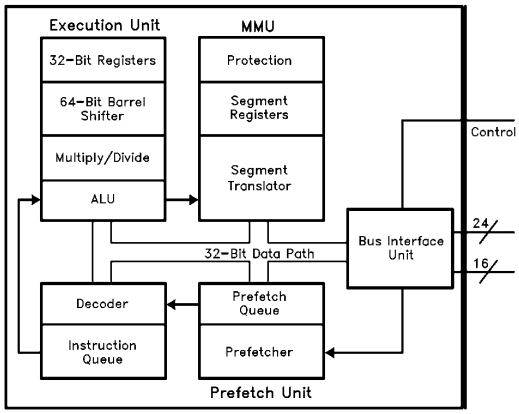
PRELIMINARY

376™ HIGH PERFORMANCE 32-BIT EMBEDDED PROCESSOR

- **Full 32-Bit Internal Architecture**
  - **— 8-, 16-, 32-Bit Data Types**
  - **— 8 General Purpose 32-Bit Registers**
  - **— Extensive 32-Bit Instruction Set**
- **High Performance 16-Bit Data Bus**
  - **— 16 or 20 MHz CPU Clock**
  - **— Two-Clock Bus Cycles**
  - **— 16 Mbytes/Sec Bus Bandwidth**
- **16 Mbyte Physical Memory Size**
- **High Speed Numerics Support with the 80387SX**
- **Low System Cost with the 82370 Integrated System Peripheral**
- **On-Chip Debugging Support Including Break Point Registers**
- **Complete Intel Development Support**
  - **— C, PL/M, Assembler**
  - **— ICE™-376, In-Circuit Emulator**
  - **— iRMK Real Time Kernel**
  - **— iSDM Debug Monitor**
  - **— DOS Based Debug**
- **Extensive Third-Party Support:**
  - **— Languages: C, Pascal, FORTRAN, BASIC and ADA***
  - **— Hosts: VMS*, UNIX*, MS-DOS*, and Others**
  - **— Real-Time Kernels**
- **High Speed CHMOS IV Technology**
- **Available in 100 Pin Plastic Quad Flat-Pack Package and 88-Pin Pin Grid Array**
  (See Packaging Outlines and Dimensions #231369)

INTRODUCTION

The 376 32-bit embedded processor is designed for high performance embedded systems. It provides the performance benefits of a highly pipelined 32-bit internal architecture with the low system cost associated with 16-bit hardware systems. The 80376 processor is based on the 80386 and offers a high degree of compatibility with the 80386. All 80386 32-bit programs not dependent on paging can be executed on the 80376 and all 80376 programs can be executed on the 80386. All 32-bit 80386 language translators can be used for software development. With proper support software, any 80386-based computer can be used to develop and test 80376 programs. In addition, any 80386-based PC-AT* compatible computer can be used for hardware prototyping for designs based on the 80376 and its companion product the 82370.



80376 Microarchitecture

Intel, iRMK, ICE, 376, 386, Intel386, iSDM, Intel1376 are trademarks of Intel Corp.
*UNIX is a registered trademark of AT&T.
ADA is a registered trademark of the U.S. Government, Ada Joint Program Office.
PC-AT is a registered trademark of IBM Corporation.
VMS is a trademark of Digital Equipment Corporation.
MS-DOS is a trademark of MicroSoft Corporation.

December 1990 **Order Number: 240182-004**

1.0 PIN DESCRIPTION

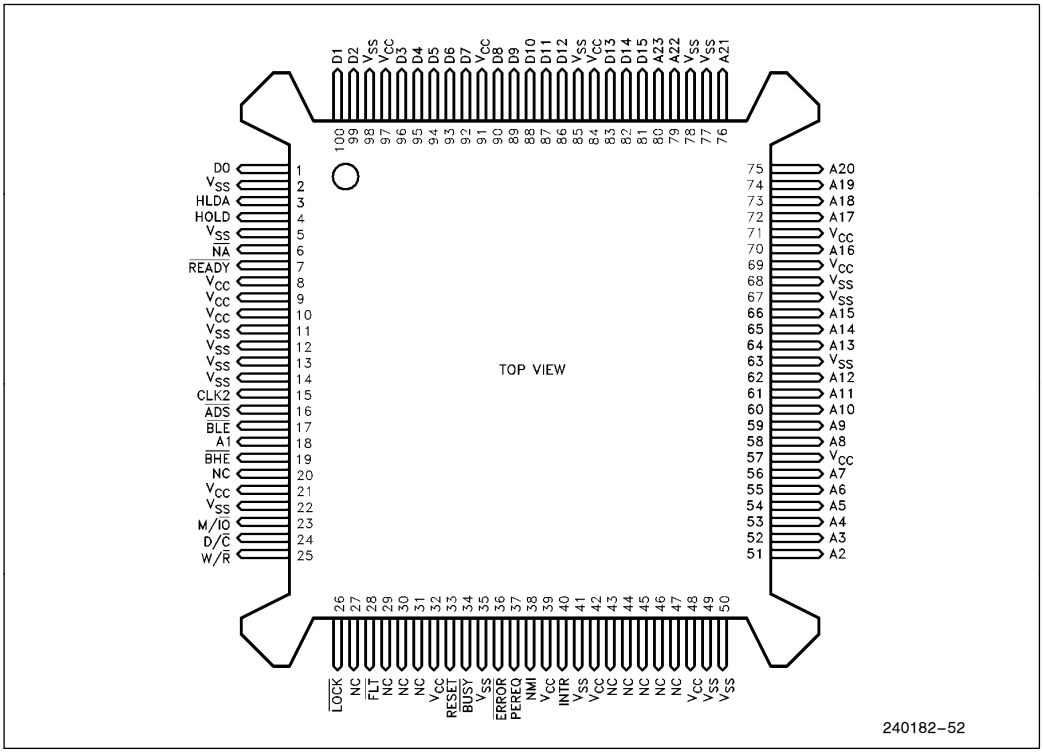
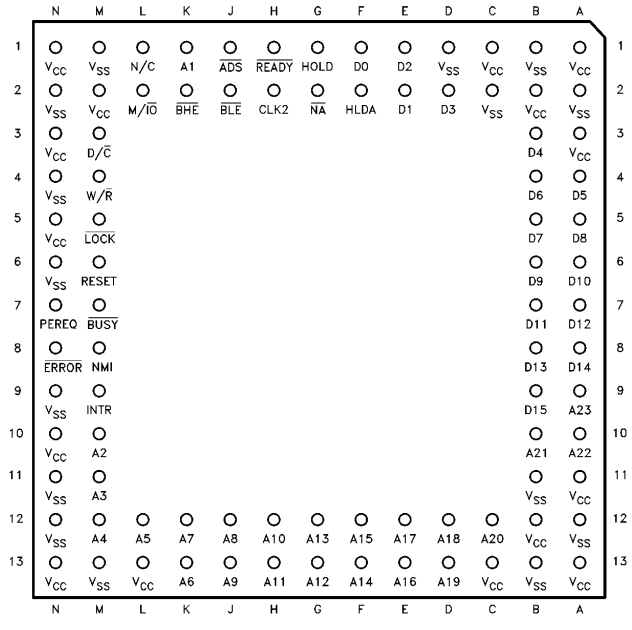


Figure 1.1. 80376 100-Pin Quad Flat-Pack Pin Out (Top View)

Table 1.1. 100-Pin Plastic Quad Flat-Pack Pin Assignments

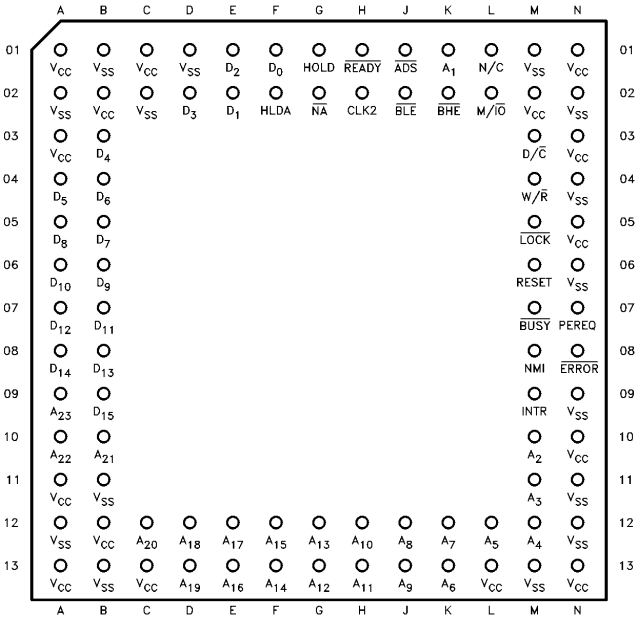
Address		Data		Control		N/C	V_{CC}	V_{SS}
A_1	18	D_0	1	$\overline{ADS}$	16	20	8	2
A_2	51	D_1	100	$\overline{BHE}$	19	27	9	5
A_3	52	D_2	99	$\overline{BLE}$	17		10	11
A_4	53	D_3	96	$\overline{BUSY}$	34	29	21	12
A_5	54	D_4	95	CLK2	15	30	32	13
A_6	55	D_5	94	D/$\overline{C}$	24	31	39	14
A_7	56	D_6	93	$\overline{ERROR}$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

