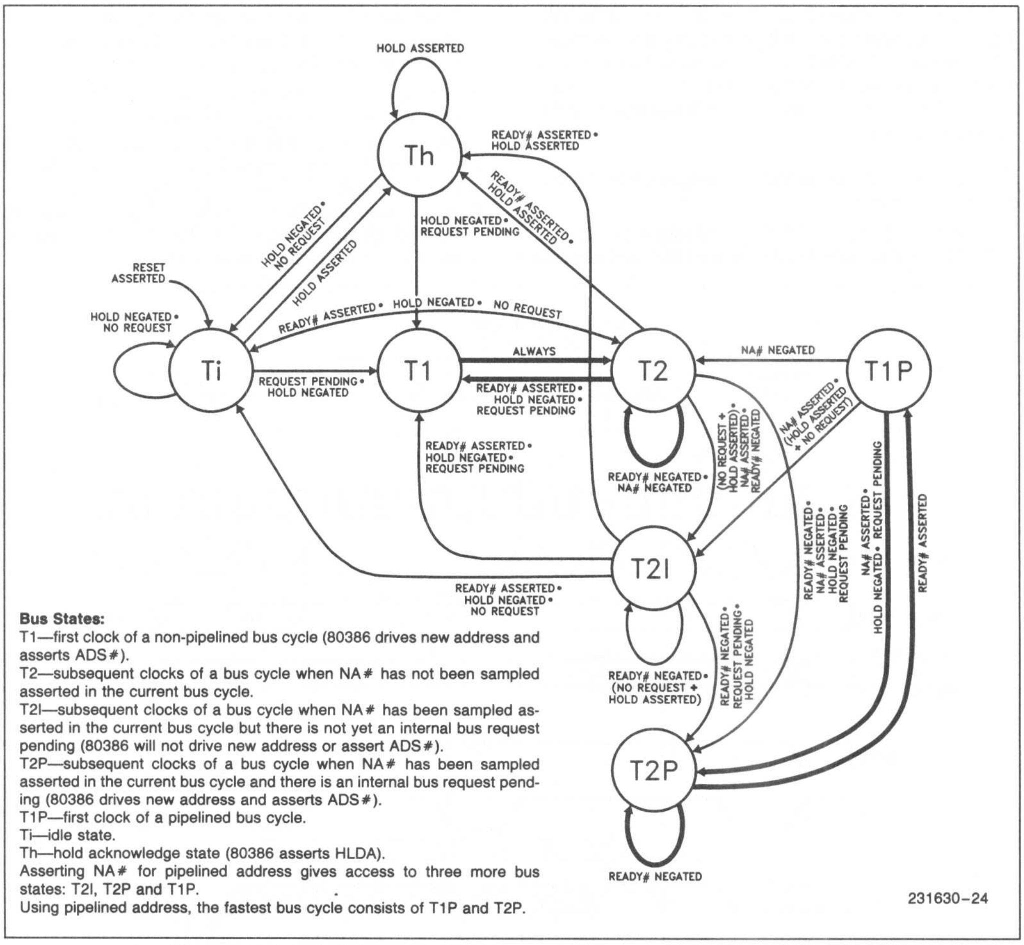

CYCLE 1 PIPELINED (WRITE)
CYCLE 2 PIPELINED (READ)
CYCLE 3 PIPELINED (WRITE)
CYCLE 4 PIPELINED (READ)
T1P T2P T2P T1P T2 T2P T1P T2I T2P T1P
CLK2
(82384 CLK)
BE0#–BE1#, A2–A31, M/IO#, D/C#
VALID 1 VALID 2 VALID 3 VALID 4
ADS# IS ASSERTED AS SOON AS 80386 HAS ANOTHER BUS CYCLE TO PERFORM, WHICH IS NOT ALWAYS IMMEDIATELY AFTER NA# IS ASSERTED
W/R#
ADS#
NOTE ADS# IS ASSERTED IN EVERY T2P STATE
AS LONG AS 80386 ENTERS THE T2P STATE DURING CYCLE 3, ADDRESS PIPELINING IS MAINTAINED IN CYCLE 4
NA#
ASSERTING NA# MORE THAN ONCE DURING ANY CYCLE HAS NO ADDITIONAL EFFECTS
NA# COULD HAVE BEEN ASSERTED IN T1P IF DESIRED. ASSERTION NOW IS THE LATEST TIME POSSIBLE TO ALLOW 80386 TO ENTER T2P STATE TO MAINTAIN PIPELINING IN CYCLE 3
BS16#
READY#
LOCK#
VALID 1 VALID 2 VALID 3 VALID 4
D0–D31
OUT OUT IN OUT
231630-23

**Figure 5-19. Details of Address Pipelining During Cycles with Wait States**


HOLD ASSERTED
Th
READY# ASSERTED • HOLD ASSERTED
READY# ASSERTED • HOLD ASSERTED •
HOLD NEGATED • NO REQUEST
HOLD ASSERTED
HOLD NEGATED • REQUEST PENDING
RESET ASSERTED
HOLD NEGATED • NO REQUEST
READY# ASSERTED • HOLD NEGATED • NO REQUEST
Ti
T1
T2
T1P
REQUEST PENDING • HOLD NEGATED
ALWAYS
READY# ASSERTED • HOLD NEGATED • REQUEST PENDING
NA# NEGATED
READY# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# NEGATED • NA# NEGATED
(NO REQUEST + HOLD ASSERTED) • NA# ASSERTED • READY# NEGATED
NA# ASSERTED • (HOLD ASSERTED + NO REQUEST)
READY# NEGATED • NA# ASSERTED • HOLD NEGATED • REQUEST PENDING
NA# ASSERTED • HOLD NEGATED • REQUEST PENDING
READY# ASSERTED
T2I
READY# ASSERTED • HOLD NEGATED • NO REQUEST
READY# NEGATED • (NO REQUEST + HOLD ASSERTED)
READY# NEGATED • REQUEST PENDING • HOLD NEGATED
T2P
READY# NEGATED
231630–24

**Bus States:**
T1—first clock of a non-pipelined bus cycle (80386 drives new address and asserts ADS#).
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle.
T2I—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (80386 will not drive new address or assert ADS#).
T2P—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (80386 drives new address and asserts ADS#).
T1P—first clock of a pipelined bus cycle.
Ti—idle state.
Th—hold acknowledge state (80386 asserts HLDA).
Asserting NA# for pipelined address gives access to three more bus states: T2I, T2P and T1P.
Using pipelined address, the fastest bus cycle consists of T1P and T2P.

**Figure 5-20. 80386 Complete Bus States (including pipelined address)**

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 and NA# is sampled asserted in each of the bus cycles.

### 5.4.3.6 PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

The BS16# feature allows easy interface to 16-bit data buses. When asserted, the 80386 bus interface hardware performs appropriate action to make the transfer using a 16-bit data bus connected on D0–D15.

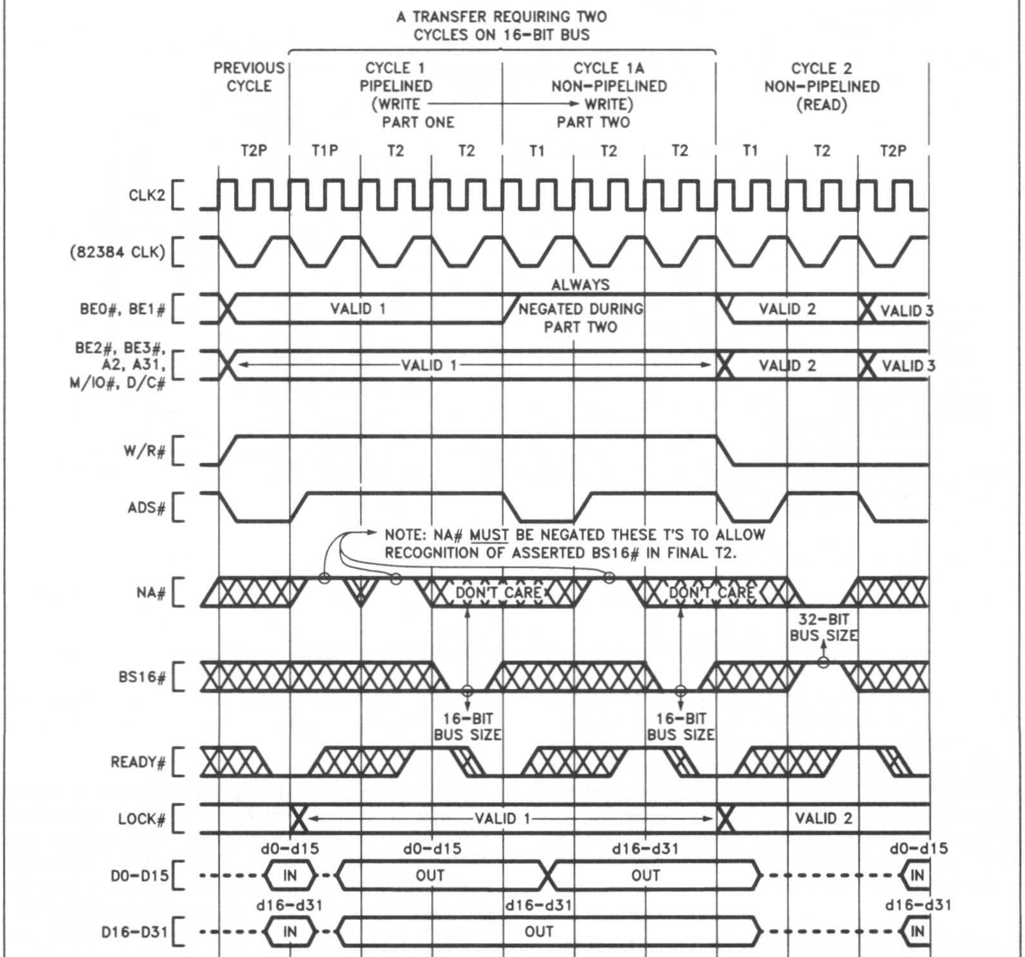
There is a degree of interaction, however, between the use of Address Pipelining and the use of Bus Size 16. The interaction results from the multiple bus cycles required when transferring 32-bit operands over a 16-bit bus. If the operand requires both 16-bit halves of the 32-bit bus, the appropriate 80386 action is a second bus cycle to complete the operand's transfer. It is this necessity that conflicts with NA# usage.

When NA# is sampled asserted, the 80386 commits itself to perform the next internally pending bus request, and is allowed to drive the next internally pending address onto the bus. Asserting NA# therefore makes it impossible for the next bus cycle to again access the current address on A2–A31, such as may be required when BS16# is asserted by the external hardware.

To avoid conflict, the 80386 is designed with following two provisions:

1) To avoid conflict, the 80386 is designed to ignore BS16# in the current bus cycle if NA# has already been sampled asserted in the current cycle. If NA# is sampled asserted, the current data bus size is assumed to be 32 bits.

2) To also avoid conflict, if NA# and BS16# are both asserted during the same sampling window, BS16# asserted has priority and the 80386 acts as if NA# was negated at that time. Internal 80386 circuitry, shown conceptually in Figure 5-18, assures that BS16# is sampled asserted and NA# is sampled negated if both inputs are externally asserted at the same sampling window.



A TRANSFER REQUIRING TWO CYCLES ON 16-BIT BUS
PREVIOUS CYCLE
CYCLE 1 PIPELINED (WRITE PART ONE
CYCLE 1A NON-PIPELINED WRITE) PART TWO
CYCLE 2 NON-PIPELINED (READ)
T2P T1P T2 T2 T1 T2 T2 T1 T2 T2P
CLK2
(82384 CLK)
BE0#, BE1#
VALID 1
ALWAYS NEGATED DURING PART TWO
VALID 2
VALID 3
BE2#, BE3#, A2, A31, M/IO#, D/C#
VALID 1
VALID 2
VALID 3
W/R#
ADS#
NOTE: NA# MUST BE NEGATED THESE T'S TO ALLOW RECOGNITION OF ASSERTED BS16# IN FINAL T2.
NA#
DON'T CARE
DON'T CARE
32-BIT BUS SIZE
BS16#
16-BIT BUS SIZE
16-BIT BUS SIZE
READY#
LOCK#
VALID 1
VALID 2
D0-D15
d0-d15 IN
d0-d15 OUT
d16-d31 OUT
d0-d15 IN
D16-D31
d16-d31 IN
d16-d31 OUT
d16-d31 IN

Key: Dn = physical data pin n
dn = logical data bit n
Cycles 1 and 2 are pipelined. Cycle 1a cannot be pipelined, but its address can be inferred from that of Cycle 1, to externally simulate address pipelining during Cycle 1a.

231630–25

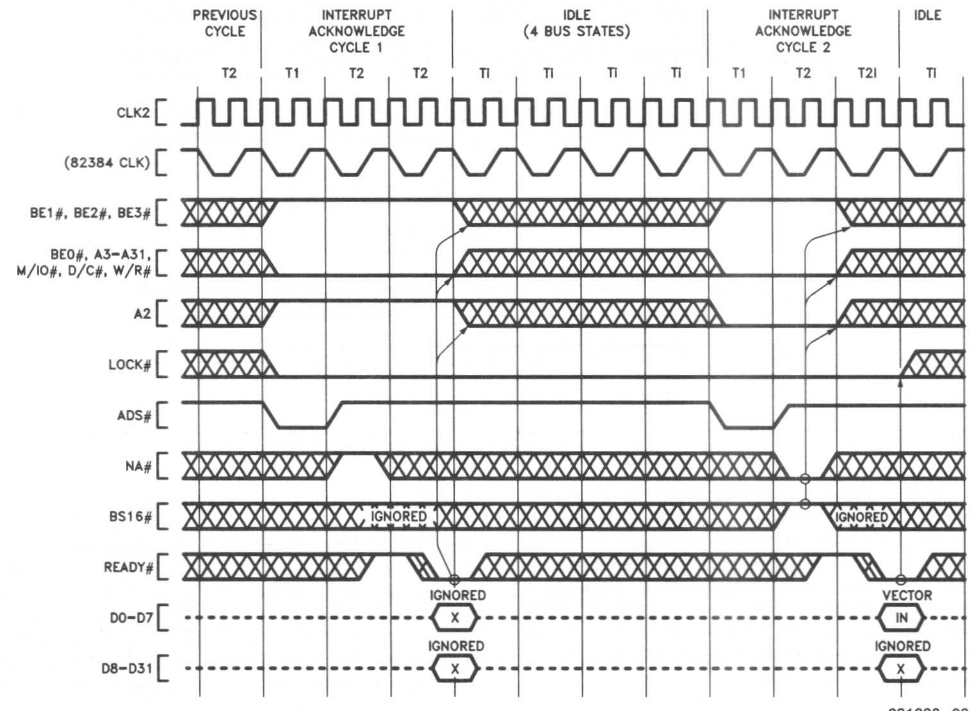
**Figure 5-21. Using NA# and BS16#**

Certain types of operands require no adjustment for correct transfer on a 16-bit bus. Those are read or write operands using only the lower half of the data bus, and write operands using only the upper half of the bus since the 80386 simultaneously duplicates the write data on the lower half of the data bus. For these patterns of Byte Enables and the R/W# signals, BS16# need not be asserted at the 80386, allowing NA# to be asserted during the bus cycle if desired.

### 5.4.4 Interrupt Acknowledge (INTA) Cycles

In response to an interrupt request on the INTR input when interrupts are enabled, the 80386 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 asserted.

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



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

**Figure 5-22. Interrupt Acknowledge Cycles**

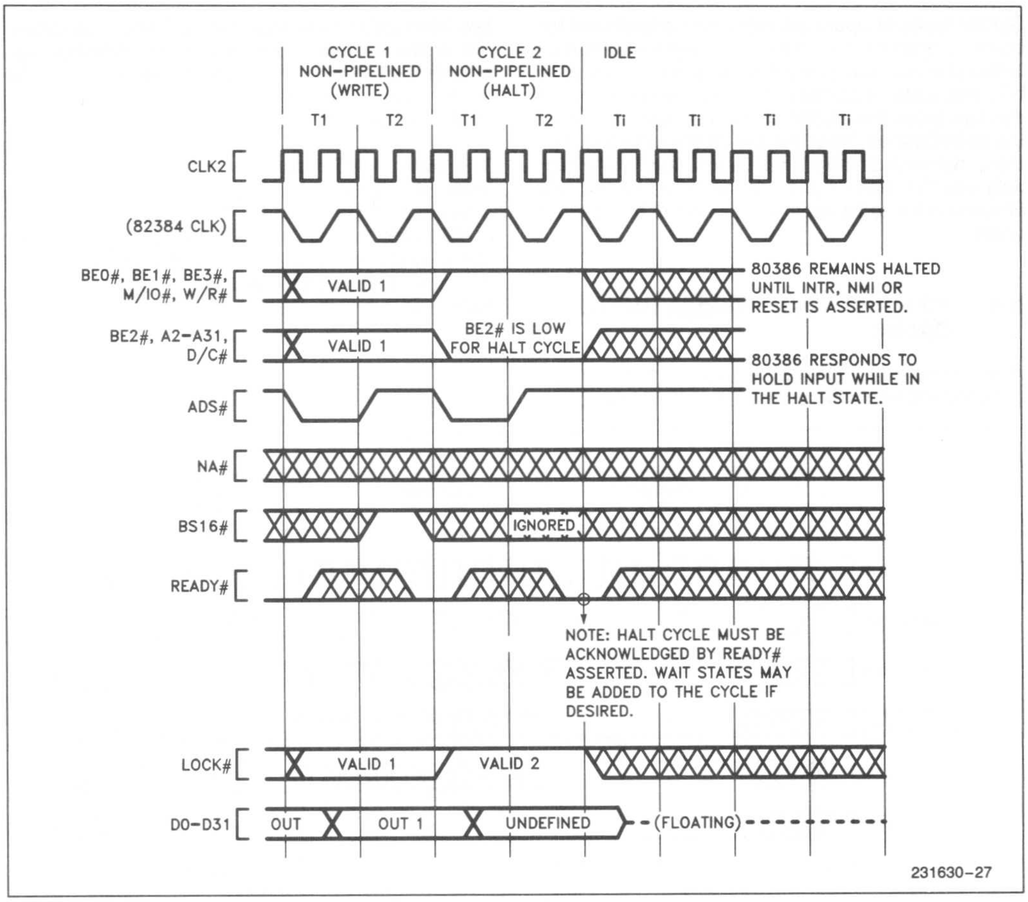

CYCLE 1 NON-PIPELINED (WRITE)
CYCLE 2 NON-PIPELINED (HALT)
IDLE
T1
T2
T1
T2
Ti
Ti
Ti
Ti
CLK2
(82384 CLK)
BE0#, BE1#, BE3#, M/IO#, W/R#
VALID 1
80386 REMAINS HALTED UNTIL INTR, NMI OR RESET IS ASSERTED.
BE2#, A2-A31, D/C#
VALID 1
BE2# IS LOW FOR HALT CYCLE
80386 RESPONDS TO HOLD INPUT WHILE IN THE HALT STATE.
ADS#
NA#
BS16#
IGNORED
READY#
NOTE: HALT CYCLE MUST BE ACKNOWLEDGED BY READY# ASSERTED. WAIT STATES MAY BE ADDED TO THE CYCLE IF DESIRED.
LOCK#
VALID 1
VALID 2
D0-D31
OUT
OUT 1
UNDEFINED
(FLOATING)
231630-27

**Figure 5-23. Halt Indication Cycle**

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, Ti, are inserted by the 80386 between the two interrupt acknowledge cycles, allowing at least 160 ns of locked idle time for future 80386 speed selections up to 24 MHz (CLK2 up to 48 MHz), for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D0–D31 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 80386 will read an external interrupt vector from D0–D7 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.