Top View
(Component Side)



240182-49

Bottom View
(Pin Side)



240182-2

Figure 1.2. 80376 88-Pin Grid Array Pin Out

Table 1.2. 88-Pin Grid Array Pin Assignments

Pin	Label	Pin	Label	Pin	Label	Pin	Label
2H	CLK2	12D	A_{18}	2L	M/$\overline{IO}$	11A	V_{CC}
9B	D_{15}	12E	A_{17}	5M	$\overline{LOCK}$	13A	V_{CC}
8A	D_{14}	13E	A_{16}	1J	$\overline{ADS}$	13C	V_{CC}
8B	D_{13}	12F	A_{15}	1H	$\overline{READY}$	13L	V_{CC}
7A	D_{12}	13F	A_{14}	2G	$\overline{NA}$	1N	V_{CC}
7B	D_{11}	12G	A_{13}	1G	HOLD	13N	V_{CC}
6A	D_{10}	13G	A_{12}	2F	HLDA	11B	V_{SS}
6B	D_9	13H	A_{11}	7N	PEREQ	2C	V_{SS}
5A	D_8	12H	A_{10}	7M	$\overline{BUSY}$	1D	V_{SS}
5B	D_7	13J	A_9	8N	$\overline{ERROR}$	1M	V_{SS}
4B	D_6	12J	A_8	9M	INTR	4N	V_{SS}
4A	D_5	12K	A_7	8M	NMI	9N	V_{SS}
3B	D_4	13K	A_6	6M	RESET	11N	V_{SS}
2D	D_3	12L	A_5	2B	V_{CC}	2A	V_{SS}
1E	D_2	12M	A_4	12B	V_{CC}	12A	V_{SS}
2E	D_1	11M	A_3	1C	V_{CC}	1B	V_{SS}
1F	D_0	10M	A_2	2M	V_{CC}	13B	V_{SS}
9A	A_{23}	1K	A_1	3N	V_{CC}	13M	V_{SS}
10A	A_{22}	2J	$\overline{BLE}$	5N	V_{CC}	2N	V_{SS}
10B	A_{21}	2K	$\overline{BHE}$	10N	V_{CC}	6N	V_{SS}
12C	A_{20}	4M	W/$\overline{R}$	1A	V_{CC}	12N	V_{SS}
13D	A_{19}	3M	D/$\overline{C}$	3A	V_{CC}	1L	N/C

The following table lists a brief description of each pin on the 80376. The following definitions are used in these descriptions:

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

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the 80376. For additional information see **Clock** in Section 4.1.
RESET	I	**RESET** suspends any operation in progress and places the 80376 in a known reset state. See **Interrupt Signals** in Section 4.1 for additional information.
$D_{15}-D_0$	I/O	**DATA BUS** inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See **Data Bus** in Section 4.1 for additional information.
$A_{23}-A_1$	O	**ADDRESS BUS** outputs physical memory or port I/O addresses. See **Address Bus** in Section 4.1 for additional information.
W/$\overline{R}$	O	**WRITE/READ** is a bus cycle definition pin that distinguishes write cycles from read cycles. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
D/$\overline{C}$	O	**DATA/CONTROL** is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and instruction fetching. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
M/$\overline{IO}$	O	**MEMORY I/O** is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
$\overline{LOCK}$	O	**BUS LOCK** is a bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
$\overline{ADS}$	O	**ADDRESS STATUS** indicates that a valid bus cycle definition and address (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{BHE}$, $\overline{BLE}$ and $A_{23}-A_1$) are being driven at the 80376 pins. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{NA}$	I	**NEXT ADDRESS** is used to request address pipelining. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{READY}$	I	**BUS READY** terminates the bus cycle. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{BHE}$, $\overline{BLE}$	O	**BYTE ENABLES** indicate which data bytes of the data bus take part in a bus cycle. See **Address Bus** in Section 4.1 for additional information.
HOLD	I	**BUS HOLD REQUEST** input allows another bus master to request control of the local bus. See **Bus Arbitration Signals** in Section 4.1 for additional information.

Symbol	Type	Name and Function
HLDA	O	**BUS HOLD ACKNOWLEDGE** output indicates that the 80376 has surrendered control of its local bus to another bus master. See **Bus Arbitration Signals** in Section 4.1 for additional information.
INTR	I	**INTERRUPT REQUEST** is a maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** in Section 4.1 for additional information.
NMI	I	**NON-MASKABLE INTERRUPT REQUEST** is a non-maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** in Section 4.1 for additional information.
$\overline{\text{BUSY}}$	I	**BUSY** signals a busy condition from a processor extension. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
$\overline{\text{ERROR}}$	I	**ERROR** signals an error condition from a processor extension. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
PEREQ	I	**PROCESSOR EXTENSION REQUEST** indicates that the processor extension has data to be transferred by the 80376. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
$\overline{\text{FLT}}$	I	**FLOAT,** when active, forces all bidirectional and output signals, including HLDA, to the float condition. FLOAT is not available on the PGA package. See **Float** for additional information.
N/C	—	**NO CONNECT** should always remain unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 80376.
V_{CC}	I	**SYSTEM POWER** provides the +5V nominal D.C. supply input.
V_{SS}	I	**SYSTEM GROUND** provides 0V connection from which all inputs and outputs are measured.

2.0 ARCHITECTURE OVERVIEW

The 80376 supports the protection mechanisms needed by sophisticated multitasking embedded systems and real-time operating systems. The use of these protection mechanisms is completely optional. For embedded applications not needing protection, the 80376 can easily be configured to provide a 16 Mbyte physical address space.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 80376 is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The 80376 offers on-chip testability and debugging features. Four break point registers allow conditional or unconditional break point traps on code execution or data accesses for powerful debugging of even ROM based systems. Other testability features include self-test and tri-stating of output buffers during RESET.

The Intel 80376 embedded processor 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 registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The Memory Management Unit (MMU) consists of a segmentation and protection 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 protection 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 and simplifies debugging.

Finally, to facilitate high performance system hardware designs, the 80376 bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 80376 has twenty-nine registers as shown in Figure 2.1. These registers are grouped into the following six categories:

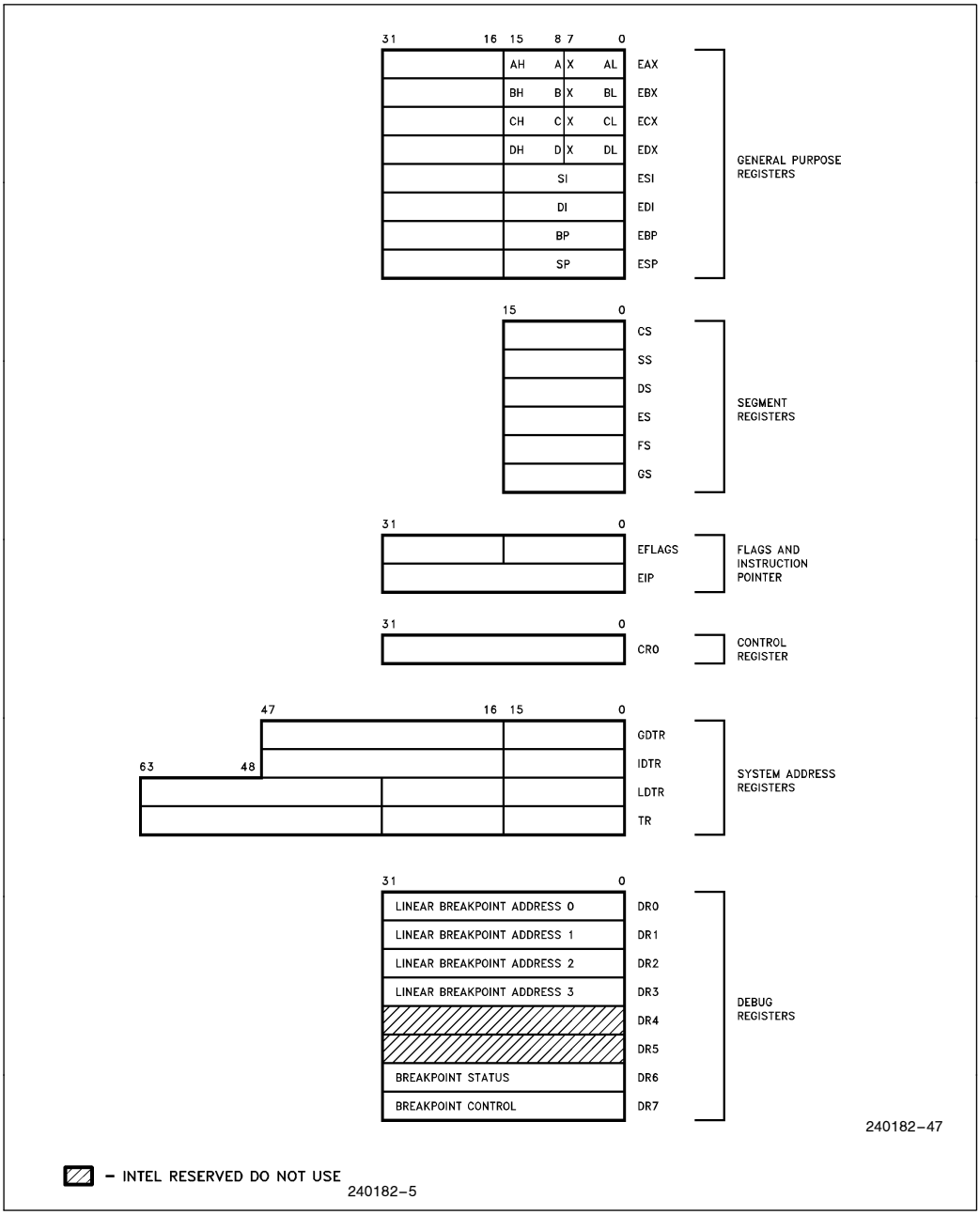


Figure 2.1. 80376 Base Architecture Registers

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

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

Flags and Instruction Pointer Registers: These two 32-bit special purpose registers in Figure 2.1 record or control certain aspects of the 80376 processor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Register: The 32-bit control register, CR0, is used to control Coprocessor Emulation.

System Address Registers: These four special registers reference the tables or segments supported by the 80376/80386 protection model. These tables or segments are:

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

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.11 **Debugging Support**.

EFLAGS REGISTER

The flag Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 80376 processor. The function of the flag bits is given in Table 2.1.

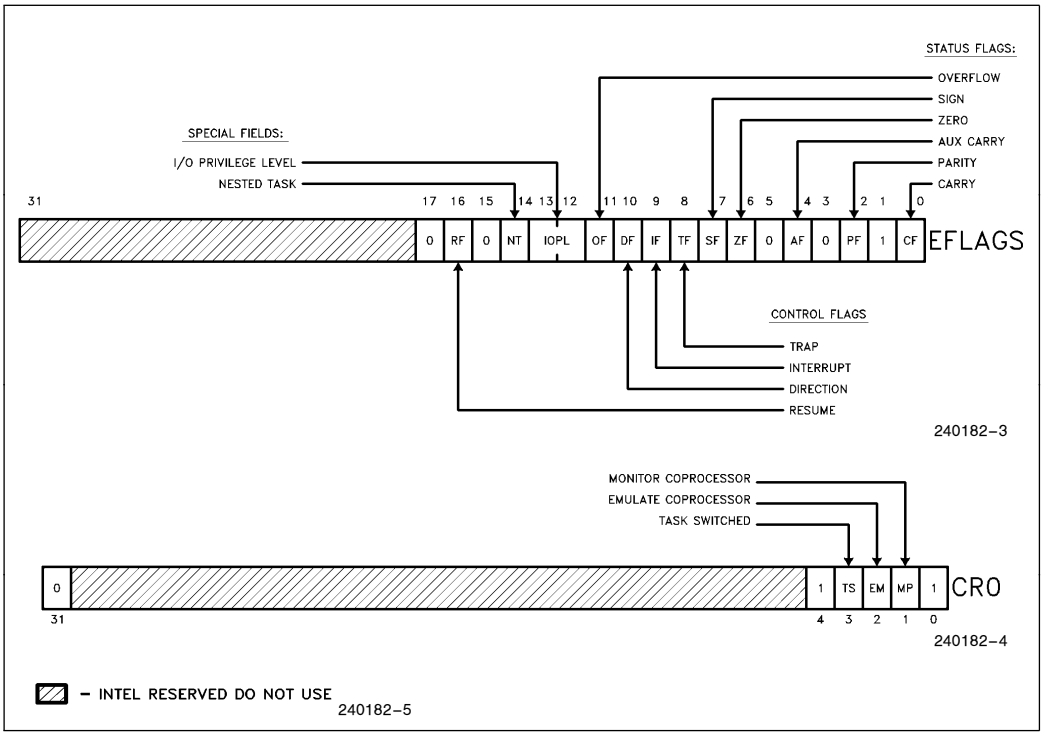


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	**Carry Flag**—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	**Parity Flag**—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	**Auxiliary Carry Flag**—Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	**Zero Flag**—Set if result is zero; cleared otherwise.
7	SF	**Sign Flag**—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	**Single Step Flag**—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	**Interrupt-Enable Flag**—When set, external interrupts signaled on the INTR pin will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	**Direction Flag**—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	**Overflow Flag**—Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12, 13	IOPL	**I/O Privilege Level**—Indicates the maximum CPL permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. It also indicates the maximum CPL value allowing alteration of the IF bit.
14	NT	**Nested Task**—Indicates that the execution of the current task is nested within another task (see **Task Switching**).
16	RF	**Resume Flag**—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction. It is reset at the successful completion of any instruction except IRET, POPF, and those instructions causing task switches.

CONTROL REGISTER

The 80376 has a 32-bit control register called CR0 that is used to control coprocessor emulation. This register is shown in Figures, 2.1 and 2.2. The defined CR0 bits are described in Table 2.2. Bits 0, 4 and 31 of CR0 have fixed values in the 80376. These values cannot be changed. Programs that load CR0 should always load bits 0, 4 and 31 with values previously there to be compatible with the 80386.

Table 2.2. CR0 Definitions

Bit Position	Name	Function
1	MP	**Monitor Coprocessor Extension**—Allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	**Emulate Processor Extension**—When set, this bit causes a processor extension not present exception (number 7) on ESC instructions to allow processor extension emulation.
3	TS	**Task Switched**—When set, this bit indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task (see **Task Switching**).

2.2 Instruction Set

The instruction set is divided into nine categories of operations:

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

These 80376 processor instructions are listed in Table 8.1 **80376 Instruction Set and Clock Count Summary**.

All 80376 processor instructions operate on either 0, 1, 2 or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 80376 has a 16-byte prefetch instruction queue an average of 5 instructions can 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 are either 8-, 16- or 32-bit long.

2.3 Memory Organization

Memory on the 80376 is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a word or Dword is the byte address of the low-order byte. For maximum performance word and dword values should be at even physical addresses.

In addition to these basic data types the 80376 processor supports segments. Memory can be divided up into one or more variable length segments, which can be shared between programs.

ADDRESS SPACES

The 80376 has three types of address spaces: **logical, linear,** and **physical.** A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, and DISPLACEMENT), discussed in Section 2.4 **Addressing Modes,** into an effective address.

Every selector has a **logical base** address associated with it that can be up to 32 bits in length. This 32-bit **logical base** address is added to either a 32-bit offset address or a 16-bit offset address (by using the ***address length prefix***)to form a final 32-bit **linear** address. This final **linear** address is then truncated so that only the lower 24 bits of this address are used to address the 16 Mbytes physical memory address space. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

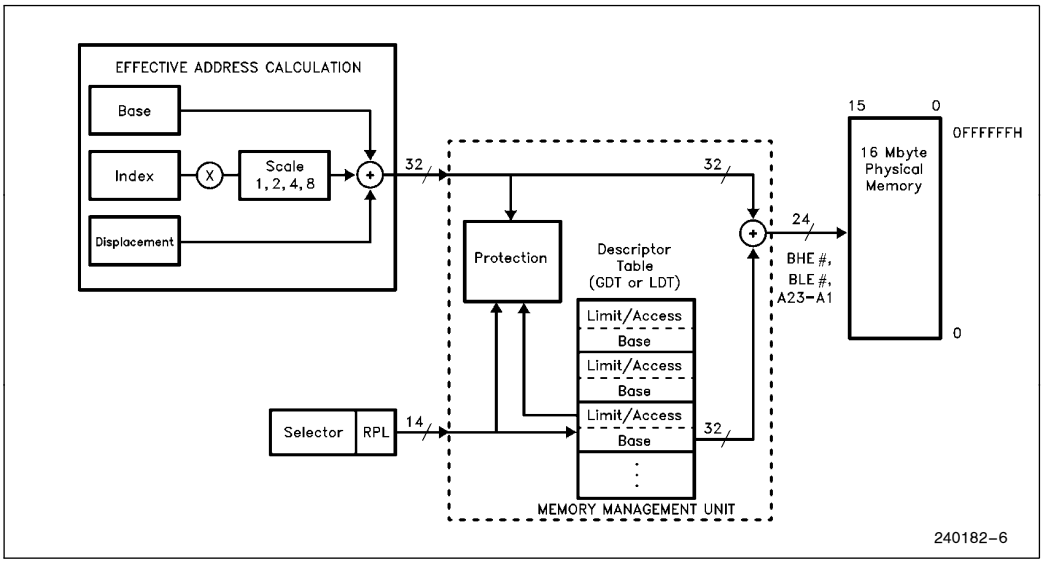


Figure 2.3. Address Translation

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 80376, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The simplest use of segments is to have one code and data segment. Each segment is 16 Mbytes in size overlapping each other. This allows code and data to be directly addressed by the same offset.