### 5.4.5 Halt Indication Cycle

The 80386 halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown in **5.2.5 Bus Cycle Definition** and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by READY# asserted.

A halted 80386 resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

### 5.4.6 Shutdown Indication Cycle

The 80386 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 **5.2.5 Bus Cycle Definition** and a byte address of 0. BE0# and BE2# are the only signals distinguishing shutdown indication from halt indication, which drives an address of 2. During the shutdown cycle undefined data is driven on D0–D31. The shutdown indication cycle must be acknowledged by READY# asserted.

A shutdown 80386 resumes execution when NMI or RESET is asserted.

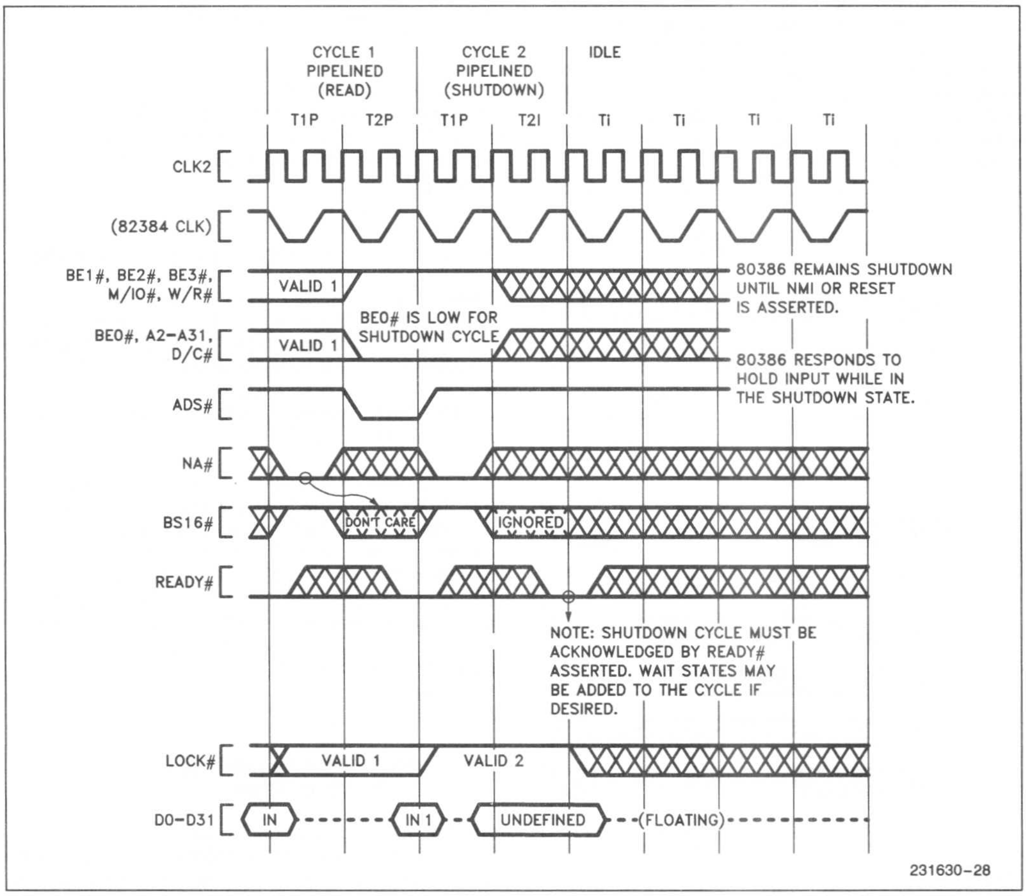
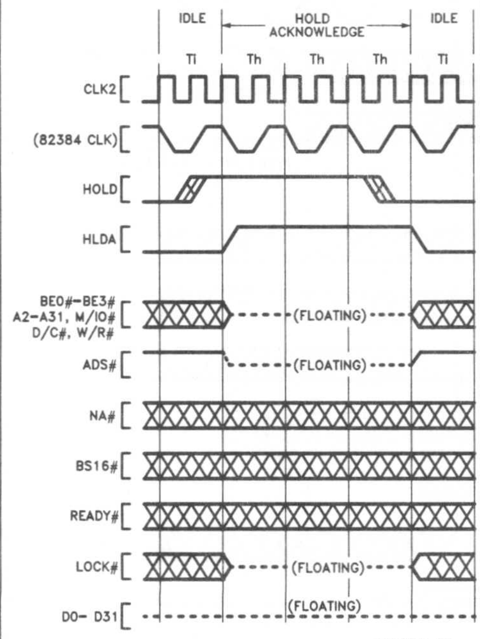

CYCLE 1 PIPELINED (READ)
CYCLE 2 PIPELINED (SHUTDOWN)
IDLE
T1P
T2P
T1P
T2I
Ti
Ti
Ti
Ti
CLK2
(82384 CLK)
BE1#, BE2#, BE3#, M/IO#, W/R#
VALID 1
BE0# IS LOW FOR SHUTDOWN CYCLE
80386 REMAINS SHUTDOWN UNTIL NMI OR RESET IS ASSERTED.
BE0#, A2–A31, D/C#
VALID 1
80386 RESPONDS TO HOLD INPUT WHILE IN THE SHUTDOWN STATE.
ADS#
NA#
BS16#
DON'T CARE
IGNORED
READY#
NOTE: SHUTDOWN CYCLE MUST BE ACKNOWLEDGED BY READY# ASSERTED. WAIT STATES MAY BE ADDED TO THE CYCLE IF DESIRED.
LOCK#
VALID 1
VALID 2
D0–D31
IN
IN 1
UNDEFINED
(FLOATING)
231630–28

**Figure 5-24. Shutdown Indication Cycle**

## 5.5 OTHER FUNCTIONAL DESCRIPTIONS

### 5.5.1 Entering and Exiting Hold Acknowledge

The bus hold acknowledge state, Th, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 80386 floats all output or bidirectional signals, except for HLDA. HLDA is asserted as long as the 80386 remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD and RESET are ignored (also up to one rising edge on NMI is remembered for processing when HOLD is no longer asserted).



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

**Figure 5-25. Requesting Hold from Idle Bus**

Th may be entered from a bus idle state as in Figure 5-25 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5-26 and 5-27. If asserting BS16# requires a second 16-bit bus cycle to complete a physical operand transfer, it is performed before HOLD is acknowledged, although the bus state diagrams in Figures 5-13 and 5-20 do not indicate that detail.

Th is exited in response to the HOLD input being negated. The following state will be Ti as in Figure 5-25 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 5-26 and 5-27.

Th is also exited in response to RESET being asserted.

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

### 5.5.2 Reset During Hold Acknowledge

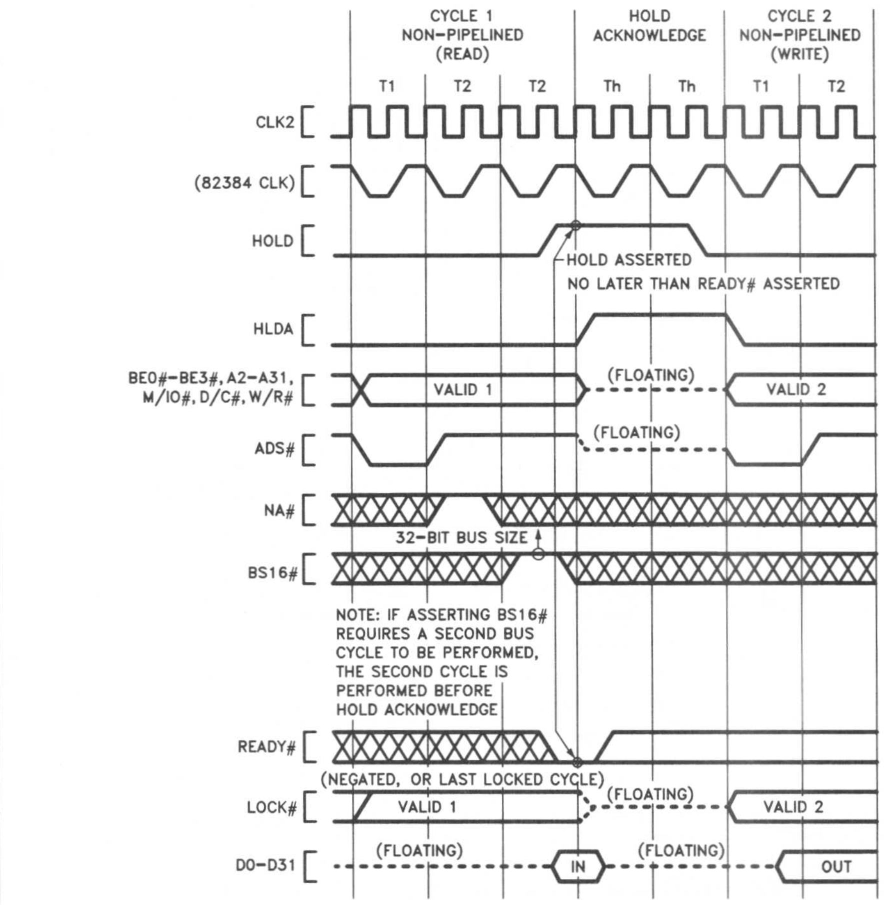
RESET being asserted takes priority over HOLD being asserted. Therefore, Th is exited in reponse to the RESET input being asserted. If RESET is asserted while HOLD remains asserted, the 80386 drives its pins to defined states during reset, as in **Table 5-3 Pin State During Reset**, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is negated, the 80386 enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 80386 would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is negated, the BUSY# input is still sampled as usual to determine whether a self test is being requested, and ERROR# is still sampled as usual to determine whether an 80387 vs. an 80287 (or none) is present.

### 5.5.3 Bus Activity During and Following Reset

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

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 80386, and at least 78 CLK2 periods if 80386 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 78 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists. The additional RESET pulse width is required to clear additional state prior to a valid self-test.



231630-30

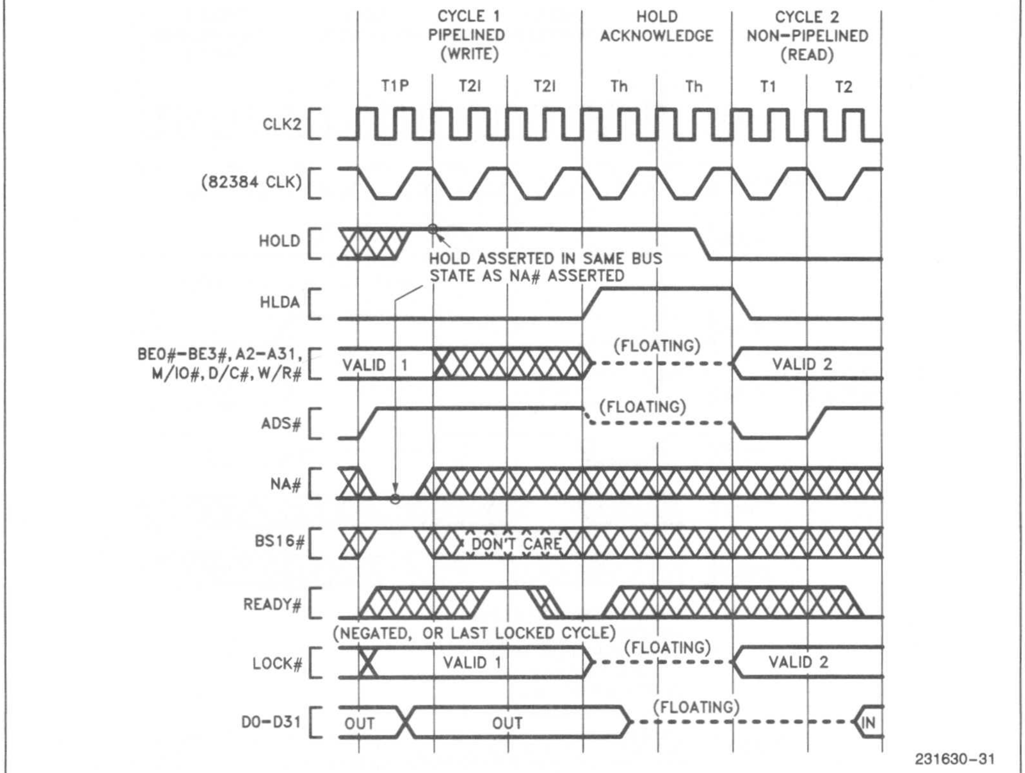
**NOTE:**
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold ($t_{23}$ and $t_{24}$) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

**Figure 5-26. Requesting Hold from Active Bus (NA# negated)**

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 5-28 and Figure 7-7.

An 80386 self-test may be requested at the time RESET is negated by having the BUSY# input at a LOW level, as shown in Figure 5-28. The self-test requires ($2^{20}$) + approximately 60 CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 80386 attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 80386 performs and internal initialization sequence for approximately 350 to 450 CLK2 periods. Also during the initialization, between the 20th CLK2 period and the first bus cycle, the ERROR# input is sampled to determine the presence of an 80387 coprocessor versus the presence of an 80287 (or no coprocessor). To distinguish between an 80287 being present and no coprocessor being present requires a software test.



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

**Figure 5-27. Requesting Hold from Idle Bus (NA# asserted)**

## 5.6 SELF-TEST SIGNATURE

Upon completion of self-test (if self-test was requested by driving BUSY# low at the falling edge of the RESET signal) the AX and DX registers will each indicate 0000H if the 80386 passed with no faults detected. This applies to all 80386 revision levels. Non-zero values in either AX or DX after self-test indicate that particular 80386 unit is faulty.

## 5.7 COMPONENT AND REVISION IDENTIFIERS

To assist 80386 users, the 80386 after reset holds a component identifier and revision identifier in its BH and BL registers respectively. BH contains 03H as identification of the 80386 component. BL contains an unsigned binary number related to the component revision level. The 80386 revision identifier in BL begins chronologically with value zero and is subject to change (typically it will be incremented) with component steppings intended to have certain improvements or distinctions from previous steppings.

These features are intended to assist 80386 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 materials required to be changed. Intel has sole discretion over these characteristics of the component.

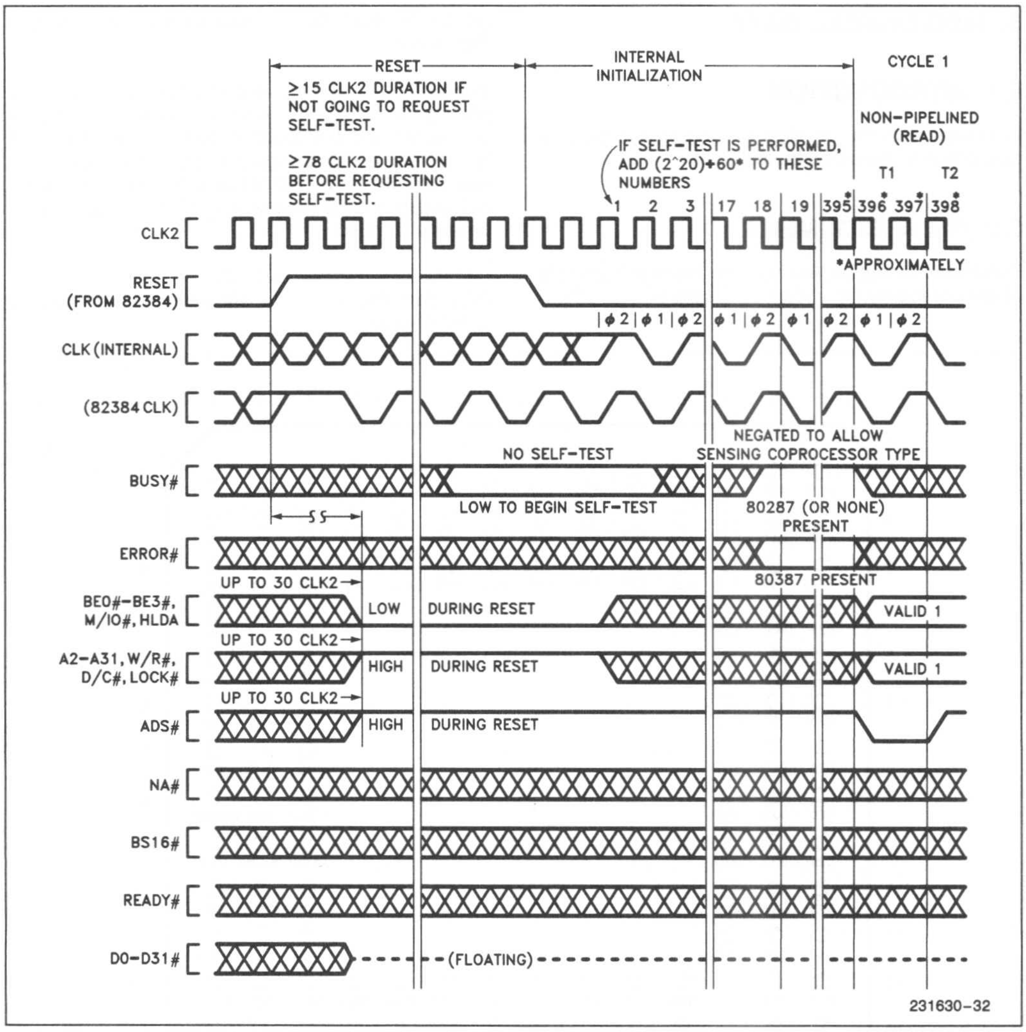


**Figure 5-28. Bus Activity from Rest Until First Code Fetch**

**Table 5-10. Component and Revision Identifier History**

| 80386 Stepping Name | Component Identifier | Revision Identifier | 80386 Stepping Name | Component Identifier | Revision Identifier |
|---|---|---|---|---|---|
| | | | | | |

## 6. MECHANICAL DATA

### 6.1 INTRODUCTION

In this section, the physical packaging and its connections are described in detail.

### 6.2 PIN ASSIGNMENT

The 80386 pinout as viewed from the Substrate side of the component is shown by Figure 6-1. Its pinout as viewed from the Pin side of the component is Figure 6-2.

$V_{CC}$ and GND connections must be made to multiple $V_{CC}$ and GND pins. Each $V_{CC}$ and GND must be connected to the appropriate voltage level. Externally strap all $V_{CC}$ pins together close to the package, and similarly strap all GND pins. Preferrably, the circuit board should include $V_{CC}$ and GND planes for power distribution.