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

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero. Further details of segmentation are discussed in Section 3.0 Architecture.

Table 2.3. Segment Register Selection Rules

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

2.4 Addressing Modes

The 80376 provides a total of 8 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

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

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

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

The remaining 6 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see Figure 2.3):

DISPLACEMENT: an 8-, 16- or 32-bit immediate value following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area. Note that if the ***Address Length Prefix*** is used, only BX and BP can be used as a BASE register.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures. Note that if the ***Address Length Prefix*** is used, no Scaling is available and only the registers SI and DI can be used to INDEX.

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

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

$$EA = BASE_{Register} + (INDEX_{Register} \times scaling) + DISPLACEMENT$$

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

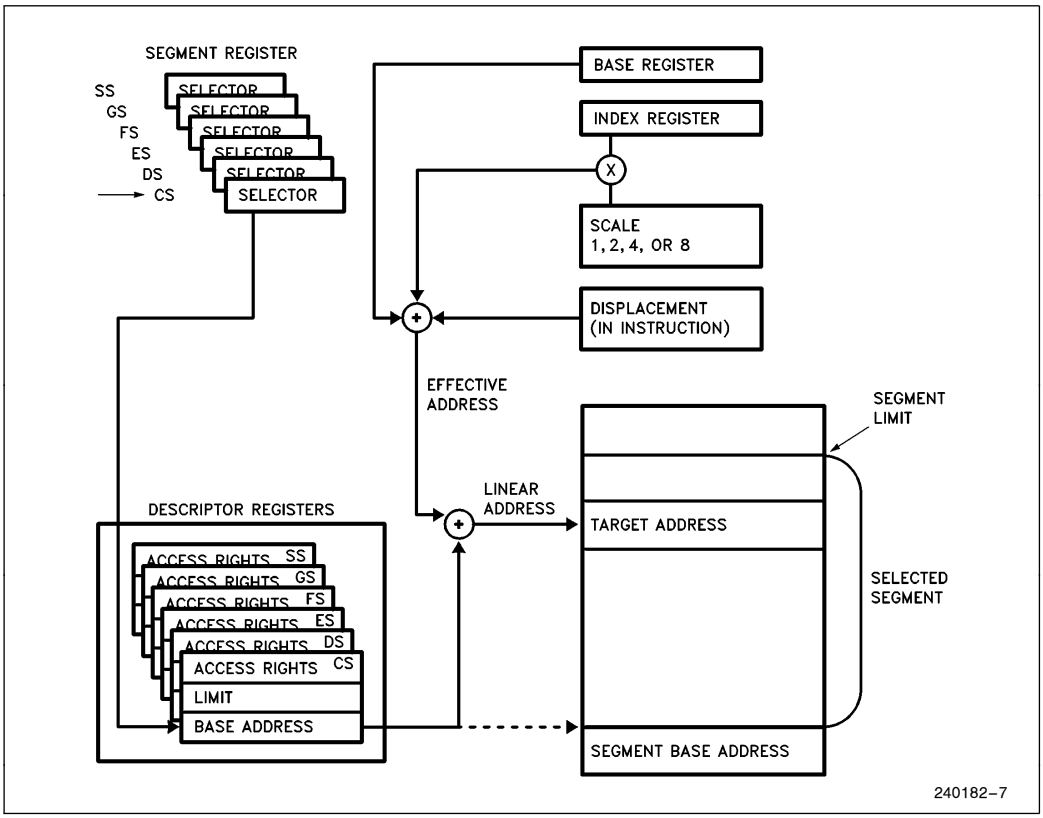


Figure 2.4. Addressing Mode Calculations

GENERATING 16-BIT ADDRESSES

The 80376 executes code with a default length for operands and addresses of 32 bits. The 80376 is also able to execute operands and addresses of 16 bits. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix,** override the default 32-bit length on an individual instruction basis. These prefixes are automatically added by assemblers. The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction.

The 80376 normally executes 32-bit code and uses either 8- or 32-bit displacements, and any register can be used as based or index registers. When executing 16-bit code (by prefix overrides), the displacements are either 8 or 16 bits, and the base and index register conform to the 16-bit model. Table 2.4 illustrates the differences.

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

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX, BP	Any 32-Bit GP Register
INDEX REGISTER	SI, DI	Any 32-Bit GP Register except ESP
SCALE FACTOR	None	1, 2, 4, 8
DISPLACMENT	0, 8, 16 Bits	0, 8, 32 Bits

2.5 Data Types

The 80376 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 80376 bit strings can be up to 16 Mbits long.
Byte:	A signed 8-bit quantity.
Unsigned Byte:	An unsigned 8-bit quantity.
Integer (Word):	A signed 16-bit quantity.
Long Integer (Double Word):	A signed 32-bit quantity. All operations assume a 2's complement representation.
Unsigned Integer (Word):	An unsigned 16-bit quantity.
Unsigned Long Integer (Double Word):	An unsigned 32-bit quantity.
Signed Quad Word:	A signed 64-bit quantity.
Unsigned Quad Word:	An unsigned 64-bit quantity.
Pointer:	A 16- or 32-bit offset only quantity which indirectly references another memory location.
Long Pointer:	A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.
Char:	A byte representation of an ASCII Alphanumeric or control character.
String:	A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 16 Mbytes.
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 80376 is coupled with a numerics Coprocessor such as the 80387SX then the following common Floating Point types are supported.

Floating Point: A signed 32-, 64- or 80-bit real number representation. Floating point numbers are supported by the 80387SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 80376 processor and the 80387SX coprocessor.