**NOTE:**
Pins identified as "N.C." should remain completely unconnected.

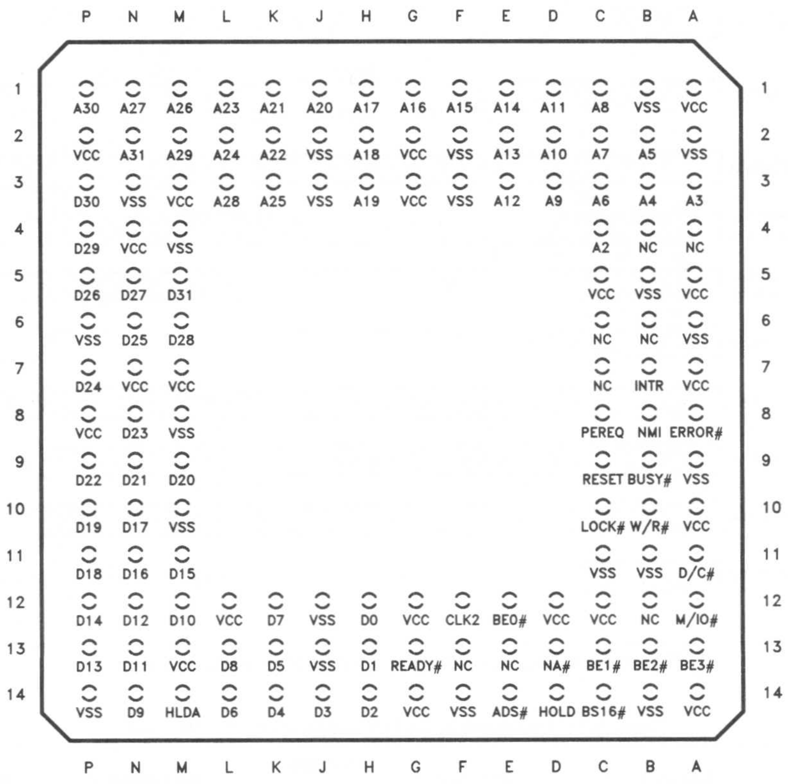
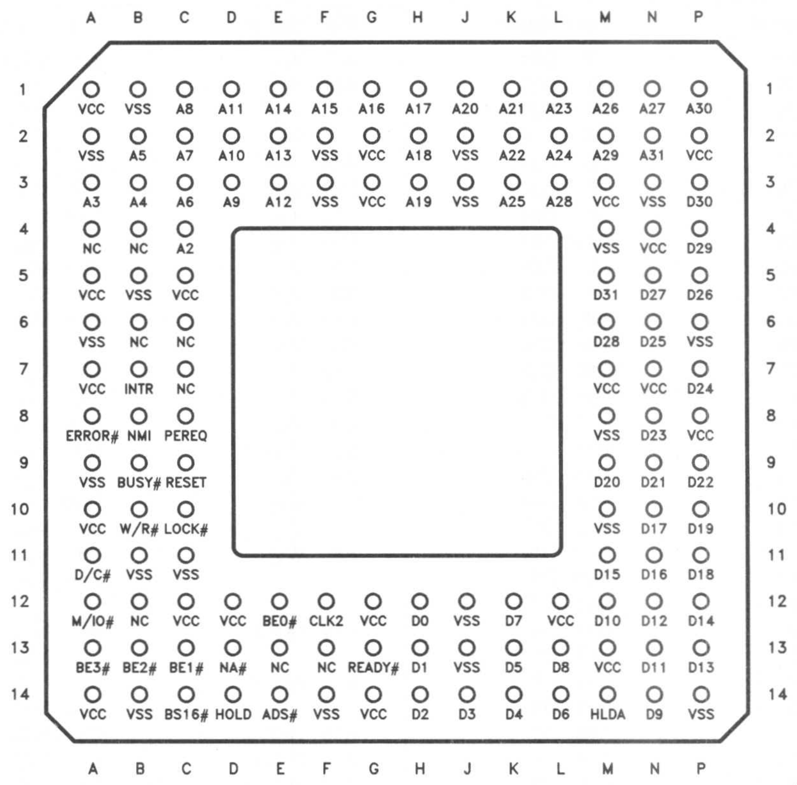

P N M L K J H G F E D C B A
1 A30 A27 A26 A23 A21 A20 A17 A16 A15 A14 A11 A8 VSS VCC 1
2 VCC A31 A29 A24 A22 VSS A18 VCC VSS A13 A10 A7 A5 VSS 2
3 D30 VSS VCC A28 A25 VSS A19 VCC VSS A12 A9 A6 A4 A3 3
4 D29 VCC VSS A2 NC NC 4
5 D26 D27 D31 VCC VSS VCC 5
6 VSS D25 D28 NC NC VSS 6
7 D24 VCC VCC NC INTR VCC 7
8 VCC D23 VSS PEREQ NMI ERROR# 8
9 D22 D21 D20 RESET BUSY# VSS 9
10 D19 D17 VSS LOCK# W/R# VCC 10
11 D18 D16 D15 VSS VSS D/C# 11
12 D14 D12 D10 VCC D7 VSS D0 VCC CLK2 BE0# VCC VCC NC M/IO# 12
13 D13 D11 VCC D8 D5 VSS D1 READY# NC NC NA# BE1# BE2# BE3# 13
14 VSS D9 HLDA D6 D4 D3 D2 VCC VSS ADS# HOLD BS16# VSS VCC 14
P N M L K J H G F E D C B A
231630-33

**Figure 6-1. 80386 PGA Pinout—View from Top Side**


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

231630–34

**Figure 6-2. 80386 PGA Pinout—View from Pin Side**

Table 6-1. 80386 PGA Pinout—Functional Grouping

| Pin / Signal | Pin / Signal | Pin / Signal | Pin / Signal |
|---|---|---|---|
| N2 A31 | M5 D31 | A1 $V_{CC}$ | A2 $V_{SS}$ |
| P1 A30 | P3 D30 | A5 $V_{CC}$ | A6 $V_{SS}$ |
| M2 A29 | P4 D29 | A7 $V_{CC}$ | A9 $V_{SS}$ |
| L3 A28 | M6 D28 | A10 $V_{CC}$ | B1 $V_{SS}$ |
| N1 A27 | N5 D27 | A14 $V_{CC}$ | B5 $V_{SS}$ |
| M1 A26 | P5 D26 | C5 $V_{CC}$ | B11 $V_{SS}$ |
| K3 A25 | N6 D25 | C12 $V_{CC}$ | B14 $V_{SS}$ |
| L2 A24 | P7 D24 | D12 $V_{CC}$ | C11 $V_{SS}$ |
| L1 A23 | N8 D23 | G2 $V_{CC}$ | F2 $V_{SS}$ |
| K2 A22 | P9 D22 | G3 $V_{CC}$ | F3 $V_{SS}$ |
| K1 A21 | N9 D21 | G12 $V_{CC}$ | F14 $V_{SS}$ |
| J1 A20 | M9 D20 | G14 $V_{CC}$ | J2 $V_{SS}$ |
| H3 A19 | P10 D19 | L12 $V_{CC}$ | J3 $V_{SS}$ |
| H2 A18 | P11 D18 | M3 $V_{CC}$ | J12 $V_{SS}$ |
| H1 A17 | N10 D17 | M7 $V_{CC}$ | J13 $V_{SS}$ |
| G1 A16 | N11 D16 | M13 $V_{CC}$ | M4 $V_{SS}$ |
| F1 A15 | M11 D15 | N4 $V_{CC}$ | M8 $V_{SS}$ |
| E1 A14 | P12 D14 | N7 $V_{CC}$ | M10 $V_{SS}$ |
| E2 A13 | P13 D13 | P2 $V_{CC}$ | N3 $V_{SS}$ |
| E3 A12 | N12 D12 | P8 $V_{CC}$ | P6 $V_{SS}$ |
| D1 A11 | N13 D11 | | P14 $V_{SS}$ |
| D2 A10 | M12 D10 | | |
| D3 A9 | N14 D9 | F12 CLK2 | A4 N.C. |
| C1 A8 | L13 D8 | | D4 N.C. |
| C2 A7 | K12 D7 | E14 ADS# | B6 N.C. |
| C3 A6 | L14 D6 | | B12 N.C. |
| B2 A5 | K13 D5 | B10 W/R# | C6 N.C. |
| B3 A4 | K14 D4 | A11 D/C# | C7 N.C. |
| A3 A3 | J14 D3 | A12 M/IO# | E13 N.C. |
| C4 A2 | H14 D2 | C10 LOCK# | F13 N.C. |
| A13 BE3# | H13 D1 | | |
| B13 BE2# | H12 D0 | D13 NA# | C8 PEREQ |
| C13 BE1# | | C14 BS16# | B9 BUSY# |
| E12 BE0# | | G13 READY# | A8 ERROR# |
| | D14 HOLD | | |
| C9 RESET | M14 HLDA | B7 INTR | B8 NMI |

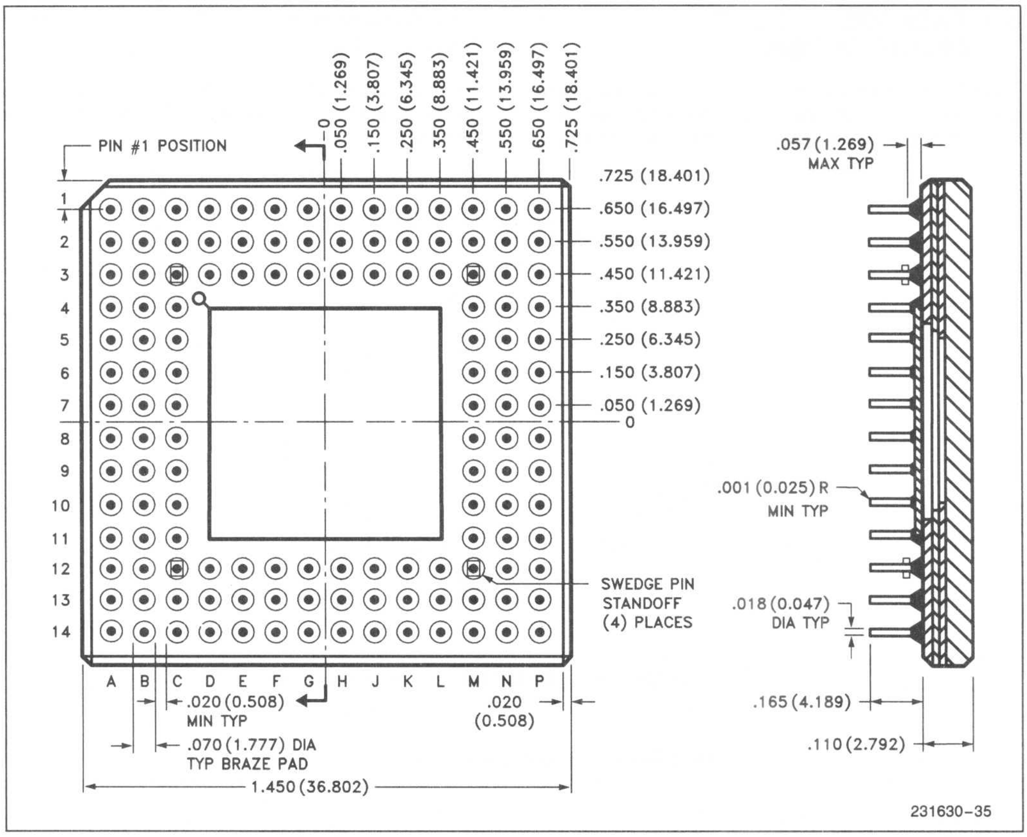


**Figure 6-3. 132-Pin Ceramic PGA Package Dimensions**

## 6.3 Package Dimensions and Mounting

The initial 80386 package is a 132-pin ceramic pin grid array (PGA). Pins of this package are arranged 0.100 inch (2.54mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are listed in Table 6-2.

## 6.4 PACKAGE THERMAL SPECIFICATION

The 80386 is specified for operation when case temperature is within the range of 0°C–85°C. The case temperature may be measured in any environment, to determine whether the 80386 is within specified operating range.

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 6-4.

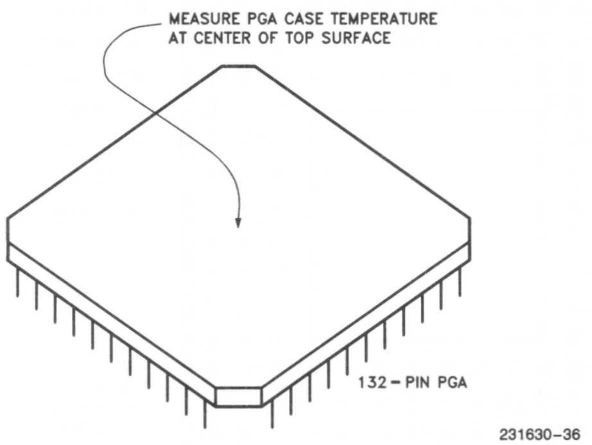


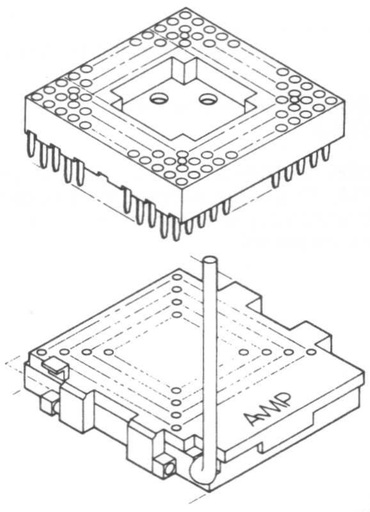
**Figure 6-4. Measuring 80386 PGA Case Temperature**

**Table 6-2. Several Socket Options for 132-Pin PGA**

* Low insertion force (LIF) soldertail
  55274-1
* Amp tests indicate 50% reduction in insertion force compared to machined sockets

Other socket options
* Zero insertion force (ZIF) soldertail
  55583-1
* Zero insertion force (ZIF) Burn-in version
  55573-2

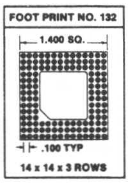
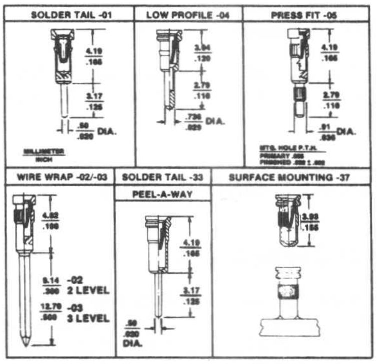
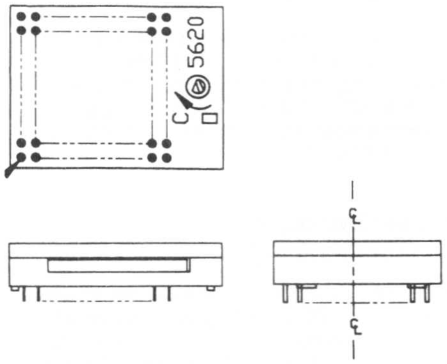
**Amp Incorporated**
(Harrisburg, PA 17105 U.S.A.
Phone 717-564-0100)



231630–45
Cam handle locks in low profile position when substrate is installed (handle UP for open and DOWN for closed positions)

courtesy Amp Incorporated

**Table 6-2. Several Socket Options for 132-Pin PGA** (Continued)

| | |
|---|---|
| Peel-A-Way™ Mylar and Kapton Socket Terminal Carriers<br>• Low insertion force surface mount CS132-37TG<br>• Low insertion force soldertail CS132-01TG<br>• Low insertion force wire-wrap CS132-02TG (two level) CS132-03TG (three-level)<br>• Low insertion force press-fit CS132-05TG<br><br>**Advanced Interconnections**<br>(5 Division Street<br>Warwick, RI 02818 U.S.A.<br>Phone 401-885-0485) | Peel-A-Way Carrier No. 132: Kapton Carrier is KS132 Mylar Carrier is MS132<br>Molded Plastic Body KS132 is shown below:<br> FOOT PRINT NO. 132<br>1.400 SQ.<br>.100 TYP<br>14 x 14 x 3 ROWS<br>231630–46<br> SOLDER TAIL -01<br>LOW PROFILE -04<br>PRESS FIT -05<br>WIRE WRAP -02/-03<br>SOLDER TAIL -33<br>PEEL-A-WAY<br>SURFACE MOUNTING -37<br>MILLIMETER / INCH<br>DIA.<br>-02 2 LEVEL<br>-03 3 LEVEL<br>231630–47<br>courtesy Advanced Interconnections (Peel-A-Way Terminal Carriers U.S. Patent No. 4442938) |
| • Low insertion force socket soldertail (for production use) 2XX-6576-00-3308 (new style) 2XX-6003-00-3302 (older style)<br>• Zero insertion force soldertail (for test and burn-in use) 2XX-6568-00-3302<br><br>**Textool Products**<br>**Electronic Products Division/3M**<br>(1410 West Pioneer Drive<br>Irving, Texas 75601 U.S.A.<br>Phone 214-259-2676) |  <br>231630–48<br>courtesy Textool Products/3M |

# 7. ELECTRICAL DATA

## 7.1 INTRODUCTION

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

## 7.2 POWER AND GROUNDING

### 7.2.1 Power Connections

The 80386 is implemented in CHMOS III technology and has modest power requirements. However, its high clock frequency and 72 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 20 $V_{CC}$ and 21 $V_{SS}$ pins separately feed functional units of the 80386.

Power and ground connections must be made to all external $V_{CC}$ and GND pins of the 80386. On the circuit board, all $V_{CC}$ pins must be strapped closely together, preferrably on a $V_{CC}$ plane. All $V_{SS}$ pins must be likewise strapped on the circuit board, preferrably on a GND plane.

### 7.2.2 Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 80386. The 80386 driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 80386 and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

### 7.2.3 Resistor Recommendations

The ERROR# and BUSY# inputs have resistor pullups (of approximately 20 KΩ built-in to the 80386 to keep these signals negated when neither 80287 or 80387 are present in the system (or temporarily removed from its socket).

In typical designs, the external pullup 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 pullup resistors in other ways.

### 7.2.4 Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. N.C. pins should always remain unconnected.

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

| Pin | Signal |
|---|---|
| B7 | INTR |
| B8 | NMI |
| D14 | HOLD |

If not using address pipelining, pullup D13 NA# to $V_{CC}$.

If not using 16-bit bus size, pullup C14 BS16# to $V_{CC}$.

Pullups in the range of 20 KΩ are recommended.