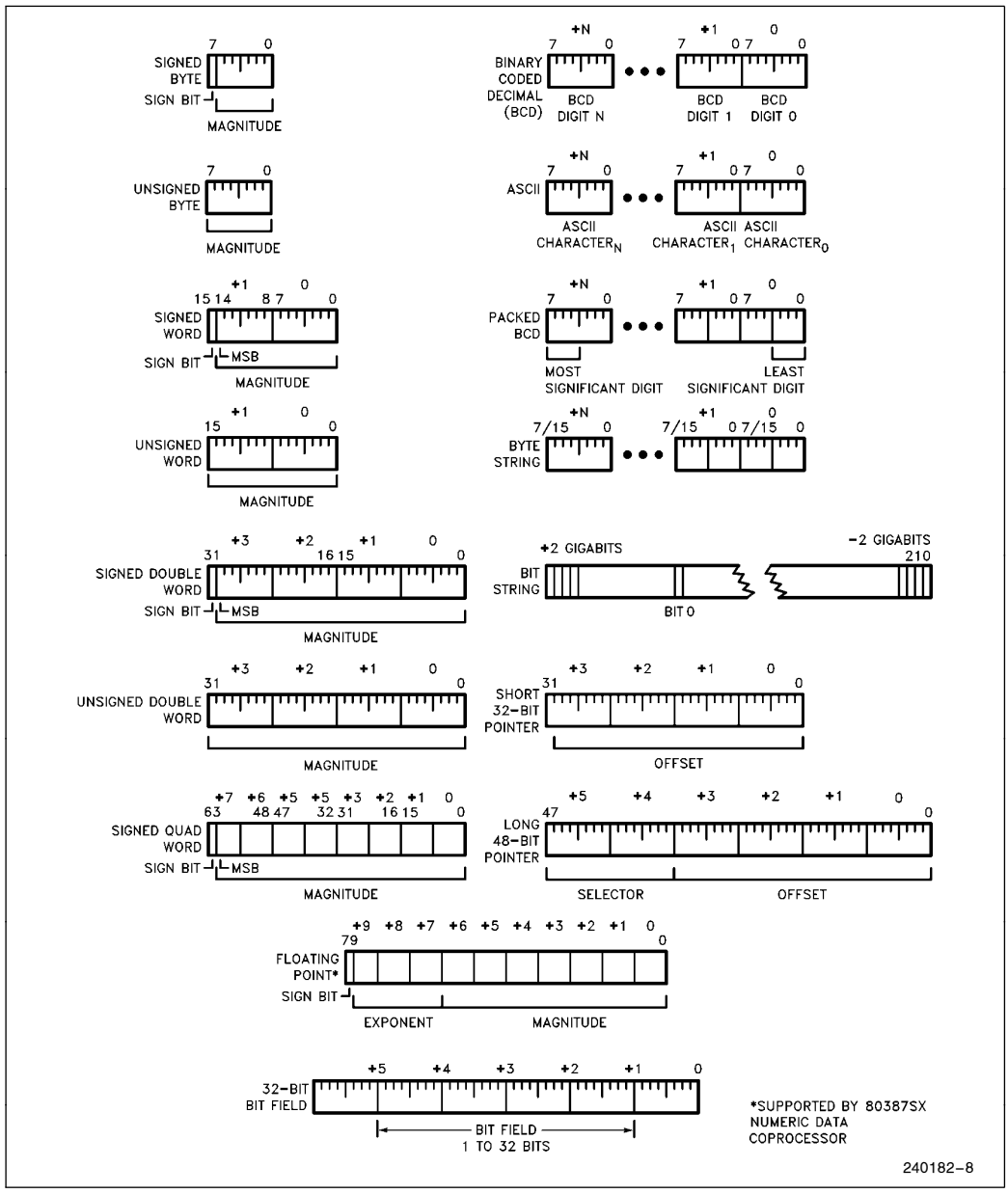


Figure 2.5. 80376 Supported Data Types

2.6 I/O Space

The 80376 has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 80376 also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes which can be divided into 64K 8-bit ports, 32K 16-bit ports, or any combination of ports which add to no more than 64 Kbytes. The M/$\overline{IO}$ pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing. Note that the I/O address refers to a physical address.

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

2.7 Interrupts and Exceptions

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

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

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is suported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand the return address from an exception/fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 80376 and shows where the return address points to.

The 80376 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. The interrupt vectors are 8-byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 80376 which identifies the appropriate entry in the interrupt table. The table contains either an Interrupt Gate, a Trap Gate or a Task Gate that will point to an interrupt procedure or task. 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 80376 in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

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

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	Yes	FAULT
Debug Exception	1	Any Instruction	Yes	TRAP*
NMI Interrupt	2	INT 2 or NMI	No	NMI
One-Byte Interrupt	3	INT	No	TRAP
Interrupt on Overflow	4	INTO	No	TRAP
Array Bounds Check	5	BOUND	Yes	FAULT
Invalid OP-Code	6	Any Illegal Instruction	Yes	FAULT
Device Not Available	7	ESC, WAIT	Yes	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	No	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	Yes	FAULT
Segment Not Present	11	Segment Register Instructions	Yes	FAULT
Stack Fault	12	Stack References	Yes	FAULT
General Protection Fault	13	Any Memory Reference	Yes	FAULT
Intel Reserved	14–15	—	—	—
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.

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

Non-Maskable Interrupt

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

While executing the NMI servicing procedure, the 80376 will not service any further NMI request, or INT requests, until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The disabling of INTR requests depends on the gate in IDT location 2.

Software Interrupts

A third type of interrupt/exception for the 80376 is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the n^{th} vector in the interrupt table.

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

A final type of software interrupt, is the single step interrupt. It is discussed in **Single-Step Trap** (page 22).

INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 80376 invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 80376 will invoke the appropriate interrupt service routine.

As the 80376 executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 80376 fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 9 in Table 2.6), the 80376 device invokes the appropriate exception service routine. The 80376 is in a state that permits restart of the instruction.

DOUBLE FAULT

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

2.8 Reset and Initialization

When the processor is Reset the registers have the values shown in Table 2.7. The 80376 will then start executing instructions near the top of physical memory, at location 0FFFFF0H. A short JMP should be executed within the segment defined for power-up (see Table 2.7). The GDT should then be initialized for a start-up data and code segment followed by a far JMP that will load the segment descriptor cache with the new descriptor values. The IDT table, after reset, is located at physical address 0H, with a limit of 256 entries.

RESET forces the 80376 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 80376 will start executing instructions at the top of physical memory.

Table 2.6. Sequence of Exception Checking

Consider the case of the 80376 having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Faults decoding the next instruction (exception 6 if illegal opcode; or exception 13 if instruction is longer than 15 bytes, or privilege violation (i.e. not at IOPL or at CPL = 0).
6. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
7. If ESCape opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
8. If WAIT opcode or ESCape opcode for numeric coprocessor, check $\overline{\text{ERROR}}$ input signal (exception 16 if $\overline{\text{ERROR}}$ input is asserted).
9. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).

Table 2.7. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	(Note 1)
Machine Status Word (CR0)	uuuuuuu1H	(Note 2)
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	(Note 3)
Data Segment (DS)	0000H	(Note 4)
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	(Note 4)
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX Register	0000H	(Note 5)
EDX Register	Component and Stepping ID	(Note 6)
All Other Registers	Undefined	(Note 7)

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

2.9 Initialization

Because the 80376 processor starts executing in protected mode, certain precautions need be taken during initialization. Before any far jumps can take place the GDT and/or LDT tables need to be setup and their respective registers loaded. Before interrupts can be initialized the IDT table must be setup and the IDTR must be loaded. The example code is shown below:

```
; ************************************************************
;
; This is an example of startup code to put either an 80376,
; 80386SX or 80386 into flat mode. All of memory is treated as
; simple linear RAM. There are no interrupt routines. The
; Builder creates the GDT-alias and IDT-alias and places them,
; by default, in GDT[1] and GDT[2].  Other entries in the GDT
; are specified in the Build file.  After initialization it jumps
; to a C startup routine. To use this template, change this jmp
; address to that of your code, or make the label of your code
; "c_startup".
;
; This code was assembled and built using version 1.2 of the
; Intel RLL utilities and Intel 386ASM assembler.
;
;             ***    This code was tested    ***
;
; ************************************************************
```

```
NAME FLAT               ; name of the object module

EXTRN    c_startup:near ; this is the label jmped to after init

pe_flag       equ 1
data_selc     equ 20h   ; assume code is GDT[3], data GDT[4]

INIT_CODE    SEGMENT ER PUBLIC USE32    ; Segment base at 0ffffff80h

PUBLIC GDT_DESC

gdt_desc    dq  ?

PUBLIC    START

start:
    cld                       ; clear direction flag
    smsw bx                   ; check for processor (80376) at reset
    test bl,l                 ; use SMSW rather than MOV for speed
    jnz pestart
realstart                     ; is an 80386 and in real mode
    db 66h                    ; force the next operand into 32-bit mode.
    mov eax,offset gdt_desc   ; move address of the GDT descriptor into eax
    xor ebx,ebx               ; clear ebx
    mov bh,ah                 ; load 8 bits of address into bh
    move bl,al                ; load 8 bits of address into bl
    db 67h
    db 66h                    ; use the 32-bit form of LGDT to load
    lgdt cs:[ebx]             ; the 32-bits of address into the GDTR
    smsw ax                   ; go into protected mode (set PE bit)
    or al,pe_flag
    lmsw ax
    jmp next                  ; flush prefetch queue
pestart:
    mov ebx,offset gdt_desc
    xor eax,eax
    mov ax,bx                 ; lower portion of address only
    lgdt cs:[eax]
    xor ebx,ebx               ; initialize data selectors
    mov bl,data_selc          ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    jmp pejump
next:
    xor ebx,ebx               ; initialize data selectors
    mov bl,data_selc          ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    db 66h                    ; for the 80386, need to make a 32-bit jump
pejump:
    jmp far ptr c_startup     ; but the 80376 is already 32-bit.

    org 70h                   ; only if segment base is at 0ffffff80h
    jmp short start
INIT_CODE ENDS
END
```

This code should be linked into your application for boot loadable code. The following build file illustrates how this is accomplished.

```
FLAT; -- build program id

SEGMENT
    *segments (dpl=0),            -- Give all user segments a DPL of 0.
    _phantom_code_ (dpl=0),       -- These two segments are created by
    _phantom_data_ (dpl=0),       -- the builder when the FLAT control is used.
    init_code  (base=0ffffff80h); -- Put startup code at the reset vector area.

GATE
    gl3 (entry=l3, dpl=0, trap),       -- trap gate disables interrupts
    i32 (entry=32, dpl=0, interrupt),  -- interrupt gates doesn't

TABLE
    -- create GDT

    GDT (LOCATION = GDT_DESC,     -- In a buffer starting at GDT_DESC,
                                  -- BLD386 places the GDT base and
                                  -- GDT limit values. Buffer must be
                                  -- 6 bytes long. The base and limit
                                  -- values are places in this buffer
                                  -- as two bytes of limit plus
                                  -- four bytes of base in the format
                                  -- required for use by the LGDT
                                  -- instruction.
        ENTRY = (3:_phantom_code_,     -- Explicitly place segment
                 4:_phantom_data_,     -- entries into the GDT.
                 5:code32,
                 6:data,
                 7:init_code)
        );
TASK

    MAIN_TASK
        (
         DPL  = 0,                -- Task privilege level is 0.
         DATA = DATA,             -- Points to a segment that
                                  -- indicates initial DS value.
         CODE = main,             -- Entry point is main, which
                                  -- must be a public id.

         STACKS = (DATA),         -- Segment id points to stack
                                  -- segment. Sets the initial SS:ESP.
         NO INTENABLED,           -- Disable interrupts.
         PRESENT                  -- Present bit in TSS set to 1.
        );

    MEMORY
        (RANGE = (EPROM = ROM(0ffff8000h..0ffffffffh),
                  DRAM = RAM(0..0ffffh)),
        ALLOCATE = (EPROM = (MAIN_TASK)));

END
```

```
asm386 flatsim.a38 debug
asm386 application.a38 debug
bnd386 application.obj,flatsim.obj nolo debug oj (application.bnd)
bld386 application.bnd bf (flatsim.bld) bl flat
```

Commands to assemble and build a boot-loadable application named ‘‘application.a38’’. The initialization code is called ‘‘flatsim.a38’’, and build file is called ‘‘application.bld’’.

2.10 Self-Test

The 80376 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 80376 can be tested during self-test.

Self-Test is initiated on the 80376 when the RESET pin transitions from HIGH to LOW, and the BUSY pin is LOW. The self-test takes about 2^{20} clocks, or approximately 33 ms with a 16 MHz 80376 processor. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register is zero. If the EAX register is not zero then the self-test has detected a flaw in the part. If self-test is not selected after reset, EAX may be non-zero after reset.

2.11 Debugging Support

The 80376 provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

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

BREAKPOINT INSTRUCTION

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

DEBUG REGISTERS

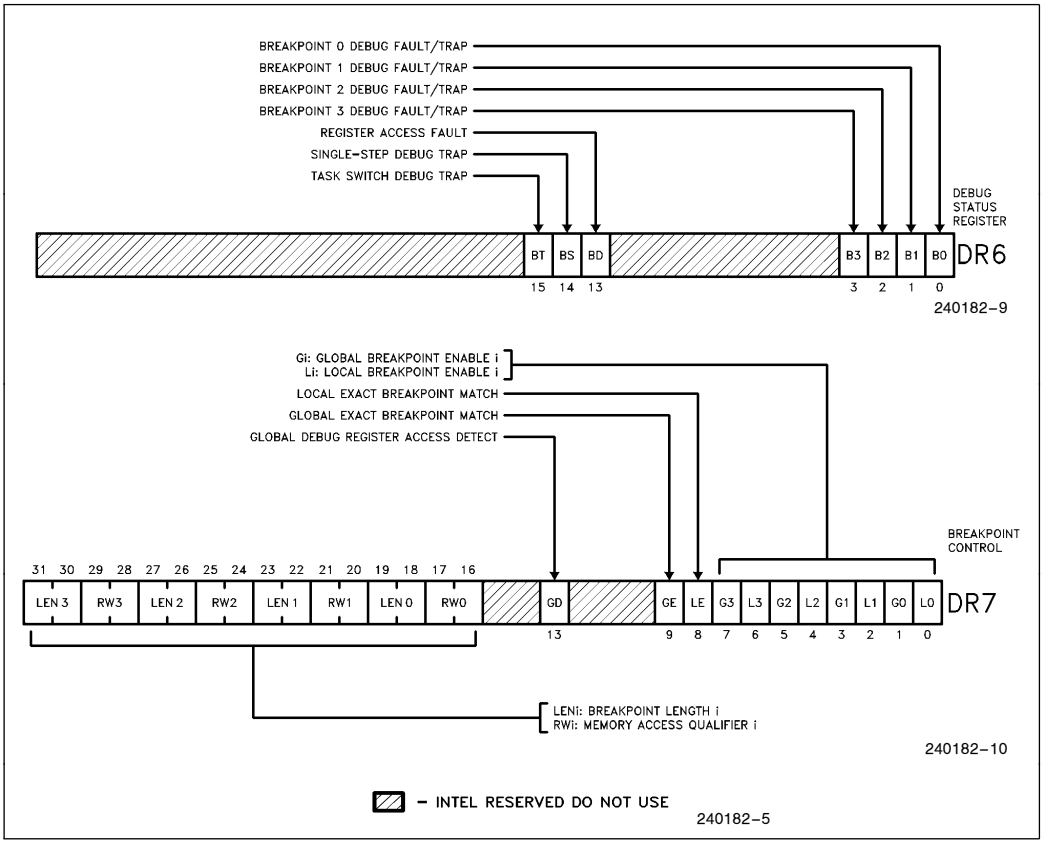


Figure 2.6. Debug Registers

SINGLE-STEP TRAP

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

The Debug Registers are an advanced debugging feature of the 80376. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 80376 contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.6 shows the breakpoint status and control registers.

3.0 ARCHITECTURE

The Intel 80376 Embedded Processor has a physical address space of 16 Mbytes (2^{24} bytes) and allows the running of virtual memory programs of almost unlimited size (16 Kbytes $\times$ 16 Mbytes or 256 Gbytes (2^{38} bytes)). In addition the 80376 provides a sophisticated memory management and a hardware-assisted protection mechanism.

3.1 Addressing Mechanism

The 80376 uses two components to form the logical address, a 16-bit selector which determines the linear base address of a segment, and a 32-bit effective address. The selector is used to specify an index into an operating system defined table (see Figure 3.1). The table contains the 32-bit base address of a given segment. The linear address is formed by adding the base address obtained from the table to the 32-bit effective address. This value is truncated to 24 bits to form the physical address, which is then placed on the address bus.

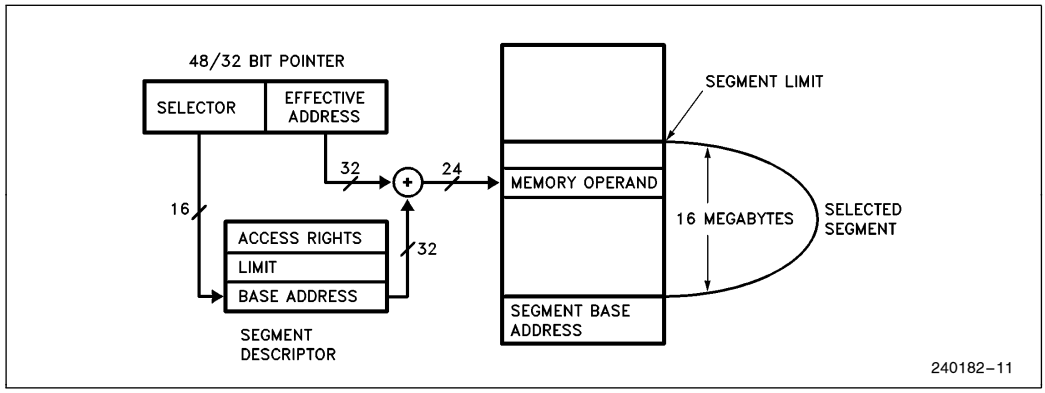


Figure 3.1. Address Calculation

3.2 Segmentation

Segmentation is one method of memory management and provides the basis for protection in the 80376. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment, is stored in an 8-byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

TERMINOLOGY

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

PL: **Privilege Level**—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.

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

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

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

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

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

DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in an 80376 system. There are three types of tables on the 80376 which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Decriptor Table. All of the tables are variable length memory arrays, they can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables have a register associated with it: GDTR, LDTR and IDTR; see Figure 3.2. 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 these base and limit values. These are privileged instructions.

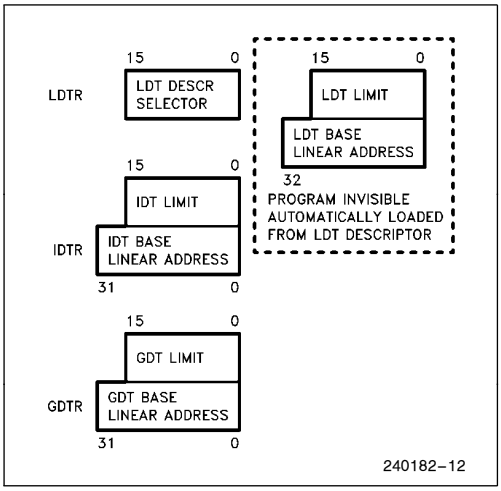


Figure 3.2. Descriptor Table Registers

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 interrupt and trap descriptors. Every 80376 system contains a GDT. A simple 80376 system contains only 2 entries in the GDT; a code and a data descriptor. For maximum performance, descriptor tables should begin on even addresses.