**Table 7-1. Recommended Resistor Pullups to $V_{CC}$**

| Pin and Signal | Pullup Value | Purpose |
|---|---|---|
| E14 ADS# | 20 KΩ ±10% | Lightly Pull ADS# Negated During 80386 Hold Acknowledge States |
| C10 LOCK# | 20 KΩ ±10% | Lightly Pull LOCK# Negated During 80386 Hold Acknowledge States |

## 7.3 MAXIMUM RATINGS

**Table 7-2. Maximum Ratings**

| Parameter | 80386-12<br>80386-16<br>Maximum Rating |
|---|---|
| Storage Temperature | −65°C to +150°C |
| Case Temperature Under Bias | −65°C to +110°C |
| Supply Voltage with Respect to $V_{SS}$ | −0.5V to +6.5V |
| Voltage on Other Pins | −0.5V to $V_{CC}$ + 0.5V |

Table 7-2 is a stress rating only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **7.3 D.C. Specifications** and **7.4 A.C. Specifications.**

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

## 7.4 D.C. SPECIFICATIONS

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

**Table 7-3. 80386-16 and 80386-12 D.C. Characteristics**

| Symbol | Parameter | 80386-12<br>80386-16<br>Min | 80386-12<br>80386-16<br>Max | Unit | Notes |
|---|---|---|---|---|---|
| $V_{IL}$ | Input Low Voltage | −0.3 | 0.8 | V | |
| $V_{IH}$ | Input High Voltage | 2.0 | $V_{CC}$ + 0.3 | V | |
| $V_{ILC}$ | 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<br>$I_{OL}$ = 4 mA: A2–A31, D0–D31<br>$I_{OL}$ = 5 mA: BE0#–BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA | <br><br> | <br>0.45<br>0.45 | <br>V<br>V | |
| $V_{OH}$ | Output High Voltage<br>$I_{OH}$ = 1 mA: A2–A31, D0–D31<br>$I_{OH}$ = 0.9 mA: BE0#–BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA | <br>2.4<br>2.4 | | <br>V<br>V | |
| $I_{LI}$ | Input Leakage Current | | ±15 | μA | 0V ≤ $V_{IN}$ ≤ $V_{CC}$ |
| $I_{LO}$ | Output Leakage Current | | ±15 | μA | 0.45V ≤ $V_{OUT}$ ≤ $V_{CC}$ |
| ICC | Supply Current<br>CLK2 = 25 MHz: with 80386-12<br>CLK2 = 32 MHz: with 80386-16 | | <br>400<br>400 | <br>mA<br>mA | |
| $C_{IN}$ | Input Capacitance | | 10 | pF | $F_c$ = 1 MHz (Note 1) |
| $C_{OUT}$ | Output or I/O Capacitance | | 12 | pF | $F_c$ = 1 MHz (Note 1) |
| $C_{CLK}$ | CLK2 Capacitance | | 20 | pF | $F_c$ = 1 MHz (Note 1) |

**NOTE:**
1. Not tested.

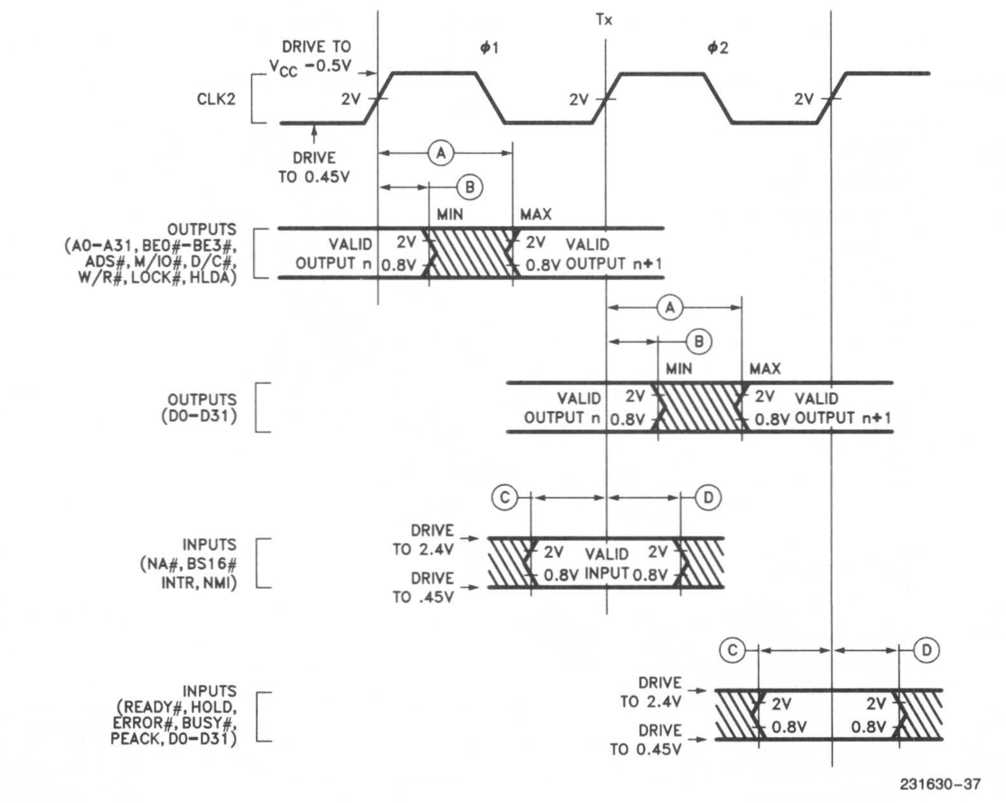
## 7.5 A.C. SPECIFICATIONS

### 7.5.1 A.C. Spec Definitions

The A.C. specifications, given in Tables 7-4 and 7-5, 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. 80386 output delays are specified with minimum and maximum limits, measured as shown. The minimum 80386 delay times are hold times provided to external circuitry. 80386 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 80386 operation.

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



**LEGEND:**

- A — maximum output delay spec
- B — minimum output delay spec
- C — minimum input setup spec
- D — minimum input hold spec

**Figure 7-1. Drive Levels and Measurement Points for A.C. Specifications**

### 7.5.2 A.C. Specification Tables

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

**Table 7-4. 80386-16 A.C. Characteristics**

| Symbol | Parameter | 80386-16 Min | 80386-16 Max | Unit | Ref. Figure | Notes |
|---|---|---|---|---|---|---|
| | Operating Frequency | 4 | 16 | MHz | — | Half of CLK2 Frequency |
| $t_1$ | CLK2 Period | 31 | 125 | ns | 7-3 | |
| $t_{2a}$ | CLK2 High Time | 9 | | ns | 7-3 | at 2V |
| $t_{2b}$ | CLK2 High Time | 5 | | ns | 7-3 | at ($V_{CC}$ − 0.8V) |
| $t_{3a}$ | CLK2 Low Time | 9 | | ns | 7-3 | at 2V |
| $t_{3b}$ | CLK2 Low Time | 7 | | ns | 7-3 | at 0.8V |
| $t_4$ | CLK2 Fall Time | | 8 | ns | 7-3 | ($V_{CC}$ − 0.8V) to 0.8V |
| $t_5$ | CLK2 Rise Time | | 8 | ns | 7-3 | 0.8V to ($V_{CC}$ − 0.8V) |
| $t_6$ | A2–A31 Valid Delay | 1 | 40 | ns | 7-5 | $C_L$ = 120 pF |
| $t_7$ | A2–A31 Float Delay | 1 | 40 | ns | 7-6 | (Note 1) |
| $t_8$ | BE0#–BE3#, LOCK# Valid Delay | 1 | 40 | ns | 7-5 | $C_L$ = 75 pF |
| $t_9$ | BE0#–BE3#, LOCK# Float Delay | 1 | 40 | ns | 7-6 | (Note 1) |
| $t_{10}$ | W/R#, M/IO#, D/C#, ADS#, Valid Delay | 1 | 35 | ns | 7-5 | $C_L$ = 75 pF |
| $t_{11}$ | W/R#, M/IO#, D/C#, ADS# Float Delay | 1 | 35 | ns | 7-6 | (Note 1) |
| $t_{12}$ | D0–D31 Write Data Valid Delay | 1 | 50 | ns | 7-5 | $C_L$ = 120 pF |
| $t_{13}$ | D0–D31 Write Data Float Delay | 1 | 40 | ns | 7-6 | (Note 1) |
| $t_{14}$ | HLDA Valid Delay | 4 | 35 | ns | 7-6 | $C_L$ = 75 pF |
| $t_{15}$ | NA# Setup Time | 10 | | ns | 7-4 | |
| $t_{16}$ | NA# Hold Time | 20 | | ns | 7-4 | |
| $t_{17}$ | BS16# Setup Time | 12 | | ns | 7-4 | |
| $t_{18}$ | BS16# Hold Time | 20 | | ns | 7-4 | |
| $t_{19}$ | READY# Setup Time | 20 | | ns | 7-4 | |
| $t_{20}$ | READY# Hold Time | 3 | | ns | 7-4 | |
| $t_{21}$ | D0–D31 Read Setup Time | 10 | | ns | 7-4 | |
| $t_{22}$ | D0–D31 Read Hold Time | 2 | | ns | 7-4 | |
| $t_{23}$ | HOLD Setup Time | 25 | | ns | 7-4 | |
| $t_{24}$ | HOLD Hold Time | 4 | | ns | 7-4 | |
| $t_{25}$ | RESET Setup Time | 10 | | ns | 7-4 | (Note 2) |

**Table 7-4. 80386-16 A.C. Characteristics** (Continued)

| Symbol | Parameter | 80386-16 Min | 80386-16 Max | Unit | Ref. Figure | Notes |
|---|---|---|---|---|---|---|
| $t_{26}$ | RESET Hold Time | 5 | | ns | 7-4 | (Note 2) |
| $t_{27}$ | NMI, INTR Setup Time | 25 | | ns | 7-4 | (Note 2) |
| $t_{28}$ | NMI, INTR Hold Time | 4 | | ns | 7-4 | (Note 2) |
| $t_{29}$ | PEREQ, ERROR#, BUSY# Setup Time | 25 | | ns | 7-4 | (Note 2) |
| $t_{30}$ | PEREQ, ERROR#, BUSY# Hold Time | 4 | | 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 tested but should be no longer than the valid delay.
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.

**Table 7-5. 80386-12 A.C. Characteristics**

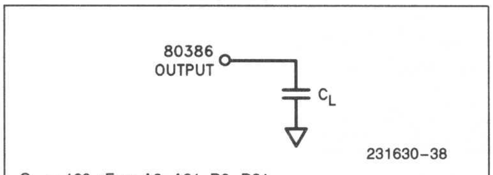
| Symbol | Parameter | 80386-12 Min | 80386-12 Max | Unit | Ref. Figure | Notes |
|---|---|---|---|---|---|---|
| | Operating Frequency | 4 | 12.5 | MHz | — | Half of CLK2 Frequency |
| $t_1$ | CLK2 Period | 40 | 125 | ns | 7-3 | |
| $t_{2a}$ | CLK2 High Time | 11 | | ns | 7-3 | at 2V |
| $t_{2b}$ | CLK2 High Time | 7 | | ns | 7-3 | at ($V_{CC}$ − 0.8V) |
| $t_{3a}$ | CLK2 Low Time | 11 | | ns | 7-3 | at 2V |
| $t_{3b}$ | CLK2 Low Time | 9 | | ns | 7-3 | at 0.8V |
| $t_4$ | CLK2 Fall Time | | 8 | ns | 7-3 | ($V_{CC}$ − 0.8V) to 0.8V |
| $t_5$ | CLK2 Rise Time | | 8 | ns | 7-3 | 0.8V to ($V_{CC}$ − 0.8V) |
| $t_6$ | A2–A31 Valid Delay | 1 | 44 | ns | 7-5 | $C_L$ = 120 pF |
| $t_7$ | A2–A31 Float Delay | 1 | 44 | ns | 7-6 | (Note 1) |
| $t_8$ | BE0#–BE3#, LOCK# Valid Delay | 1 | 44 | ns | 7-5 | $C_L$ = 75 pF |
| $t_9$ | BE0#–BE3#, LOCK# Float Delay | 1 | 44 | ns | 7-6 | (Note 1) |
| $t_{10}$ | W/R#, M/IO#, D/C#, ADS# Valid Delay | 1 | 39 | ns | 7-5 | $C_L$ = 75 pF |
| $t_{11}$ | W/R#, M/IO#, D/C#, ADS# Float Delay | 1 | 39 | ns | 7-6 | (Note 1) |
| $t_{12}$ | D0–D31 Write Data Valid Delay | 1 | 55 | ns | 7-5 | $C_L$ = 120 pF |
| $t_{13}$ | D0–D31 Write Data Float Delay | 1 | 55 | ns | 7-6 | (Note 1) |
| $t_{14}$ | HLDA Valid Delay | 4 | 39 | ns | 7-6 | $C_L$ = 75 pF |

**Table 7-5. 80386-12 A.C. Characteristics** (Continued)

| Symbol | Parameter | 80386-12 Min | 80386-12 Max | Unit | Ref. Figure | Notes |
|---|---|---|---|---|---|---|
| $t_{15}$ | NA# Setup Time | 11 | | ns | 7-4 | |
| $t_{16}$ | NA# Hold Time | 22 | | ns | 7-4 | |
| $t_{17}$ | BS16# Setup Time | 13 | | ns | 7-4 | |
| $t_{18}$ | BS16# Hold Time | 22 | | ns | 7-4 | |
| $t_{19}$ | READY# Setup Time | 22 | | ns | 7-4 | |
| $t_{20}$ | READY# Hold Time | 4 | | ns | 7-4 | |
| $t_{21}$ | D0–D31 Read Setup Time | 11 | | ns | 7-4 | |
| $t_{22}$ | D0–D31 Read Hold Time | 3 | | ns | 7-4 | |
| $t_{23}$ | HOLD Setup Time | 28 | | ns | 7-4 | |
| $t_{24}$ | HOLD Hold Time | 5 | | ns | 7-4 | |
| $t_{25}$ | RESET Setup Time | 11 | | ns | 7-4 | (Note 2) |
| $t_{26}$ | RESET Hold Time | 6 | | ns | 7-4 | (Note 2) |
| $t_{27}$ | NMI, INTR Setup Time | 28 | | ns | 7-4 | (Note 2) |
| $t_{28}$ | NMI, INTR Hold Time | 5 | | ns | 7-4 | (Note 2) |
| $t_{29}$ | PEREQ, ERROR#, BUSY# Setup Time | 28 | | ns | 7-4 | (Note 2) |
| $t_{30}$ | PEREQ, ERROR#, BUSY# Hold Time | 5 | | ns | 7-4 | (Note 2) |

**NOTES:**
1. Float condition occurs when maximum output current becomes less than $I_{LO}$ in magnitude. Float delay is not tested, but should be no longer than the valid delay.
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.

## 7.5.3 A.C. Test Loads



$C_L$ = 120 pF on A2–A31, D0–D31
$C_{IL}$ = 75 pF on BE0#–BE3#, W/R#, M/IO#, D/C#, ADS#, LOCK#, HLDA
$C_L$ includes all parasitic capacitances.

**Figure 7-2. A.C. Test Load**

## 7.5.4 A.C. Timing Waveforms

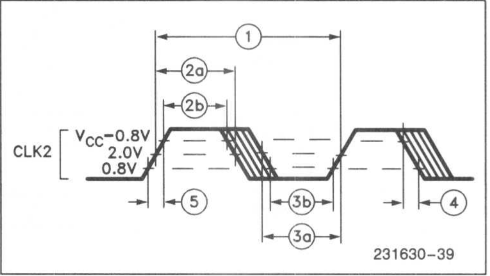


**Figure 7-3. CLK2 Timing**

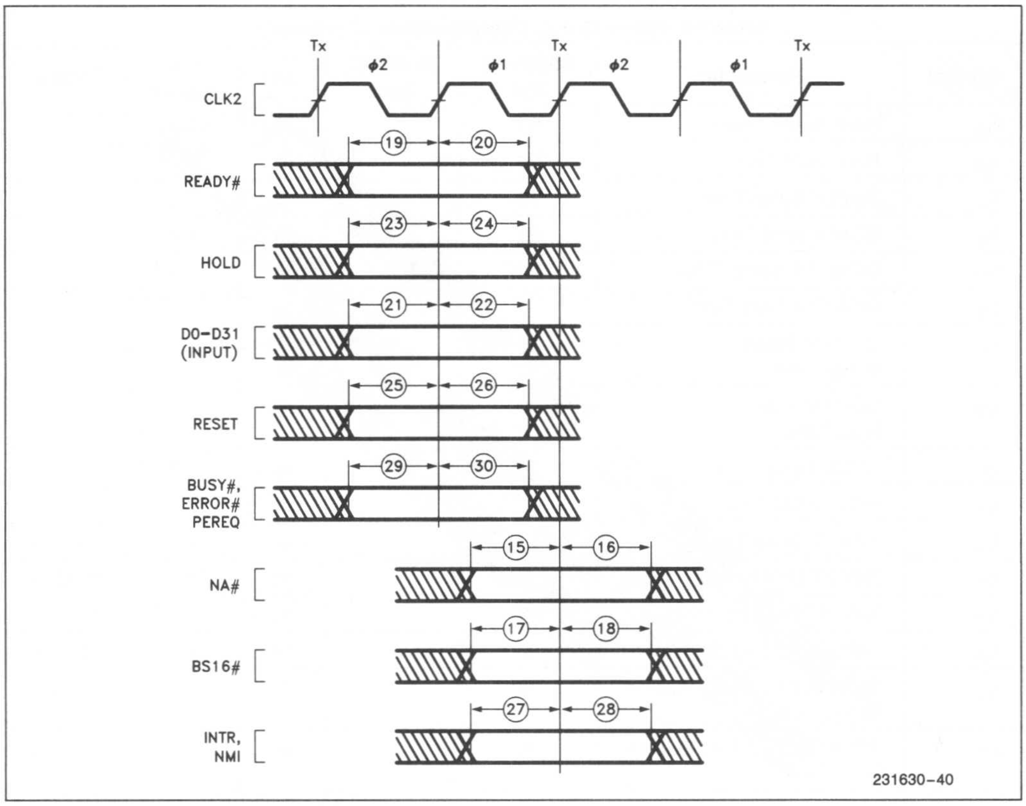


**Figure 7-4. Input Setup and Hold Timing**

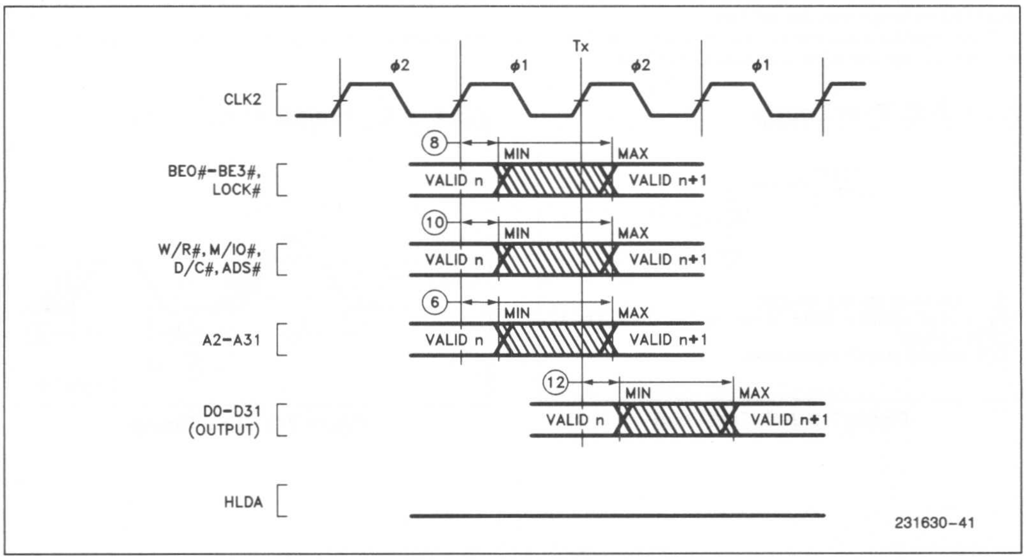


**Figure 7-5. Output Valid Delay Timing**

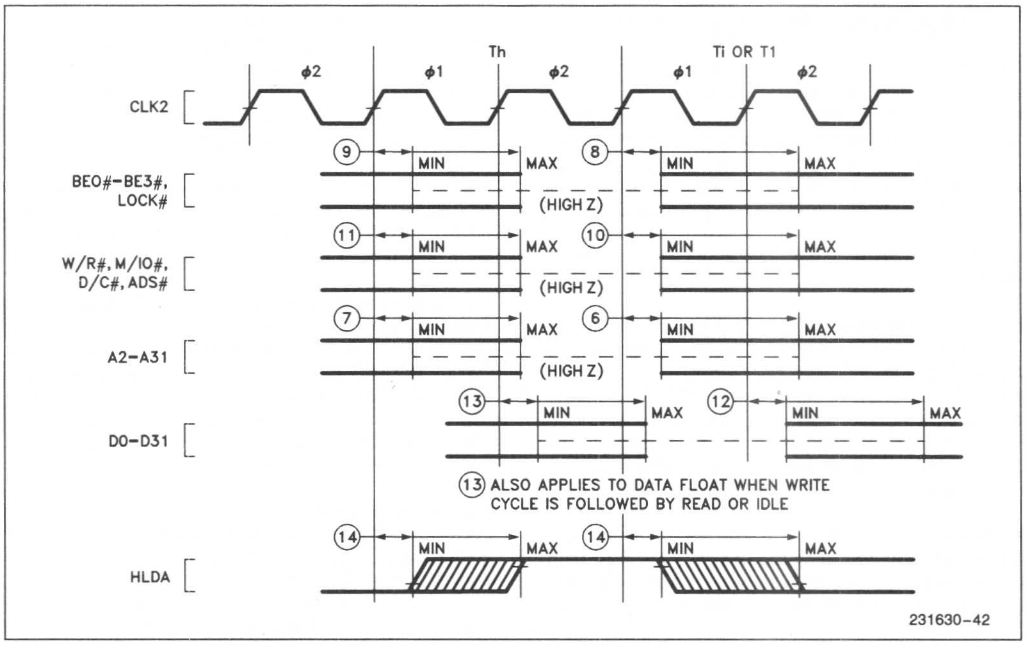
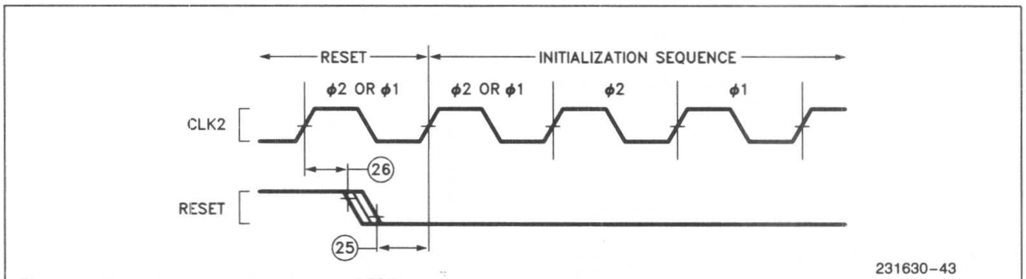


**Figure 7-6. Output Float Delay and HLDA Valid Delay Timing**



The second internal processor phase following RESET high-to-low transition (provided $t_{25}$ and $t_{26}$ are met) is $\phi 2$.