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

Local Descriptor Table

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

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

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

INTERRUPT DESCRIPTOR TABLE

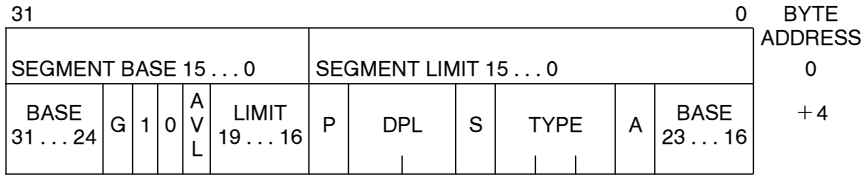
The third table needed for 80376 systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates and trap gates. The IDT should be at least 256 bytes in size in order 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 by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight-byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit logical base address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 3.3 shows the general format of a descriptor. All segments on the the 80376 have three attribute fields in common: the Present bit (P), the Descriptor Privilege Level bits (DPL) and the Segment bit (S). P=1 if the segment is loaded in physical memory, if P = 0 then any attempt to access the segment causes a not present exception (exception 11). The DPL is a two-bit field which specifies the protection level, 0–3, associated with a segment.

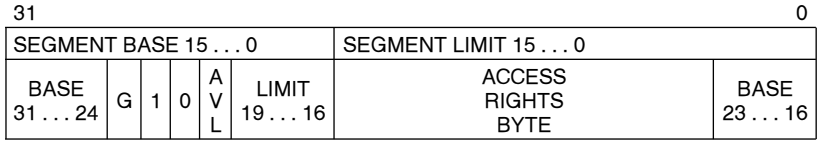
The 80376 has two main categories of segments: system segments, and non-system segments (for code and data). The segment bit, S, determines if a given segment is a system segment, a code segment or a data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

Note that although the 80376 is limited to a 16-Mbyte Physical address space (2^{24}), its base address allows a segment to be placed anywhere in a 4-Gbyte linear address space. When writing code for the 80376, users should keep code portability to an 80386 processor (or other processors with a larger physical address space) in mind. A segment base address can be placed anywhere in this 4-Gbyte linear address space, but a physical address will be



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 Descriptor
TYPE Type of Segment
A Accessed Bit
G Granularity Bit 1 = Segment length is 4 Kbyte Granular
0 = Segment length is byte granular
0 Bit must be zero (0) for compatibility with future processors
AVL Available field for user or OS

Figure 3.3. Segment Descriptors



G Granularity Bit 1 = Segment length is 4 Kbyte granular
0 = Segment length is byte granular
0 Bit must be zero (0) for compatibility with future processors
AVL Available field for user or OS

Figure 3.4. Code and Data Descriptors

Table 3.1. Access Rights Byte Definition for Code and Data Descriptors

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

generated that is a truncated version of this linear address. Truncation will be to the maximum number of address bits. It is recommended to place EPROM at the highest physical address and DRAM at the lowest physical addresses.

Code and Data Descriptors (S = 1)

Figure 3.4 shows the general format of a code and data descriptor and Table 3.1 illustrates how the bits in the Access Right Byte are interpreted.

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is 1-byte-granular or 4-Kbyte-granular. Base address bits 31–24, which are normally found in 80386 descriptors, are not made externally available on the 80376. They do not affect the operation of the 80376. The A_{31}–A_{24} field should be set to allow an 80386 to correctly execute with EPROM at the upper 4096 Mbytes of physical memory.

System Descriptor Formats (S = 0)

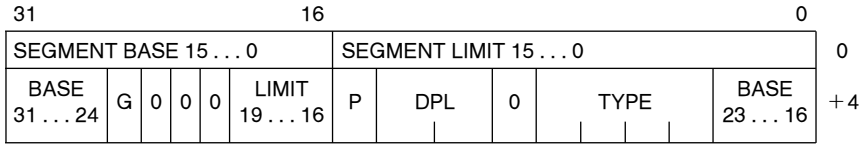
System segments describe information about operating system tables, tasks, and gates. Figure 3.5 shows the general format of system segment descriptors, and the various types of system segments. 80376 system descriptors (which are the same as 80386 descriptor types 2, 5, 9, B, C, E and F) contain a 32-bit logical base address and a 20-bit segment limit.

Selector Fields

A selector 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 3.6. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

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



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Reserved	9	Available 80376/80386 TSS
2	LDT	A	Undefined (Intel Reserved)
3	Reserved	B	Busy 80376/80386 TSS
4	Reserved	C	80376/80386 Call Gate
5	Task Gate (80376/80386 Task)	D	Undefined (Intel Reserved)
6	Reserved	E	80376/80386 Interrupt Gate
7	Reserved	F	80376/80386 Trap Gate

Figure 3.5. System Descriptors

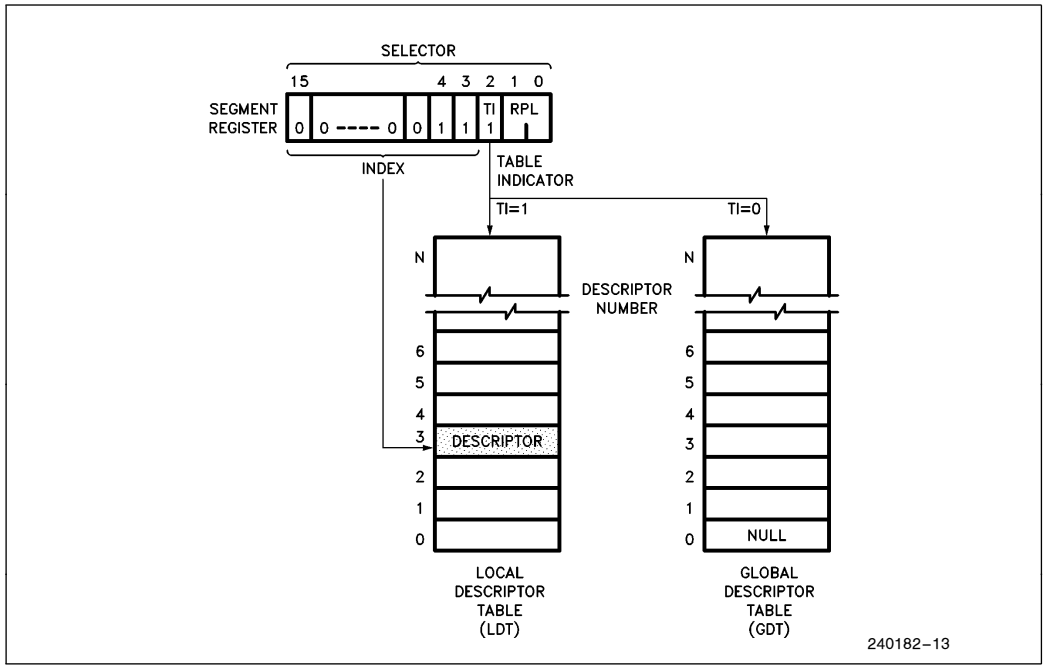


Figure 3.6. Example Descriptor Selection

3.3 Protection

The 80376 offers extensive protection features. These protection features are particularly useful in sophisticated embedded applications which use multitasking real-time operating systems. For simpler embedded applications these protection capabilities can be easily bypassed by making all applications run at privilege level (PL) 0.

RULES OF PRIVILEGE

The 80376 controls access to both data and procedures between levels of a task, according to the following rules.

—Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p.**

—A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p.**

PRIVILEGE LEVELS

At any point in time, a task on the 80376 always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed

by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

I/O Privilege

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

Descriptor Access

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

Any time an instruction loads a data segment register (DS, ES, FS, GS) the 80376 makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

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

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

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 3.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

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

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.