**Figure 7-7. RESET Setup and Hold Timing, and Internal Phase**

## 7.6 DESIGNING FOR ICE-386 USE

The 80386 in-circuit emulator product is ICE-386. Because of the high operating frequency of 80386 systems and ICE-386, there is no cable separating the ICE-386 probe module from the target system. The ICE-386 probe module has several electrical and mechanical characteristics that should be taken into consideration when designing the hardware.

**Capacitive loading**: ICE-386 adds up to 25 pF to each line.

**Drive requirement**: ICE-386 adds one standard TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load per control signal line, and one advanced low-power Schottky TTL load per address, byte enable, and data line. These loads are within the probe module and are driven by the probe's 80386, which has standard drive and loading capability listed in Tables 7-3 and 7-4.

**Power requirement**: For noise immunity the ICE-386 probe is powered by the user system. The high-speed probe circuitry draws up to 1.1A plus the maximum 80386 $I_{CC}$ from the user 80386 socket.

**80386 location and orientation**: The ICE-386 probe requires lateral clearance illustrated in Figure 7-8, viewed from above the user 80386 socket. The ICE-386 probe module alone requires vertical clearance 1.25 inches (3.2 cm) above the height of surrounding circuitry. The Optional Interface Board (OIB), used for extra electrical buffering initially, has the same lateral clearance as Figure 7-8, and **adds** 0.5 inches (1.3 cm) to the vertical clearance.

**READY# drive**: The ICE-386 system may be able to clear a user system READY# hang if the user's READY# driver is implemented with an open-collector or tri-state device.

**Optional Interface Board (OIB) and CLK2 speed reduction**: When the ICE-386 processor probe is first attached to an unverified user system, the OIB helps ICE-386 function in user systems with bus faults (shorted signals, etc.). After electrical verification it may be removed. Only when the OIB is installed, the user system must have a reduced CLK2 frequency of 16 MHz maximum.

**Cache coherence**: ICE-386 loads user memory by performing 80386 write cycles. Note that if the user system is not designed to update or invalidate its cache (if it has a cache) upon processor writes to memory, the cache could contain stale instruction code and/or data. For best use of ICE-386, the user should consider designing the cache (if any) to update itself automatically when processor writes occur, or find another method of maintaining cache data coherence with main user memory.

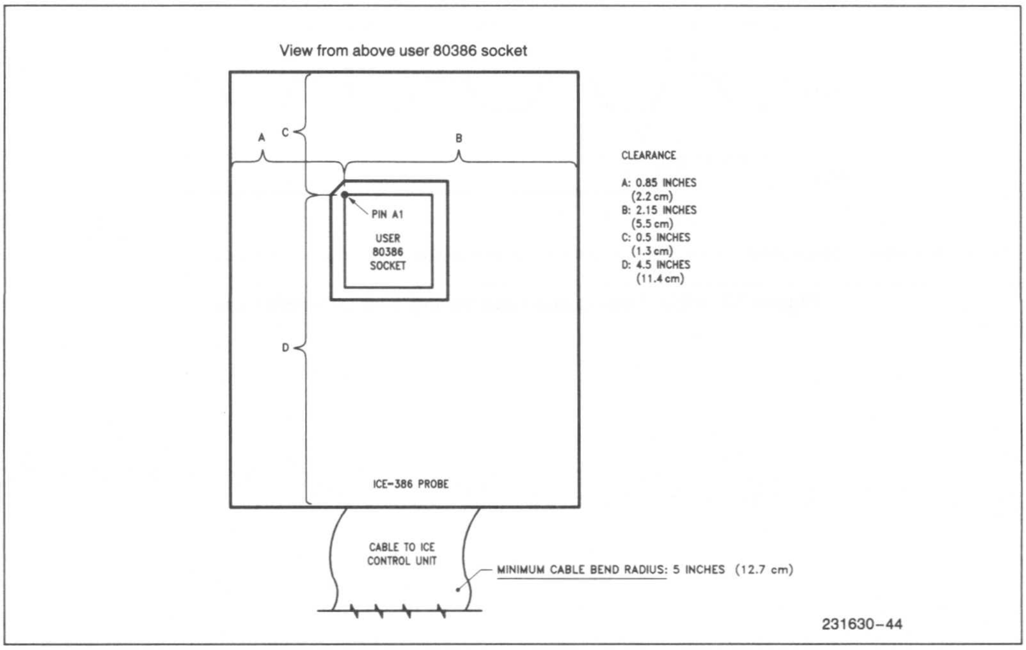


**Figure 7-8. ICE-386 Lateral Clearance Requirements (Preliminary)**

## 8. INSTRUCTION SET

This section describes the 80386 instruction set. A table lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 80386 instructions.

### 8.1 80386 INSTRUCTION ENCODING AND CLOCK COUNT SUMMARY

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 8-1 below, by the processor clock period (e.g. 62.5 ns for an 80386-16 operating at 16 MHz (32 MHz CLK2 signal)). The actual clock count of an 80386 program will average 5% more than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

For more detailed information on the encodings of instructions refer to section 8.2 Instruction Encodings. Section 8.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction encoding.

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.

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 bytes of code in next instruction executed.

**Table 8-1. 80386 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 | 1 0 0 0 1 0 0 w | mod reg r/m | | 2/2 | 2/2 | b | h |
| Register/Memory to Register | 1 0 0 0 1 0 1 w | mod reg r/m | | 2/4 | 2/4 | b | h |
| Immediate to Register/Memory | 1 1 0 0 0 1 1 w | mod 0 0 0 r/m | | 2/2 | 2/2 | b | h |
| Immediate to Register | 1 0 1 1 w reg | immediate data | | 2 | 2 | | |
| Memory to Accumulator (short form) | 1 0 1 0 0 0 0 w | full displacement | | 4 | 4 | b | h |
| Accumulator to Memory (short form) | 1 0 1 0 0 0 1 w | full displacement | | 2 | 2 | b | h |
| Register Memory to Segment Register | 1 0 0 0 1 1 1 0 | mod 0 sreg r/m | | 2/5 | 18/19 | b | h, i, j |
| Segment Register to Register/Memory | 1 0 0 0 1 1 0 0 | mod 0 sreg r/m | | 2/2 | 2/2 | b | h |
| **MOVSX = Move With Sign Extension** | | | | | | | |
| Register From Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 1 1 1 1 1 w | mod reg r/m | 3/6 | 3/6 | b | h |
| **MOVZX = Move With Zero Extension** | | | | | | | |
| Register From Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 1 1 0 1 1 w | mod reg r/m | 3/6 | 3/6 | b | h |
| **PUSH = Push:** | | | | | | | |
| Memory | 1 1 1 1 1 1 1 1 | mod 1 1 0 r/m | | 5 | 5 | b | h |
| Register | 0 1 0 1 0 reg | | | 2 | 2 | b | h |
| Segment Register (ES, CS, SS or DS) | 0 0 0 sreg 1 1 0 | | | 2 | 2 | b | h |
| Segment Register (FS or GS) | 0 0 0 0 1 1 1 1 | 1 0 sreg 0 0 0 | | 2 | 2 | b | h |
| Immediate | 0 1 1 0 1 0 s 0 | immediate data | | 2 | 2 | b | h |
| PUSHA = Push All | 0 1 1 0 0 0 0 0 | | | 18 | 18 | b | h |
| **POP = Pop** | | | | | | | |
| Memory | 1 0 0 0 1 1 1 1 | mod 0 0 0 r/m | | 5 | 5 | b | h |
| Register | 0 1 0 1 1 reg | | | 4 | 4 | b | h |
| Segment Register (ES, CS, SS or DS) | 0 0 0 sreg 1 1 1 | | | 7 | 21 | b | h, i, j |
| Segment Register (FS or GS) | 0 0 0 0 1 1 1 1 | 1 0 sreg 0 0 1 | | 7 | 21 | b | h, i, j |
| **POPA = Pop All** | 0 1 1 0 0 0 0 1 | | | 24 | 24 | b | h |
| **XCHG = Exchange** | | | | | | | |
| Register/Memory With Register | 1 0 0 0 0 1 1 w | mod reg r/m | | 3/5 | 3/5 | b, f | f, h |
| Register With Accumulator (short form) | 1 0 0 1 0 reg | | | 3 | 3 | | |
| **IN = Input from:** | | | | | | | |
| Fixed Port | 1 1 1 0 0 1 0 w | port number | | 5 | 5 | | m |
| Variable Port | 1 1 1 0 1 1 0 w | | | 6 | 6 | | m |
| **OUT = Output to:** | | | | | | | |
| Fixed Port | 1 1 1 0 0 1 1 w | port number | | 3 | 3 | | m |
| Variable Port | 1 1 1 0 1 1 1 w | | | 4 | 4 | | m |
| **LEA = Load EA to Register** | 1 0 0 0 1 1 0 1 | mod reg r/m | | 2 | 2 | | |

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

| INSTRUCTION | FORMAT | | | | CLOCK COUNT | | NOTES | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Real Address Mode or Virtual 8086 Mode | Protected Virtual Address Mode | Real Address Mode or Virtual 8086 Mode | Protected Virtual Address Mode |
| **SEGMENT CONTROL** | | | | | | | | |
| **LDS = Load Pointer to DS** | 11000101 | mod reg r/m | | | 7 | 22 | b | h, i, j |
| **LES = Load Pointer to ES** | 11000100 | mod reg r/m | | | 7 | 22 | b | h, i, j |
| **LFS = Load Pointer to FS** | 00001111 | 10110100 | mod reg r/m | | 7 | 25 | b | h, i, j |
| **LGS = Load Pointer to GS** | 00001111 | 10110101 | mod reg r/m | | 7 | 25 | b | h, i, j |
| **LSS = Load Pointer to SS** | 00001111 | 10110010 | mod reg r/m | | 7 | 22 | b | h, i, j |
| **FLAG CONTROL** | | | | | | | | |
| **CLC = Clear Carry Flag** | 11111000 | | | | 2 | 2 | | |
| **CLD = Clear Direction Flag** | 11111100 | | | | 2 | 2 | | |
| **CLI = Clear Interrupt Enable Flag** | 11111010 | | | | 3 | 3 | | m |
| **CLTS = Clear Task Switched Flag** | 00001111 | 00000110 | | | 5 | 5 | c | l |
| **CMC = Complement Carry Flag** | 11110101 | | | | 2 | 2 | | |
| **LAHF = Load AH into Flag** | 10011111 | | | | 2 | 2 | | |
| **POPF = Pop Flags** | 10011101 | | | | 5 | 5 | b | h, n |
| **PUSHF = Push Flags** | 10011100 | | | | 4 | 4 | b | h |
| **SAHF = Store AH into Flags** | 10011110 | | | | 3 | 3 | | |
| **STC = Set Carry Flag** | 11111001 | | | | 2 | 2 | | |
| **STD = Set Direction Flag** | 11111001 | | | | 2 | 2 | | |
| **STI = Set Interrupt Enable Flag** | 11111011 | | | | 3 | 3 | | m |
| **ARITHMETIC** | | | | | | | | |
| **ADD = Add** | | | | | | | | |
| Register to Register | 000000dw | mod reg r/m | | | 2 | 2 | | |
| Register to Memory | 0000000w | mod reg r/m | | | 7 | 7 | b | h |
| Memory to Register | 0000001w | mod reg r/m | | | 6 | 6 | b | h |
| Immediate to Register/Memory | 100000sw | mod 000 r/m | immediate data | | 2/7 | 2/7 | b | h |
| Immediate to Accumulator (short form) | 0000010w | immediate data | | | 2 | 2 | | |
| **ADC = Add With Carry** | | | | | | | | |
| Register to Register | 000100dw | mod reg r/m | | | 2 | 2 | | |
| Register to Memory | 0001000w | mod reg r/m | | | 7 | 7 | b | h |
| Memory to Register | 0001001w | mod reg r/m | | | 6 | 6 | b | h |
| Immediate to Register/Memory | 100000sw | mod 010 r/m | immediate data | | 2/7 | 2/7 | b | h |
| Immediate to Accumulator (short form) | 0001010w | immediate data | | | 2 | 2 | | |
| **INC = Increment** | | | | | | | | |
| Register/Memory | 1111111w | mod 000 r/m | | | 2/6 | 2/6 | b | h |
| Register | 01000 reg | | | | 2 | 2 | | |
| **SUB = Subtract** | | | | | | | | |
| Register from Register | 001010dw | mod reg r/m | | | 2 | 2 | | |

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

| INSTRUCTION | FORMAT | CLOCK COUNT: Real Address Mode or Virtual 8086 Mode | CLOCK COUNT: Protected Virtual Address Mode | NOTES: Real Address Mode or Virtual 8086 Mode | NOTES: Protected Virtual Address Mode |
|---|---|---|---|---|---|
| **ARITHMETIC** (Continued) | | | | | |
| Register from Memory | 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 | 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 | 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 | 0 1 0 0 1 reg | 2 | 2 | | |
| **CMP = Compare** | | | | | |
| Register with Register | 0 0 1 1 1 0 d w \| mod reg r/m | 2 | 2 | | |
| Memory with Register | 0 0 1 1 1 0 0 w \| mod reg r/m | 5 | 5 | b | h |
| Register with Memory | 0 0 1 1 1 0 1 w \| mod reg r/m | 6 | 6 | b | h |
| Immediate with Register/Memory | 1 0 0 0 0 0 s w \| mod 1 1 1 r/m \| immediate data | 2/5 | 2/5 | b | h |
| Immediate with Accumulator (short form) | 0 0 1 1 1 1 0 w \| immediate data | 2 | 2 | | |
| **NEG = Change Sign** | 1 1 1 1 0 1 1 w \| mod 0 1 1 r/m | 2/6 | 2/6 | b | f |
| **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 | | 9–14/12-17 | 9–14/12–17 | b, d | d, h |
| -Word | | 9–22/12-25 | 9–22/12–25 | b, d | d, h |
| -Doubleword | | 9–38/12-41 | 9–38/12–41 | 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 | | 9–14/12-17 | 9–14/12–17 | b, d | d, h |
| -Word | | 9–22/12-25 | 9–22/12–25 | b, d | d, h |
| -Doubleword | | 9–38/12-41 | 9–38/12–41 | 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 | | 9–14/12-17 | 9–14/12–17 | b, d | d, h |
| -Word | | 9–22/12-25 | 9–22/12–25 | b, d | d, h |
| -Doubleword | | 9–38/12-41 | 9–38/12–41 | b, d | d, h |
| Register/Memory with Immediate to Register | 0 1 1 0 1 0 s 1 \| mod reg r/m \| immediate data | | | | |
| Multiplier-Byte | | 9–14/12-17 | 9–14/12–17 | b, d | d, h |
| -Word | | 9–22/12-25 | 9–22/12–25 | b, d | d, h |
| -Doubleword | | 9–38/12-41 | 9–38/12–41 | b, d | d, h |

**Table 8-1. 80386 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 | 1 1 1 1 0 1 1 w · mod 1 1 0 r/m | | | | |
| Divisor—Byte | | 14/17 | 14/17 | b,e | e,h |
| —Word | | 22/25 | 22/25 | b,e | e,h |
| —Doubleword | | 38/41 | 38/41 | b,e | e,h |
| **IDIV = Integer Divide (Signed)** | | | | | |
| Accumulator By Register/Memory | 1 1 1 1 0 1 1 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 | | 43/46 | 43/46 | b,e | e,h |
| **AAD = ASCII Adjust for Divide** | 1 1 0 1 0 1 0 1 · 0 0 0 0 1 0 1 0 | 19 | 19 | | |
| **AAM = ASCII Adjust for Multiply** | 1 1 0 1 0 1 0 0 · 0 0 0 0 1 0 1 0 | 17 | 17 | | |
| **CBW = Convert Byte to Word** | 1 0 0 1 1 0 0 0 | 3 | 3 | | |
| **CWD = Convert Word to Double Word** | 1 0 0 1 1 0 0 1 | 2 | 2 | | |
| **LOGIC** | | | | | |
| Shift Rotate Instructions | | | | | |
| Not Through Carry **(ROL, ROR, SAL, SAR, SHL,** and **SHR)** | | | | | |
| Register/Memory by 1 | 1 1 0 1 0 0 0 w · mod TTT r/m | 3/7 | 3/7 | b | h |
| Register/Memory by CL | 1 1 0 1 0 0 1 w · mod TTT r/m | 3/7 | 3/7 | b | h |
| Register/Memory by Immediate Count | 1 1 0 0 0 0 0 w · mod TTT r/m | 3/7 | 3/7 | b | h |
| Through Carry **(RCL** and **RCR)** | | | | | |
| Register/Memory by 1 | 1 1 0 1 0 0 0 w · mod TTT r/m | 9/10 | 9/10 | b | h |
| Register/Memory by CL | 1 1 0 1 0 0 1 w · mod TTT r/m | 9/10 | 9/10 | b | h |
| Register/Memory by Immediate Count | 1 1 0 0 0 0 0 w · mod TTT r/m | 9/10 | 9/10 | b | h |
| | **T T T Instruction** | | | | |
| | 0 0 0 ROL | | | | |
| | 0 0 1 ROR | | | | |
| | 0 1 0 RCL | | | | |
| | 0 1 1 RCR | | | | |
| | 1 0 0 SHL/SAL | | | | |
| | 1 0 1 SHR | | | | |
| | 1 1 1 SAR | | | | |
| **SHLD = Shift Left Double** | | | | | |
| Register/Memory by Immediate | 0 0 0 0 1 1 1 1 · 1 0 1 0 0 1 0 0 · mod reg r/m · immed 8-bit data | 3/7 | 3/7 | | |
| Register/Memory by CL | 0 0 0 0 1 1 1 1 · 1 0 1 0 0 1 0 1 · mod reg r/m | 3/7 | 3/7 | | |
| **SHRD = Shift Right Double** | | | | | |
| Register/Memory by Immediate | 0 0 0 0 1 1 1 1 · 1 0 1 0 1 1 0 0 · mod reg r/m · immed 8-bit data | 3/7 | 3/7 | | |
| Register/Memory by CL | 0 0 0 0 1 1 1 1 · 1 0 1 0 1 1 0 1 · mod reg r/m | 3/7 | 3/7 | | |
| **AND = And** | | | | | |
| Register to Register | 0 0 1 0 0 0 d w · mod reg r/m | 2 | 2 | | |

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

| INSTRUCTION | FORMAT | CLOCK COUNT: Real Address Mode or Virtual 8086 Mode | CLOCK COUNT: Protected Virtual Address Mode | NOTES: Real Address Mode or Virtual 8086 Mode | NOTES: Protected Virtual Address Mode |
|---|---|---|---|---|---|
| **ARITHMETIC** (Continued) | | | | | |
| Register 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 s w \| mod 1 0 0 r/m \| immediate data | 2/7 | 2/7 | b | h |
| Immediate to Accumulator (Short Form) | 0 0 1 0 0 1 0 w \| immediate data | 2 | 2 | | |
| **TEST = And Function to Flags, No Result** | | | | | |
| Register/Memory and Register | 1 0 0 0 0 1 0 w \| mod reg r/m | 2/5 | 2/5 | b | h |
| Immediate Data and Register/Memory | 1 1 1 1 0 1 1 w \| mod 0 0 0 r/m \| immediate data | 2/5 | 2/5 | b | h |
| Immediate Data and Accumulator (Short Form) | 1 0 1 0 1 0 0 w \| immediate data | 2 | 2 | | |
| **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 s w \| mod 0 0 1 r/m \| immediate data | 2/7 | 2/7 | b | h |
| Immediate to Accumulator (Short Form) | 0 0 0 0 1 1 0 w \| immediate data | 2 | 2 | | |
| **XOR = Exclusive Or** | | | | | |
| Register to Register | 0 0 1 1 0 0 d w \| mod reg r/m | 2 | 2 | | |
| Register to Memory | 0 0 1 1 0 0 0 w \| mod reg r/m | 7 | 7 | b | h |
| Memory to Register | 0 0 1 1 0 0 1 w \| mod reg r/m | 6 | 6 | b | h |
| Immediate to Register/Memory | 1 0 0 0 0 0 s w \| mod 1 1 0 r/m \| immediate data | 2/7 | 2/7 | b | h |
| Immediate to Accumulator (Short Form) | 0 0 1 1 0 1 0 w \| immediate data | 2 | 2 | | |
| **NOT = Invert Register/Memory** | 1 1 1 1 0 1 1 w \| mod 0 1 0 r/m | 2/6 | 2/6 | b | h |
| **STRING MANIPULATION** | | | | | |
| **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 | 8 | 8 | b | 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 | 7 | 7 | b | 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 by Count in CX | | | | | |
| **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 |

**Table 8-1. 80386 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 |
| **STRING MANIPULATION** (Continued) | | | | | |
| **REPNE CMPS = Compare String** | | | | | |
| (Find Match) | 11110010 1010011w | 5+9n | 5+9n | b | h |
| **REP INS = Input String** | 11110010 0110110w | 6+6n | 6+6n | b | h, m |
| **REP LODS = Load String** | 11110010 1010110w | 5+6n | 5+6n | b | h |
| **REP MOVS = Move String** | 11110010 1010010w | 7+4n | 7+4n | b | h |
| **REP OUTS = Output String** | 11110010 0110111w | 5+5n | 5+5n | b | h, m |
| **REPE SCAS = Scan String** | | | | | |
| (Find Non-AL/AX/EAX) | 11110011 1010111w | 5+8n | 5+8n | b | h |
| **REPNE SCAS = Scan String** | | | | | |
| (Find AL/AX/EAX) | 11110010 1010111w | 5+8n | 5+8n | b | h |
| **REP STOS = Store String** | 11110010 1010101w | 5+5n | 5+5n | b | h |
| **BIT MANIPULATION** | | | | | |
| **BSF = Scan Bit Forward** | 00001111 10111100 mod reg r/m | 10+3n | 10+3n | b | h |
| **BSR = Scan Bit Reverse** | 00001111 10111100 mod reg r/m | 10+3n | 10+3n | b | h |
| **BT = Test Bit** | | | | | |
| Register/Memory, Immediate | 00001111 10111010 mod 100 r/m immed 8-bit data | 3/6 | 3/6 | b | h |
| Register/Memory, Register | 00001111 10100011 mod reg r/m | 3/12 | 3/12 | b | h |
| **BTC = Test Bit and Complement** | | | | | |
| Register/Memory, Immediate | 00001111 10111010 mod 111 r/m immed 8-bit data | 6/8 | 6/8 | b | h |
| Register/Memory, Register | 00001111 10111011 mod reg r/m | 6/13 | 6/13 | b | h |
| **BTR = Test Bit and Reset** | | | | | |
| Register/Memory, Immediate | 00001111 10111010 mod 110 r/m immed 8-bit data | 6/8 | 6/8 | b | h |
| Register/Memory, Register | 00001111 10110011 mod reg r/m | 6/13 | 6/13 | b | h |
| **BTS = Test Bit and Set** | | | | | |
| Register/Memory, Immediate | 00001111 10111010 mod 010 r/m immed 8-bit data | 6/8 | 6/8 | b | h |
| Register/Memory, Register | 00001111 10101011 mod reg r/m | 6/13 | 6/13 | b | h |
| **BIT STRING MANIPULATION** | | | | | |
| **IBTS = Insert Bit String** | 00001111 10100111 | 12/19 | 12/19 | b | h |
| **XBTS = Extract Bit String** | 00001111 10100110 | 6/13 | 6/13 | b | h |
| **CONTROL TRANSFER** | | | | | |
| **CALL = Call** | | | | | |
| Direct Within Segment | 11101000 full displacement | 7+m | 7+m | b | r |
| Register/Memory | | | | | |
| Indirect Within Segment | 11111111 mod 010 r/m | 7+m/ 10+m | 7+m/ 10+m | b | h, r |
| Direct Intersegment | 10011010 offset, selector | 17+m | 35 | b | j,k,r |

**Table 8-1. 80386 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 | | | | 58 | | h,j,k,r |
| Via Call Gate to Different Privilege Level, (No Parameters) | | | | 108 | | h,j,k,r |
| Via Call Gate to Different Privilege Level, (x Parameters) | | | | 111+4x | | h,j,k,r |
| From 286 Task to 286 TSS | | | | 235 | | h,j,k,r |
| From 286 Task to 386 TSS | | | | 265 | | h,j,k,r |
| From 286 Task to Virtual 8086 Mode | | | | 145 | | h,j,k,r |
| From 386 Task to 286 TSS | | | | 245 | | h,j,k,r |
| From 386 Task to 386 TSS | | | | 275 | | h,j,k,r |
| From 386 Task to Virtual 8086 Mode | | | | 155 | | h,j,k,r |
| Indirect Intersegment | 11111111 | mod 0 1 1 r/m | 22+m | 40 | b | h,j,k,r |
| Protected Mode Only (Indirect Intersegment) | | | | | | |
| Via Call Gate to Same Privilege Level | | | | 63 | | h,j,k,r |
| Via Call Gate to Different Privilege Level, (No Parameters) | | | | 113 | | h,j,k,r |
| Via Call Gate to Different Privilege Level, (x Parameters) | | | | 116+4x | | h,j,k,r |
| From 286 Task to 286 TSS | | | | 240 | | h,j,k,r |
| From 286 Task to 386 TSS | | | | 270 | | h,j,k,r |
| From 286 Task to Virtual 8086 Mode | | | | 150 | | h,j,k,r |
| From 386 Task to 286 TSS | | | | 250 | | h,j,k,r |
| From 386 Task to 386 TSS | | | | 280 | | h,j,k,r |
| From 386 Task to Virtual 8086 Mode | | | | 160 | | h,j,k,r |
| **JMP = Unconditional Jump** | | | | | | |
| Short | 11101001 | 8-bit displacement | 7+m | 7+m | | r |
| Direct within Segment | 11101001 | full displacement | 7+m | 7+m | | r |
| Register/Memory Indirect within Segment | 11111111 | mod 1 0 0 r/m | 7+m/ 10+m | 7+m/ 10+m | b | h,r |
| Direct Intersegment | 11101010 | offset, selector | 12+m | 23 | | j,k,r |
| Protected Mode Only (Direct Intersegment) | | | | | | |
| Via Call Gate to Same Privilege Level | | | | 39+m | | h,j,k,r |
| From 286 Task to 286 TSS | | | | 223 | | h,j,k,r |
| From 286 Task to 386 TSS | | | | 253 | | h,j,k,r |
| From 286 Task to Virtual 8086 Mode | | | | 133 | | h,j,k,r |
| From 386 Task to 286 TSS | | | | 233 | | h,j,k,r |
| From 386 Task to 386 TSS | | | | 263 | | h,j,k,r |
| From 386 Task to Virtual 8086 Mode | | | | 143 | | h,j,k,r |
| Indirect Intersegment | 11111111 | mod 1 0 1 r/m | 17+m | 28 | b | h,j,k,r |
| Protected Mode Only (Indirect Intersegment) | | | | | | |
| Via Call Gate to Same Privilege Level | | | | 49 | | h,j,k,r |
| From 286 Task to 286 TSS | | | | 228 | | h,j,k,r |
| From 286 Task to 386 TSS | | | | 258 | | h,j,k,r |
| From 286 Task to Virtual 8086 Mode | | | | 143 | | h,j,k,r |
| From 386 Task to 286 TSS | | | | 238 | | h,j,k,r |
| From 386 Task to 386 TSS | | | | 268 | | h,j,k,r |
| From 386 Task to Virtual 8086 Mode | | | | 148 | | h,j,k,r |

**Table 8-1. 80386 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 | 11000011 | | | 10 + m | 10 + m | b | g, h, r |
| Within Segment Adding Immediate to SP | 11000010 | 16-bit displ | | 10 + m | 10 + m | b | g, h, r |
| Intersegment | 11001011 | | | 18 + m | 35 | b | g, h, j, k, r |
| Intersegment Adding Immediate to SP | 11001010 | 16-bit displ | | 18 + m | 35 | b | g, h, j, k, r |
| Protected Mode Only (RET): | | | | | | | |
| to Different Privilege Level | | | | | | | |
| Intersegment | | | | | 77 | | h, j, k, r |
| Intersegment Adding Immediate to SP | | | | | 77 | | h, j, k, r |
| **CONDITIONAL JUMPS** | | | | | | | |
| NOTE: Times Are Jump "Taken or Not Taken" | | | | | | | |
| **JO = Jump on Overflow** | | | | | | | |
| 8-Bit Displacement | 01110000 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000000 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNO = Jump on Not Overflow** | | | | | | | |
| 8-Bit Displacement | 01110001 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000001 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JB/JNAE = Jump on Below/Not Above or Equal** | | | | | | | |
| 8-Bit Displacement | 01110010 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000010 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNB/JAE = Jump on Not Below/Above or Equal** | | | | | | | |
| 8-Bit Displacement | 01110011 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000011 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JE/JZ = Jump on Equal/Zero** | | | | | | | |
| 8-Bit Displacement | 01110100 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000100 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNE/JNZ = Jump on Not Equal/Not Zero** | | | | | | | |
| 8-Bit Displacement | 01110101 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000101 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JBE/JNA = Jump on Below or Equal/Not Above** | | | | | | | |
| 8-Bit Displacement | 01110110 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000110 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNBE/JA = Jump on Not Below or Equal/Above** | | | | | | | |
| 8-Bit Displacement | 01110111 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10000111 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JS = Jump on Sign** | | | | | | | |
| 8-Bit Displacement | 01111000 | 8-bit displ | | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 | 10001000 | full displacement | 7 + m or 3 | 7 + m or 3 | | r |

**Table 8-1. 80386 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 JUMPS** (Continued) | | | | | |
| **JNS = Jump on Not Sign** | | | | | |
| 8-Bit Displacement | 01111001 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001001 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JP/JPE = Jump on Parity/Parity Even** | | | | | |
| 8-Bit Displacement | 01111010 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001010 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNP/JPO = Jump on Not Parity/Parity Odd** | | | | | |
| 8-Bit Displacement | 01111011 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001011 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JL/JNGE = Jump on Less/Not Greater or Equal** | | | | | |
| 8-Bit Displacement | 01111100 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001100 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNL/JGE = Jump on Not Less/Greater or Equal** | | | | | |
| 8-Bit Displacement | 01111101 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001101 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JLE/JNG = Jump on Less or Equal/Not Greater** | | | | | |
| 8-Bit Displacement | 01111110 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001110 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JNLE/JG = Jump on Not Less or Equal/Greater** | | | | | |
| 8-Bit Displacement | 01111111 8-bit displ | 7 + m or 3 | 7 + m or 3 | | r |
| Full Displacement | 00001111 10001111 full displacement | 7 + m or 3 | 7 + m or 3 | | r |
| **JCXZ = Jump on CX Zero** | 11100011 8-bit displ | 9 + m or 5 | 9 + m or 5 | | r |
| **JECXZ = Jump on ECX Zero** | 11100011 8-bit displ | 9 + m or 5 | 9 + m or 5 | | r |
| (Operand Size Prefix Differentiates JCXZ from JECXZ) | | | | | |
| **LOOP = Loop CX Times** | 11100010 8-bit displ | 11 + m | 11 + m | | r |
| **LOOPZ/LOOPE = Loop with Zero/Equal** | 11100001 8-bit displ | 11 + m | 11 + m | | r |
| **LOOPNZ/LOOPNE = Loop While Not Zero** | 11100000 8-bit displ | 11 + m | 11 + m | | r |
| **CONDITIONAL BYTE SET** | | | | | |
| NOTE: Times Are Register/Memory | | | | | |
| **SETO = Set Byte on Overflow** | | | | | |
| To Register/Memory | 00001111 10010000 mod 000 r/m | 4/5 | 4/5 | | h |
| **SETNO = Set Byte on Not Overflow** | | | | | |
| To Register/Memory | 00001111 10010001 mod 000 r/m | 4/5 | 4/5 | | h |
| **SETB/SETNAE = Set Byte on Below/Not Above or Equal** | | | | | |
| To Register/Memory | 00001111 10010010 mod 000 r/m | 4/5 | 4/5 | | h |

**Table 8-1. 80386 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 | 0 0 0 0 1 1 1 1 | 1 0 0 1 0 0 1 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETE/SETZ = Set Byte on Equal/Zero** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 0 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNE/SETNZ = Set Byte on Not Equal/Not Zero** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 0 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETBE/SETNA = Set Byte on Below or Equal/Not Above** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 1 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNBE/SETA = Set Byte on Not Below or Equal/Above** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 0 1 1 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETS = Set Byte on Sign** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 0 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNS = Set Byte on Not Sign** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 0 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETP/SETPE = Set Byte on Parity/Parity Even** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 1 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNP/SETPO = Set Byte on Not Parity/Parity Odd** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 0 1 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETL/SETNGE = Set Byte on Less/Not Greater or Equal** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 0 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNL/SETGE = Set Byte on Not Less/Greater or Equal** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 0 1 1 1 1 1 0 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETLE/SETNG = Set Byte on Less or Equal/Not Greater** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 1 0 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **SETNLE/SETG = Set Byte on Not Less or Equal/Greater** | | | | | | | |
| To Register/Memory | 0 0 0 0 1 1 1 1 | 1 0 0 1 1 1 1 1 | mod 0 0 0 r/m | 4/5 | 4/5 | | h |
| **ENTER = Enter Procedure** | 1 1 0 0 1 0 0 0 | 16-bit displacement, 8-bit level | | | | | |
| L = 0 | | | | 10 | 10 | b, g | g, h |
| L = 1 | | | | 12 | 12 | b, g | g, h |
| L > 1 | | | | 15 + 4(n − 1) | 15 + 4(n − 1) | b, g | g, h |
| **LEAVE = Leave Procedure** | 1 1 0 0 1 0 0 1 | | | 4 | 4 | b, g | g, h |

**Table 8-1. 80386 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 | 11001101 \| type | 37 | | b, f, g | |
| Type 3 | 11001100 | 33 | | b, f, g | |
| **INTO = Interrupt 4 if Overflow Flag Set** | 11001110 | | | | |
| If OF = 1 | | 35 | | b, e, g | |
| If OF = 0 | | 3 | 3 | b, e, g | |
| Bound = Interrupt 5 if Detect Value Out of Range | 01100010 \| mod reg r/m | | | | |
| If Out of Range | | 44 | | b, e | e, g, h, j, k, r |
| If In Range | | 10 | 10 | b, e | e, g, h, j, k, r |
| Protected Mode Only (INT) | | | | | |
| INT: Type Specified | | | | | |
| Via Interrupt or Trap Gate to Same Privilege Level | | | 59 | | f, g, j, k, r |
| Via Interrupt or Trap Gate to Different Privilege Level | | | 121 | | f, g, j, k, r |
| From 286 Task to 286 TSS via Task Gate | | | 247 | | f, g, j, k, r |
| From 286 Task to 386 TSS via Task Gate | | | 277 | | f, g, j, k, r |
| From 268 Task to virt 8086 md via Task Gate | | | 157 | | f, g, j, k, r |
| From 386 Task to 286 TSS via Task Gate | | | 257 | | f, g, j, k, r |
| From 386 Task to 386 TSS via Task Gate | | | 287 | | f, g, j, k, r |
| From 368 Task to virt 8086 md via Task Gate | | | 167 | | f, g, j, k, r |
| From virt 8086 md to 286 TSS via Task Gate | | | 257 | | f, g, j, k, r |
| From virt 8086 md to 386 TSS via Task Gate | | | 287 | | f, g, j, k, r |
| INT: TYPE 3 | | | | | |
| Via Interrupt or Trap Gate to Same Privilege Level | | | 59 | | f, g, j, k, r |
| Via Interrupt or Trap Gate to Different Privilege Level | | | 121 | | f, g, j, k, r |
| From 286 Task to 286 TSS via Task Gate | | | 243 | | f, g, j, k, r |
| From 286 Task to 386 TSS via Task Gate | | | 273 | | f, g, j, k, r |
| From 268 Task to Virt 8086 md via Task Gate | | | 157 | | f, g, j, k, r |
| From 386 Task to 286 TSS via Task Gate | | | 253 | | f, g, j, k, r |
| From 386 Task to 386 TSS via Task Gate | | | 283 | | f, g, j, k, r |
| From 368 Task to Virt 8086 md via Task Gate | | | 163 | | f, g, j, k, r |
| From Virt 8086 md to 286 TSS via Task Gate | | | 253 | | f, g, j, k, r |
| From Virt 8086 md to 386 TSS via Task Gate | | | 283 | | f, g, j, k, r |
| INTO: | | | | | |
| Via Interrupt or Trap Grate to Same Privilege Level | | | 59 | | f, g, j, k, r |
| Via Interrupt or Trap Gate to Different Privilege Level | | | 121 | | f, g, j, k, r |
| From 286 Task to 286 TSS via Task Gate | | | 245 | | f, g, j, k, r |
| From 286 Task to 386 TSS via Task Gate | | | 275 | | f, g, j, k, r |
| From 268 Task to virt 8086 md via Task Gate | | | 155 | | f, g, j, k, r |
| From 386 Task to 286 TSS via Task Gate | | | 255 | | f, g, j, k, r |
| From 386 Task to 386 TSS via Task Gate | | | 285 | | f, g, j, k, r |
| From 368 Task to virt 8086 md via Task Gate | | | 165 | | f, g, j, k, r |
| From virt 8086 md to 286 TSS via Task Gate | | | 255 | | f, g, j, k, r |
| From virt 8086 md to 386 TSS via Task Gate | | | 285 | | f, g, j, k, r |

**Table 8-1. 80386 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 | | | | | 59 | | f, g, j, k, r |
| Via Interrupt or Trap Gate to Different Privilege Level | | | | | 121 | | f, g, j, k, r |
| From 286 Task to 286 TSS via Task Gate | | | | | 254 | | f, g, j, k, r |
| From 286 Task to 386 TSS via Task Gate | | | | | 284 | | f, g, j, k, r |
| From 268 Task to virt 8086 Mode via Task Gate | | | | | 164 | | f, g, j, k, r |
| From 386 Task to 286 TSS via Task Gate | | | | | 264 | | f, g, j, k, r |
| From 386 Task to 386 TSS via Task Gate | | | | | 294 | | f, g, j, k, r |
| From 368 Task to virt 8086 Mode via Task Gate | | | | | 174 | | f, g, j, k, r, |
| From virt 8086 Mode to 286 TSS via Task Gate | | | | | 264 | | f, g, j, k, r |
| From virt 8086 Mode to 386 TSS via Task Gate | | | | | 294 | | f, g, j, k, r |
| **INTERRUPT RETURN** | | | | | | | |
| **IRET = Interrupt Return** | 1 1 0 0 1 1 1 1 | | | 22 | | | g, h, j, k, r |
| Protected Mode Only (IRET) | | | | | | | |
| To the Same Privilege Level | | | | | 38 | | g, h, j, k, r |
| To Different Privilege Level | | | | | 82 | | g, h, j, k, r |
| From 286 Task to 286 TSS | | | | | 232 | | h, j, k, r |
| From 286 Task to 386 TSS | | | | | 265 | | h, j, k, r |
| From 286 Task to Virtual 8086 Mode | | | | | 132 | | h, j, k, r |
| From 386 Task to 286 TSS | | | | | 271 | | h, j, k, r |
| From 386 Task to 386 TSS | | | | | 142 | | h, j, k, r |
| From 386 Task to Virtual 8086 Mode | | | | | 120 | | h, j, k, r |
| **PROCESSOR CONTROL** | | | | | | | |
| **HLT = HALT** | 1 1 1 1 0 1 0 0 | | | 5 | 5 | | l |
| **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 | 0 0 eee reg | 10/4/5 | 10/4/5 | | l |
| Register From CR0–3 | 0 0 0 0 1 1 1 1 | 0 0 1 0 0 0 0 0 | 0 0 eee reg | 6 | 6 | | l |
| 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 | | l |
| 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 | | l |
| 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 | | l |
| 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 | | l |
| 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 | | l |
| 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 | | l |
| **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 8-1. 80386 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 |
| **PROCESSOR EXTENSION INSTRUCTIONS** | | | | | | | |
| Processor Extension Escape | 11011TTT | mod LLL r/m | | See 80287/80387 data sheets for clock counts | | g | g, q |
| | TTT and LLL bits are opcode information for coprocessor. | | | | | | |
| **PREFIX BYTES** | | | | | | | |
| **Address Size Prefix** | 01100111 | | | 0 | 0 | | |
| **LOCK = Bus Lock Prefix** | 11110000 | | | 0 | 0 | | m |
| **Operand Size Prefix** | 01100110 | | | 0 | 0 | | |
| Segment Override Prefix | | | | | | | |
| Segment Override Prefix | | | | | | | |
| **CS:** | 00101110 | | | 0 | 0 | | |
| **DS:** | 00111110 | | | 0 | 0 | | |
| **ES:** | 00100110 | | | 0 | 0 | | |
| **FS:** | 01100100 | | | 0 | 0 | | |
| **GS:** | 01100101 | | | 0 | 0 | | |
| **SS:** | 00110110 | | | 0 | 0 | | |
| **PROTECTION CONTROL** | | | | | | | |
| **ARPL = Adjust Requested Privilege Level** | | | | | | | |
| From Register/Memory | 01100011 | mod reg r/m | | N/A | 20/21 | a | g, h |
| **LAR = Load Access Rights** | | | | | | | |
| From Register/Memory | 00001111 | 00000010 | mod reg r/m | N/A | 15/16 | a | h, j, p |
| **LGDT = Load Global Descriptor** | | | | | | | |
| Table Register | 00001111 | 00000001 | mod 010 r/m | 11 | 11 | b, c | h, l |
| **LIDT = Load Interrupt Descriptor** | | | | | | | |
| Table Register | 00001111 | 00000001 | mod 011 r/m | 11 | 11 | b, c | h, l |
| **LLDT = Load Local Descriptor** | | | | | | | |
| Table Register to Register/Memory | 00001111 | 00000000 | mod 010 r/m | N/A | 20/24 | a | h, j, l |
| **LMSW = Load Machine Status Word** | | | | | | | |
| From Register/Memory | 00001111 | 00000001 | mod 110 r/m | 10/13 | 10/13 | b, c | h, l |
| **LSL = Load Segment Limit** | | | | | | | |
| From Register/Memory | 00001111 | 00000011 | mod reg r/m | | | | |
| Byte-Granular Limit | | | | N/A | 20/21 | a | h, j, p |
| Page-Granular Limit | | | | N/A | 25/26 | a | h, j, p |
| **LTR = Load Task Register** | | | | | | | |
| From Register/Memory | 00001111 | 00000000 | mod 001 r/m | N/A | 23/27 | a | h, j, l |
| **SGDT = Store Global Descriptor** | | | | | | | |
| Table Register | 00001111 | 00000001 | mod 000 r/m | 9 | 9 | b, c | h |

**Table 8-1. 80386 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 |
| SIDT | = Store Interrupt Descriptor Table Register | 00001111 | 00000001 | mod 001 r/m | 9 | 9 | b, c | h |
| SLDT | = Store Local Descriptor Table Register To Register/Memory | 00001111 | 00000000 | mod 000 r/m | N/A | 2/2 | a | h |
| SMSW | = Store Machine Status Word | 00001111 | 00000001 | mod 100 r/m | 10/13 | 10/13 | b, c | h, l |
| STR | = Store Task Register To Register/Memory | 00001111 | 00000000 | mod 001 r/m | N/A | 2/2 | a | h |
| VERR | = Verify Read Accesss Register/Memory | 00001111 | 00000000 | mod 100 r/m | N/A | 10/11 | a | h, j, p |
| VERW | = Verify Write Accesss | 00001111 | 00000000 | mod 101 r/m | N/A | 15/16 | a | h, j, p |

## INSTRUCTION NOTES FOR TABLE 8-1

**Notes a through c apply to 80386 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 (general protection) will occur in Real Mode if a 16-bit or 32-bit operand reference is made that partially or fully extends beyond the maximum segment limit, FFFFH.

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

d. the iAPX 386 uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

Actual Clock = if $m <> 0$ then max ($[\log_2 |m|]$, 3) + 6 clocks:

if $m = 0$ then 9 clocks (where m is the multiplier)

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

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

**Notes h through r apply to 80386 Protected Virtual Address Mode only:**

h. Exception 13 (general protection violation) will occur if the memory operand 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 overrun 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 (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 overrun 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 occurs if CPL is greater than 0 (0 is the most privileged level).

m. An exception 13 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 (CRO) cannot be reset by this instruction. Use MOV into CRO if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand do 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 (general protection exception) will occur before the ESC instruction is executed. An exception 12 (stack segment overrun) 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 (general protection violation) will occur.

## 8.2 INSTRUCTION ENCODING

### 8.2.1 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.

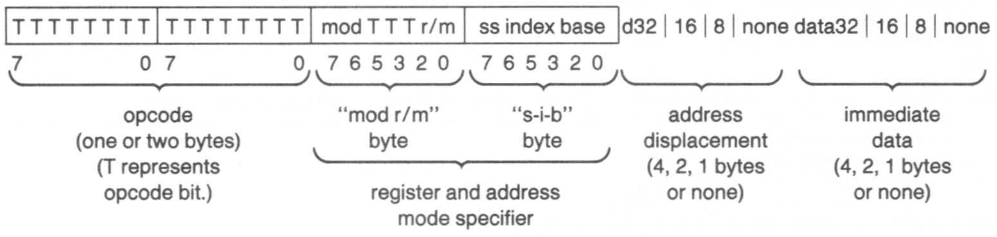
Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode bytes(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 8-1 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 8-2 is a complete list of all fields appearing in the 80386 instruction set. Further ahead, following Table 8-2, are detailed tables for each field.



TTTTTTTT | TTTTTTTT | mod TTT r/m | ss index base | d32 | 16 | 8 | none data32 | 16 | 8 | none
7 0 7 0 7 6 5 3 2 0 7 6 5 3 2 0
opcode (one or two bytes) (T represents opcode bit.)
"mod r/m" byte
"s-i-b" byte
register and address mode specifier
address displacement (4, 2, 1 bytes or none)
immediate data (4, 2, 1 bytes or none)

**Figure 8-1. General Instruction Format**

**Table 8-2. Fields within 80386 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 |
| sreg | Segment Register Specifier for CS, SS, DS, ES | 2 |
| sreg | Segment Register Specifier for CS, SS, DS, ES, FS, GS | 3 |
| tttn | For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated | 4 |

### 8.2.2 32-Bit Extensions of the Instruction Set

With the 80386, the 86/186/286 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 default 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 or 0 is assumed internally by the 80386 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 80386 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.

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.

### 8.2.3 Encoding of Instruction Fields

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

#### 8.2.3.1 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 |

#### 8.2.3.2 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 |

#### 8.2.3.3 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 80386 FS and GS segment registers to be specified.

**2-Bit sreg Field**

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

**3-Bit sreg Field**

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

#### 8.2.3.4 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" (scaled index) 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.

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 four pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

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

| mod r/m | Effective Address |
|---|---|
| 00 000 | DS:[BX+SI] |
| 00 001 | DS:[BX+DI] |
| 00 010 | SS:[BP+SI] |
| 00 011 | SS:[BP+DI] |
| 00 100 | DS:[SI] |
| 00 101 | DS:[DI] |
| 00 110 | DS:d16 |
| 00 111 | DS:[BX] |
| 01 000 | DS:[BX+SI+d8] |
| 01 001 | DS:[BX+DI+d8] |
| 01 010 | SS:[BP+SI+d8] |
| 01 011 | SS:[BP+DI+d8] |
| 01 100 | DS:[SI+d8] |
| 01 101 | DS:[DI+d8] |
| 01 110 | DS:[BX+d8] |
| 01 111 | SS:[BP+d8] |

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

**Register Specified by r/m During 16-Bit Data Operations**

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

**Register Specified by r/m During 32-Bit Data Operations**

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

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

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

| mod r/m | Effective Address |
|---|---|
| 10 000 | DS:[EBX+ESI+d32] |
| 10 001 | DS:[EBX+EDI+d32] |
| 10 010 | SS:[EBP+ESI+d32] |
| 10 011 | SS:[EBP+EDI+d32] |
| 10 100 | s-i-b is present |
| 10 101 | DS:[EDI+d32] |
| 10 110 | DS:[EBX+d32] |
| 10 111 | SS:[EBP+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 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:[EAX + (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:[EAX + (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 |

### 8.2.3.5 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<br>"reg" Field Indicates Source Operand;<br>"mod r/m" or "mod ss index base" Indicates Destination Operand |
| 1 | Register <- - Register/Memory<br>"reg" Field Indicates Destination Operand;<br>"mod r/m" or "mod ss index base" Indicates Source Operand |

### 8.2.3.6 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 |

### 8.2.3.7 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 |

### 8.2.3.8 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 | |

intel

# DOMESTIC SALES OFFICES

**ALABAMA**

Intel Corp.
5015 Bradford Drive
Suite 2
Huntsville 35805
Tel: (205) 830-4010

**ARIZONA**

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

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

**CALIFORNIA**

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

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

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

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

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8086
TWX: 910-339-9279
910-338-0255

**COLORADO**

Intel Corp.
3300 Mitchell Lane, Suite 210
Boulder 80301
Tel: (303) 442-8088

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

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

**CONNECTICUT**

Intel Corp.
26 Mill Plain Road
Danbury 06810
Tel: (203) 748-3130
TWX: 710-456-1199

EMC Corp.
222 Summer Street
Stamford 06901
Tel: (203) 327-2934

**FLORIDA**

Intel Corp.
242 N. Westmonte Drive
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

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

**FLORIDA (Cont'd)**

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33702
Tel: (813) 577-2413

**GEORGIA**

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

**ILLINOIS**

Intel Corp.*
300 N. Martingale Road, Suite 400
Schaumburg 60172
Tel: (312) 310-8031

**INDIANA**

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

**IOWA**

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

**KANSAS**

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

**LOUISIANA**

Industrial Digital Systems Corp.
Tel: (504) 899-1654

**MARYLAND**

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

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

**MASSACHUSETTS**

Intel Corp.*
Westford Corp. Center
3 Carlisle Road
Westford 01886
Tel: (617) 629-3222
TWX: 710-343-6333

**MICHIGAN**

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

**MINNESOTA**

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

**MISSOURI**

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

**NEW JERSEY**

Intel Corp.*
Raritan Plaza III
Raritan Center
Edison 08837
Tel: (201) 225-3000
TWX: 710-480-6238

**NEW MEXICO**

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

**NEW YORK**

Intel Corp.*
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

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

Intel Corp.*
211 White Spruce Boulevard
Rochester 14623
Tel: (716) 424-1050
TWX: 510-253-7391

T-Squared
6443 Ridings Road
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
7353 Pittsford-Victor Road
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

**NORTH CAROLINA**

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

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

**OHIO**

Intel Corp.*
6500 Poe Avenue
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg., No. 300
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-2736
TWX: 810-427-9298

**OKLAHOMA**

Intel Corp.
4157 S. Harvard Avenue
Suite 123
Tulsa 74135
Tel: (918) 749-8688

**OREGON**

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

**PENNSYLVANIA**

Intel Corp.
1513 Cedar Cliff Drive
Camphill 17011
Tel: (717) 737-5035

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

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

Q.E.D. Electronics
139 Terwood Road
Box T
Willow Grove 19090
Tel: (215) 657-5600

**PUERTO RICO**

Intel Microprocessor Corp.
South Industrial Park
Las Piedras 00671
Tel: (809) 733-3030

**TEXAS**

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

Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

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

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713)988-9421

**UTAH**

Intel Corp.
5201 Green Street
Suite 290
Murray 84123
Tel: (801) 263-8051

**VIRGINIA**

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

**WASHINGTON**

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

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

**WISCONSIN**

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Chancellory Park I
Brookfield 53005
Tel: (414) 784-8087

## CANADA

**BRITISH COLUMBIA**

Intel Semiconductor of Canada, Ltd.
301-2245 W. Broadway
Vancouver V6K 2E4
Tel: (604) 738-6522

**ONTARIO**

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

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

**QUEBEC**

Intel Semiconductor of Canada, Ltd.
620 St. Jean Blvd.
Pointe Claire H9R 3K3
Tel: (514) 694-9130
TWX: 514-694-9134

*Field Application Location

# DOMESTIC DISTRIBUTORS

## ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

†Pioneer Electronics
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

## ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5100
TWX: 910-950-0077

Kierulff Electronics
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

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

## CALIFORNIA

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (818) 701-7500
TWX: 910-493-2086

Arrow Electronics
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TLX: 888064

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
20501 Plummer Street
Chatsworth 91311
Tel: (818) 700-6271
TWX: 910-494-2207

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

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

Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

†Hamilton Electro Sales
10912 W. Washington Boulevard
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

## CALIFORNIA (Cont'd)

Hamilton Electro Sales
9650 De Soto Avenue
Chatsworth 91311
Tel: (818) 700-6500

Kierulff Electronics
10824 Hope Street
Cypress 90430
Tel: (714) 220-6300

Kierulff Electronics, Inc.
1180 Murphy Avenue
San Jose 95131
Tel: (408) 947-3471
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

†Kierulff Electronics, Inc.
5650 Jillson Street
Commerce 90040
Tel: (213) 725-0325
TWX: 910-580-3666

Wyle Distribution Group
26560 Agoura Street
Calabasas 91302
Tel: (818) 880-9000
TWX: 818-372-0232

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 843-9953
TWX: 910-595-1572

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

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

Wyle Military
17810 Teller Avenue
Irvine 92750
Tel: (714) 851-9958
TWX: 310-371-9127

Wyle Systems
7382 Lampson Avenue
Garden Grove 92641
Tel: (714) 851-9953
TWX: 910-595-2642

## COLORADO

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

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

## CONNECTICUT

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

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

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

## FLORIDA

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

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W., Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

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

†Hamilton/Avnet Electronics
3197 Tech. Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

Hamilton/Avnet Electronics
6947 University Boulevard
Winterpark 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer Electronics
221 N. Lake Boulevard
Suite 412
Alta Monte Springs 32701
Tel: (305) 834-9090
TWX: 810-853-0284

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

## GEORGIA

†Arrow Electronics, Inc
3155 Northwoods Parkway, Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

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

Pioneer Electronics
5835B Peachtree Corners E
Norcross 30092
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

## ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

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

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

## INDIANA

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

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

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

## KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

## MARYLAND

Arrow Electronics, Inc.
8300 Gulford Road #H
Rivers Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350
TWX: 710-828-9702

†Pioneer Electronics
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

## MASSACHUSETTS

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

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel: (617) 935-9700
TWX: 710-393-0382

Pioneer Northeast Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

## MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

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

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

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

## MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer Electronics
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

## MISSOURI

Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

†Microcomputer System Technical Demonstrator Centers

# DOMESTIC DISTRIBUTORS

### NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03104
Tel: (603) 624-9400

### NEW JERSEY

†Arrow Electronics, Inc.
6000 Lincoln East
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

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

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 710-734-4388

†Pioneer Northeast Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

### NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

### NEW YORK

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

†Arrow Electronics, Inc.
3375 Brighton-Henrietta Townline Road
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-2641
TWX: 710-541-1560

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†Pioneer Northeast Electronics
1806 Vestal Parkway East
Vestal 13850
Tel: (607) 748-8211
TWX: 510-252-0893

### NEW YORK (Cont'd)

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

Pioneer Northeast Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

†MTI Systems Sales
38 Harbor Park Drive
P.O. Box 271
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

### NORTH CAROLINA

Arrow Electronics, Inc.
5240 Greendairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer Electronics
9801 A-Southern Pine Boulevard
Charlotte 28210
Tel: (704) 524-8188
TWX: 810-621-0366

### OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

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

### OKLAHOMA

Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

### OREGON

†Almac Electronics Corporation
1885 N.W. 169th Place
Beaverton 97006
Tel: (503) 629-8090
TWX: 910-467-8743

Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

### PENNSYLVANIA

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

### PENNSYLVANIA (Cont'd)

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

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

### TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
2401 Rutland
Austin 78757
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
8750 West Park
Hosuton 77063
Tel: (713) 780-1771
TWX: 910-881-5523

Pioneer Electronics
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

### UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel: (801) 974-9953

### WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

### WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

### WISCONSIN (Cont'd)

†Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

## CANADA

### ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z2
Tel: (403) 230-3586
TWX: 03-827-642

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

### BRITISH COLUMBIA

Hamilton/Avnet Electronics
105-2550 Boundry Road
Burmalay V5M 3Z3
Tel: (604) 272-4242

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

### MANITOBA

Zentronics
590 Berry Street
Winnipeg R3H OS1
Tel: (204) 775-8661

### ONTARIO

Arrow Electronics Inc.
24 Martin Ross Avenue
Downsview M3J 2K9
Tel: (416) 661-0220
TELEX: 06-218213

Arrow Electronics Inc.
148 Colonnade Road
Nepean K2E 7J5
Tel: (613) 226-6903

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

†Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 225-8840
TWX: 06-976-78

### QUEBEC

Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TELEX: 05-25596

Arrow Electronics Inc.
909 Charest Blvd.
Quebec 61N 269
Tel: (418) 687-4231
TLX: 05-13388

Hamilton/Avnet Electronics
2795 Rue Halpern
St. Laurent H4S 1P8
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

intel

# DOMESTIC SERVICE OFFICES

**CALIFORNIA**

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

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

Intel Corp.
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8211
TWX: 910-339-9279
910-338-0255

Intel Corp.
2000 E. 4th Street
Suite 110
Santa Ana 92705
Tel: (714) 835-5577
TWX: 910-595-2475

Intel Corp.
4350 Executive Drive
Suite 150
San Diego 92121
Tel: (619) 452-5880

**COLORADO**

Intel Corp.
650 South Cherry
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

**CONNECTICUT**

Intel Corp.
26 Mill Plain Road
Danbury 06811
Tel: (203) 748-3130

**FLORIDA**

Intel Corp.
1500 N.W. 62nd Street
Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

**FLORIDA (Cont'd)**

Intel Corp.
242 N. Westmonte
Suite 105
Altamonte Springs 32714
Tel: (305) 869-5588

**GEORGIA**

Intel Corp.
3280 Pointe Parkway
Suite 200
Norcross 30092
Tel: (404) 441-1171

**ILLINOIS**

Intel Corp.
300 N. Martingale Rd.
Suite 300
Schaumburg 60194
Tel: (312) 310-8034
Dispatch: (312) 310-1803

**KANSAS**

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 642-8080

**MARYLAND**

Intel Corp.
5th Floor Product Service
7833 Walker Drive
Greenbelt 20770
Tel: (301) 441-1020

**MASSACHUSETTS**

Intel Corp.
27 Industrial Avenue
Chelmsford 01824
Tel: (617) 256-1800
TWX: 710-343-6333

**MICHIGAN**

Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48033
Tel: (313) 851-8905

**MISSOURI**

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

**NEW JERSEY**

Intel Corp.
385 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0820
TWX: 710-991-8593

Intel Corp.
Raritan Plaza III
Raritan Center
Edison 08817
Tel: (201) 225-3000

**NORTH CAROLINA**

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

**OHIO**

Intel Corp.
Chagrin-Brainard Bldg.
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

Intel Corp.
6500 Poe
Daytcn 45414
Tel: (513) 890-5350

**OREGON**

Intel Corp.
10700 S.W. Beaverton-Hillsdale
Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

**OREGON (Cont'd)**

Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

**PENNSYLVANIA**

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel: (313) 354-1540

**TEXAS**

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512)454-3628
TWX: 910-874-1347

Intel Corp.
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

**WASHINGTON**

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: 1-800-525-5560
TWX: 910-443-3002

**WISCONSIN**

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Brookfield 53005
Tel: (414) 784-8087

## EUROPEAN SALES OFFICES

**BELGIUM**

Intel Corporation S.A.
Parc Seny
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02)661 07 11
TELEX: 24814

**DENMARK**

Intel Denmark A/S*
Glentevej 61 - 3rd Floor
DK-2400 Copenhagen
Tel: (01) 19 80 33
TELEX: 19567

**FINLAND**

Intel Finland OY
Rousilantie 2
SF-00390 Helsingfors 39
Tel: (90) 544 644
TELEX: 123 332

**FRANCE**

Intel Paris
1, rue Edison, BP 303
78054 Saint-Quentin en Yvelines
Tel: (3) 064 60 00
TELEX: 699016

**FRANCE (Cont'd)**

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842 40 89
TELEX: 305153

**WEST GERMANY**

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Munchen 2
Tel: (89) 53891
TELEX: 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzerstrasse 75
D-6200 Wiesbaden 1
Tel: (6121) 70 08 74
TELEX: 04168183 INTW D

Intel Semiconductor GmbH
Bruckstrasse 61
7012 Fellbach
Stuttgart
Tel: (711) 58 00 82
TELEX: 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollernstrasse 5*
3000 Hannover 1
Tel: (511) 34 40 81
TELEX: 923625 INTH D

**ISRAEL**

Intel Semiconductors Ltd.*
Atidim Industrial Park
Neve Sharet
Dvora Hanevia
Bldg. No. 13, 4th Floor
P.O. Box 43202
Tel Aviv 61430
Tel: 3-491099
Telex: 371215

**ITALY**

Intel Corporation Italia Spa*
Milanofiori, Palazzo E
20094 Assago (Milano)
Tel: (02) 824 00 06
TELEX: 315183 INTMIL

**NETHERLANDS**

Intel Semiconductor Nederland B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21 23 77
TELEX: 22283

**NORWAY**

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

**SPAIN**

Intel Iberia
Calle Zurbaran 28
Madrid 04
Tel: (34) 1410 40 04
TELEX: 46880

**SWEDEN**

Intel Sweden A.B.*
Dalvagen 24
S-171 36 Solna
Tel: (08) 734 01 00
TELEX: 12261

**SWITZERLAND**

Intel Semiconductor A.G.*
Talackerstrasse 17
8152 Glattbrugg postfach
CH-8065 Zurich
Tel: (01) 829 29 77
TELEX: 57989 ICH CH

**UNITED KINGDOM**

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 488 388
TELEX: 444447 INT SWN

*Field Application Location

## EUROPEAN DISTRIBUTORS/REPRESENTATIVES

**AUSTRIA**

Bacher Elektronische Geraete GmbH
Rotenmuehlgasse 26
A 1120 Wien
Tel: (222) 83 56 46
TELEX: 11532 BASAT A

W. Moor GmbH
Storchengasse 1/1/1
A-1150 Wien
Tel: 222-85 86 46

**BELGIUM**

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (021) 216 01 60
TELEX: 25441

**DENMARK**

ITT MultiKomponent A/S
Naverland 29
DK-2600 Gloskrup
Tel: (02) 45 66 45
TX: 33355

**FINLAND**

Oy Fintronic AB
Melkonkalu 24 A
SF-00210 Helsinki 21
Tel: (0) 692 60 22
TELEX: 124 224 Ftron SF

**FRANCE**

Generim
Z.I. de Courtaboeuf
Avenue de la Baltique
F-91943 Les Ulis Cedex-B.P.88
Tel: (1) 907 78 78
TELEX: F691700

Jermyn S.A.
16, Avenue de Jean-Jaures
F-94600 Choisy-Le-Roi
Tel: (1) 853 12 00
TELEX: 260 967

Metrologie
La Tour d' Asnieres
4, Avenue Laurent Cely
F-92606-Asnieres
Tel: (1) 790 62 40
TELEX: 611-448

Tekelec Airtronic
Cte des Bruyeres
Rue Carle Vernet B.P. 2
F-92310 Sevres
Tel: (1) 534 75 35
TELEX: 204552

**WEST GERMANY**

Electronic 2000 Vertriebs A.G.
Stahlgruberring 12
D-8000 Munich 82
Tel: (89) 42 00 10
TELEX 522561 EEC D

Jermyn GmbH
Postfach 11 80
Schulstrasse 84
D-6277 Bad Camberg
Tel: (06434) 231
TELEX: 484426 JERM D

CES Computer Electronics Systems GmbH
Gutenbergstrasse 4
D-2359 Henstedt-Ulzburg
Tel: (04193) 4026
TELEX: 2180260

Metrologie GmbH
Hansastrasse 15
D-8000 Munich 21
Tel: (89) 57 30 84
TELEX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
D-6072 Dreieich
Tel: (6103) 33564
TELEX: 417983

**IRELAND**

Micro Marketing
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (1) 85 62 88
TELEX: 31584

**ISRAEL**

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: (3) 47 51 51
TELEX: 33638

**ITALY**

Electra 3S S.P.A.
Viale Elvezia, 18
I 20154 Milano
Tel: (2) 34 97 51
TELEX: 332332

Intesi
Milanofiori Pal. E/5
I-20090 Assago
Milano
Tel: (2) 82470
TELEX: 311351

**NETHERLANDS**

Koning & Hartman
Koperwerf 30
P.O. Box 43220
2544 EN's Gravenhage
Tel: 31 (70) 210.101
TELEX: 31528

**NORWAY**

Nordisk Elektronic (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: (2) 846 210
TELEX: 17546

**PORTUGAL**

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
P-1000 Lisboa
Tel: (19) 545 313
TELEX: 14182 Brieks-P

**SPAIN**

Interface S.A.
Av. Pompeu Fabra 12
08024 Barcelona
Tel: (3) 219 80 11
TELEX: 51508

ITT SESA
Miguel Angel 21, 6 Piso
Madrid 10
Tel: (34) 14 1954 00
TELEX: 27461

Diode Espana
Avenida De Brasil 5
28020 Madrid
Tel: 455 36 86
TELEX: 42148

**SWEDEN**

AB Gosta Backstrom
Box 12009
Alstroemergatan 22
S-10221 Stockholm
Tel: (08) 541 080
TELEX: 10135

Nordisk Electronik AB
Box 1409
Huvudstagatan 1
171 27 Solna
Tel: (08) 734 97 70
TELEX: 10547

**SWITZERLAND**

Industrade AG
Hertistrasse 31
CH-8304 Wallisellen
Tel: (01) 830 50 40
TELEX: 56788 INDEL CH

**UNITED KINGDOM**

Bytech Ltd.
Unit 57
London Road
Early, Reading
Berkshire
Tel: (0734) 61031
TELEX: 848215

Comway Microsystems Ltd.
Market Street
UK-Bracknell, Berkshire
Tel: 44 (344) 55333
TELEX: 847201

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 450144
TELEX: 95142

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: (190) 49307
TELEX: 28817

Rapid Recall, Ltd.
Rapid House/Denmark St
High Wycombe
Berks, England HP11 2ER
Tel: (494) 26 271
TELEX: 837931

**YUGOSLAVIA**

H. R. Microelectronics Enterprises
P.O. Box 5604
San Jose, California 95150
Tel: 408/978-8000
TELEX: 278-559

## INTERNATIONAL SALES OFFICES

**AUSTRALIA**

Intel Australia Pty. Ltd.*
(Mailing Address)
P.O. Box 571
North Sydney NSW, 2060

(Shipping Address)
Spectrum Building
200 Pacific Highway
Level 6
Crows Nest, NSW, 2065
Tel: 011-61-2-957-2744
TELEX: 790-20097
FAX: 011-61-2-957-2744

**CHINA**

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Avenue
Beijing, PRC

**HONG KONG**

Intel Semiconductor Ltd.*
1701-3 Connaught Centre
1 Connaught Road
Tel: 011-852-5-215-311
TWX: 60410 ITLHK

**JAPAN**

Intel Japan K.K.
5-6 Tokodai, Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511
TELEX: 03656-160

Intel Japan K.K.*
Komeshin Bldg.
2-1-15 Naka-machi
Atsugi, Kanagawa 243
Tel: 0462-23-3511

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871

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

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

**JAPAN (Cont'd)**

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621

Intel Japan K.K.*
Flower-Hill Shin-machi East Bldg.
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel: 03-426-2231

Intel Japan K.K.*
Mitsui-Seimei Musashi-Kosugi Bldg.
915 Shinmaruko, Nakahara-ku
Kawasaki-Shi, Kanagawa 211
Tel: 044-733-7011

Intel Japan K.K.
Mishima Tokyo-Kaijo Bldg.
1-1 Shibahon-cho
Mishima-shi
Shizuoka-Ken 411
Tel: 0559-72-4121

**KOREA**

Intel Semiconductor Asia Ltd.
Singsong Bldg. 8th Floor #906
25-4 Yoido-Dong, Youngdeungpo-Ku
Seoul 150
Tel: 011-82-2-784-8186 or 8286
TELEX: K29312 INTELKO

**SINGAPORE**

Intel Semiconductor Ltd.
101 Thomson Road
21-06 Goldhill Square
Singapore 1130
Tel: 011-65-250-7811
TWX: RS 39921

**TAIWAN**

Intel Semiconductor Ltd.
Rm. 808, Min Chi Bldg.
746 Min Sheng East Road
Taipei

*Field Application Location

## INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

**ARGENTINA**

VLC S.R.L Bartalome Mitre 1711
30 Piso
1037 Buenos Aires
Tel: 011-54-1-49-2092
Telex: 17575 EDARG-AR

Agent:
Soimex International Corporation
15 Park Row, Room #1730
New York, New York 10038
Tel: (212) 406-3052

**AUSTRALIA**

Total Electronics
(Mailing Address)
Private Bag 250
Burwood, Victoria 3125

(Shipping Address)
9 Harker Street
Burwood
Victoria 3125
Tel: 011-61-3-288-4044
TELEX: AA 31261

Total Electronics
P.O. Box 139
Artamon, N.S.W. 2064
Tel: 011-61-02-438-1855
TELEX: 26297

**BRAZIL**

Elebra Microelectronica S/A
R. Bogaert, 326
04298 - Sao Paulo
Tel: 011-55-11-274-9945
TELEX: 1132864

**CHILE**

DIN
(Mailing Address)
Av. VIC, MacKenna 204
Casilla 6055
Santiago
Tel: 011-56-2-277-564
TELEX: 352-0003

(Shipping Address)
A102 Greenville Center
3801 Kennett Pike
Wilmington, Delaware 19807

**CHINA**

Novel Precision Machinery Co., Ltd.
Flat D 20 Kingsford Ind. Bldg.
Phase 1 26 Kwai Hei Street NT
Hong Kong
Tel: 011-852-5-223222
TWX: 39114 JINMI HK

**CHINA (Cont'd)**

Schmidt & Co. Ltd.
18/F. Great Eagle Centre
Wanchai
Hong Kong
Tel: 011-852-5-822-0222
TWX: 74766 SCHMC HK

**HONG KONG**

Schmidt & Co. Ltd.
18/F. Great Eagle Centre
Wanchai
Tel: 011-852-5-822-0222
TWX: 74766 SCHMC HK

**INDIA**

Micronic Devices
65 Arun Complex
D V G Road
Basavan Gudi
Bangalore 560 004
Tel: 011-91-812-600-631
TELEX: 011-5947 MDEV

Micronic Devices
104/109C Nirmal Industrial Estate
Sion (E)
Bombay 400 022
Tel: 011-91-22-48-61-70
TELEX: 011-71447 MDEV IN

Micronic Devices
R-694 New Rajinder Nager
New Delhi 110 060

Ramlak International, Inc. (Agent)
465 S. Mathilda Avenue
Suite 302
Sunnyvale, CA 94086
Tel: (408) 733-8767

S & S Corporation
P.O. Box 20160
San Jose 95160-0160

**JAPAN**

Asahi Electronics Co. Ltd.
KMM Bldg. Room 407
2-14-1 Asano, Kokurakita-Ku
Kitakyushu City 802
Tel: (093) 511-6471
TELEX: AECKY 7126-16

C. Itoh Micronics Corp.
OS 85 Bldg. 2-6-5 Suda-Cho
Kanda Chiyoda-Ku, Tokyo 101
Tel: (03) 256-2211
TELEX: (03) 252-3774

**JAPAN (Cont'd)**

Ryoyo Electric Corporation
Shuwa Sakurabashi Bldg.
4-5-4 Hatchobori
Chuo-Ku, Tokyo 104
Tel: (03) 555-4811

Tokyo Electron Ltd.
Shinjuku Nomura Bldg.
1-26-2 Nishi-Shinjuku
Shinjuku-Ku, Tokyo 160
Tel. (03) 343-4411
TELEX: 232-2220 LABTEL J

**KOREA**

J-TEK Corporation
2nd Floor, Government Pension Bldg.
24-3 Yoido-Dong
Youngdungpo-Ku
Seoul 150
Tel: 011-82-2-782-8039
TELEX: KODIGIT K25299

Koram Digital USA (Agent)
14066 East Firestone Boulevard
Sante Fe Springs, CA 90670
Tel: (714) 739-2204
TELEX: 194715 KORAM DIGIT LSA

Samsung
23rd Fl. Dong Bang Bldg.
1502-KA Taepyung-RU
Chung-Ku
Seoul
Tel: 777-78
TELEX 27970 KORSST K

Tristar Semiconductor (Agent)
5150 Great America Parkway
Santa Clara, CA 95050
(408) 980-1630

**MEXICO**

DICOPEL S.A.
Tochtli 368 Fracc. Ind. Sn. Antonio
Azcapotzalco
02760-Mexico, D.F.
Tel: 90115255613211
TELEX: 1773790 DICOME

**NEW ZEALAND**

Northrup Instruments & Systems Ltd.
459 Kyber Pass Road
P.O. Box 9464, Newmarket
Auckland 1
Tel: 011-64-9-501-219, 501-801, 587-037
TELEX: NZ21570 THERMAL

Northrup Instruments & Systems Ltd.
P.O. Box 2406
Wellington 856658
TELEX: NZ3380

**PAKISTAN**

Computer Applications Ltd.
7D Gizri Boulevard
Defence
Karachi-46
Tel: 011-92-21-530-306
TELEX: 24434 GAFAR PK

Horizon Training Co., Inc. (Agent)
1 Lafayette Center
1120 20th Street N.W.
Suite 530
Washington, D.C. 20036
Tel: (202) 887-1900
TWX: 248890 HORN

**SINGAPORE**

General Engineers Corporation Pty. Ltd.
203 Henderson Road
1102 Henderson Industrial Park 0315
Tel: 011065-271-3163
TELEX: RS23987 GENERCO

**SOUTH AFRICA**

Electronic Building Elements, Pty. Ltd.
(Mailing Address)
P.O. Box 4609
Pretoria 0001
Tel: 011-27-12-469921
TELEX: 3-22786 SA

(Shipping Address)
Pine Square, 18th Street
Hazelwood Pretoria

**TAIWAN**

Mitac Corporation
No. 585 Ming Sheng E. Road
Taipei
Tel: 011-96-2-501-8231
TELEX: 11942 TAIAUTO

Mectel International, Inc. (Agent)
3385 Viso Court
Santa Clara, CA 95050
Tel: (408) 988-4513
TWX: 910-338-2201
FAX: 408-980-9742

**YUGOSLAVIA**

H. R. Microelectronics Enterprises
P.O. Box 5604
San Jose, California 95150
Tel: (408) 978-8000
TELEX: 278-559

*Field Application Location

INTEL CORPORATION, 3065 Bowers Ave., Santa Clara, CA 95051; Tel. (408) 987-8080.

INTEL CORPORATON (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 488 388.

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

Printed in U.S.A./CR-034/1085/15K/CP RM
Microprocessors