Table 3.2. Descriptor Types Used for Control Transfer

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

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

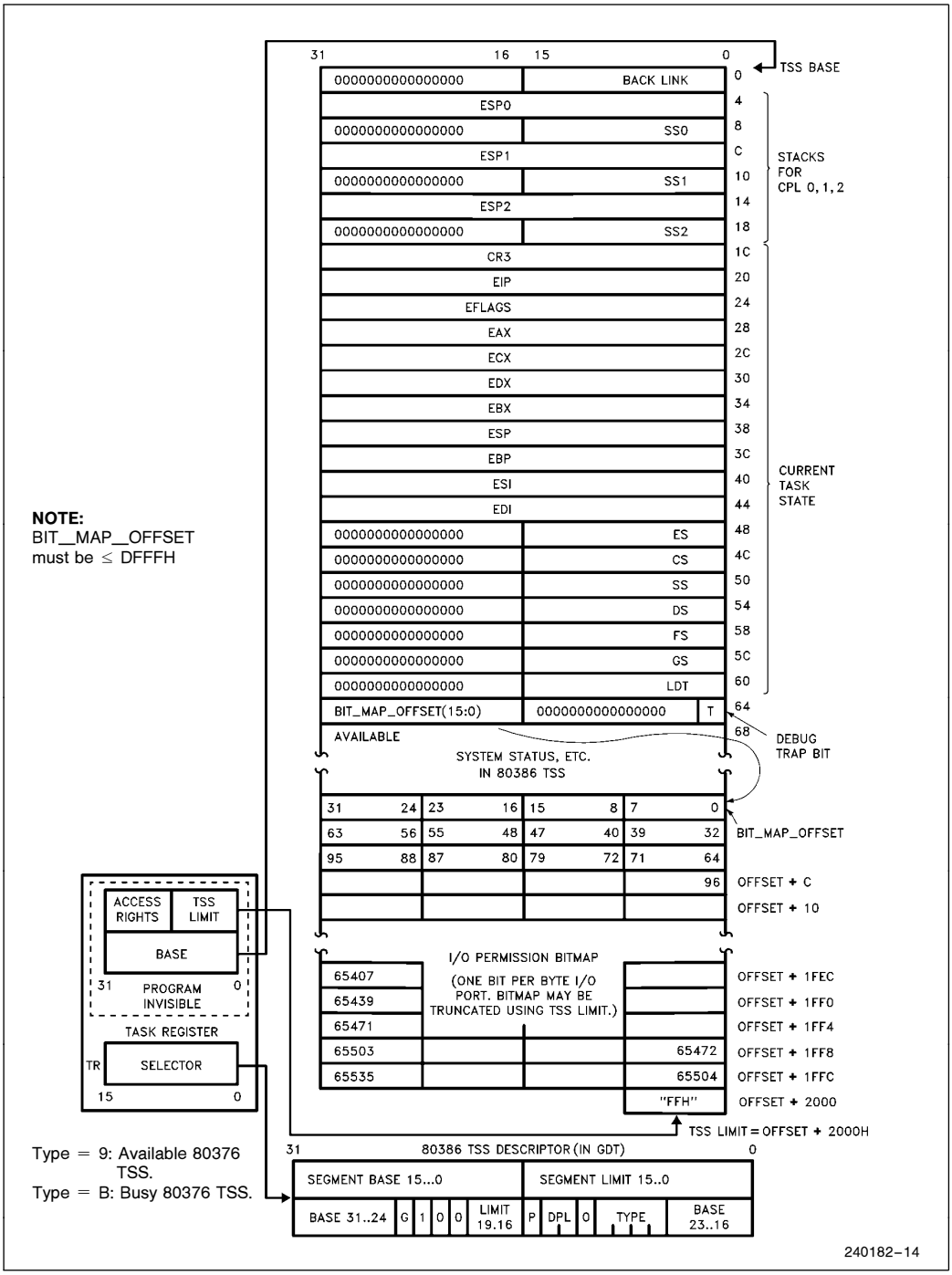


Figure 3.7. 80376 TSS And TSS Registers

TASK SWITCHING

A very important attribute of any multi-tasking operating system is its ability to rapidly switch between tasks or processes. The 80376 directly supports this operation by providing a task switch instruction in hardware. The 80376 task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot. For simple applications, the TSS and task switching may not be used. The TSS or task switch will not be used or occur if no task gates are present in the GDT, LDT or IDT.

The TSS descriptor points to a segment (see Figure 3.7) containing the entire 80376 execution state. A task gate descriptor contains a TSS selector. The limit of an 80376 TSS must be greater than 64H, and can be as large as 16 Mbytes. In the additional TSS space, the operating system is free to store additional information as the reason the task is inactive, the time the task has spent running, and open files belonging to the task. For maximum performance, TSS should start on an even address.

Each Task must have a TSS associated with it. The current TSS is identified by a special register in the 80376 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 the TSS descriptor is 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 CR0 register give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT = 0 the IRET instruction performs the regular return. If NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

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

The 80376 task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. Use of a selector that references a busy task state segment causes an exception 13.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CR0 register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80376 switches tasks, it sets the TS bit. The 80376 detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the 80376 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.

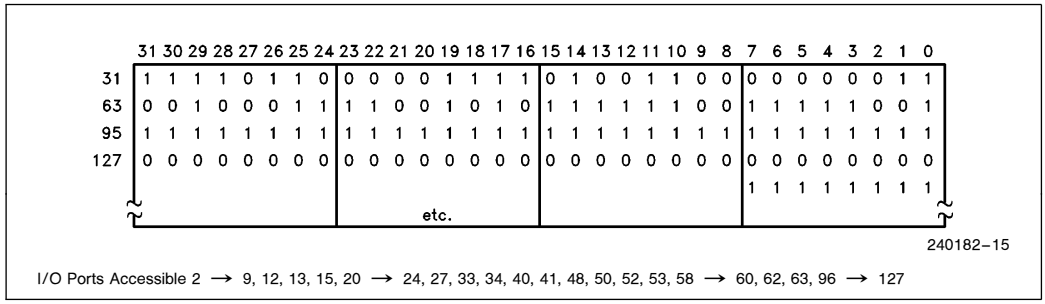


Figure 3.8. Sample I/O Permission Bit Map

PROTECTION AND I/O PERMISSION BIT MAP

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT and OUTS. The 80376 has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the ***I/O Permission Bit Map*** in the TSS segment (see Figures 3.7 and 3.8). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the **I/O map base** field in the fixed portion of the TSS. The **I/O map base** field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

If an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether CPL $\leq$ IOPL. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map.

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at **I/O map base** $+5$ linearly, ($5 \times 8 = 40$), bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

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

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

IMPORTANT IMPLEMENTATION NOTE:
Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 80376's TSS segment (see Figure 3.7).

4.0 FUNCTIONAL DATA

The Intel 80376 embedded processor features a straightforward functional interface to the external hardware. The 80376 has separate parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus outputs 24-bit address values using 23 address lines and two-byte enable signals.

The 80376 has two selectable address bus cycles: pipelined and non-pipelined. The pipelining option allows as much time as possible for data access by

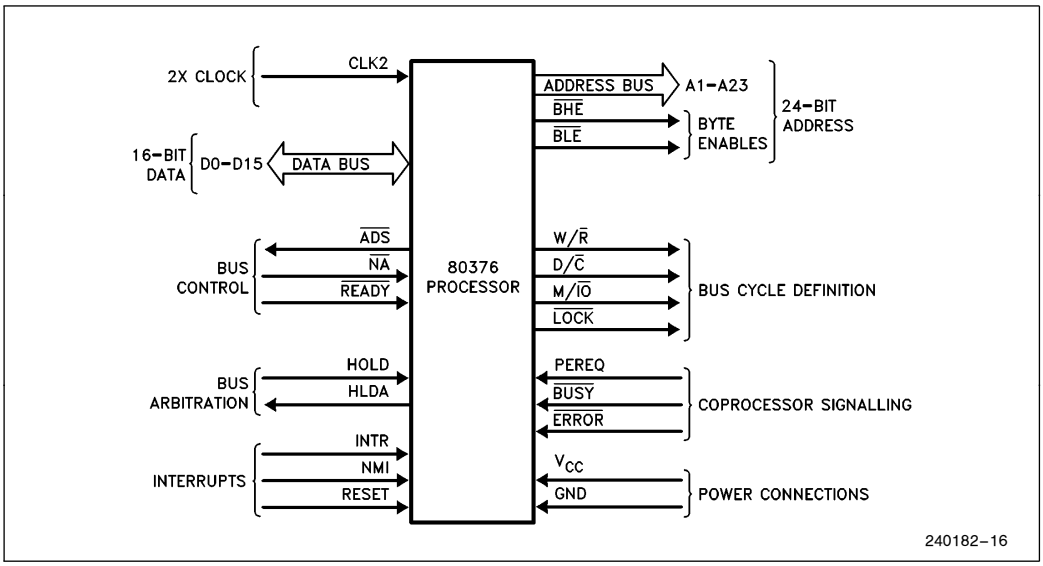


Figure 4.1. Functional Signal Groups

starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor clock 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. 80376 bus cycles perform data transfer in a minimum of only two clock periods. On a 16-bit data bus, the maximum 80376 transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 80376 can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 80376, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

4.1 Signal Description Overview

Ahead is a brief description of the 80376 input and output signals arranged by functional groups.

The signal descriptions sometimes refer to A.C. 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 6.4 and 6.5.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 80376. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two

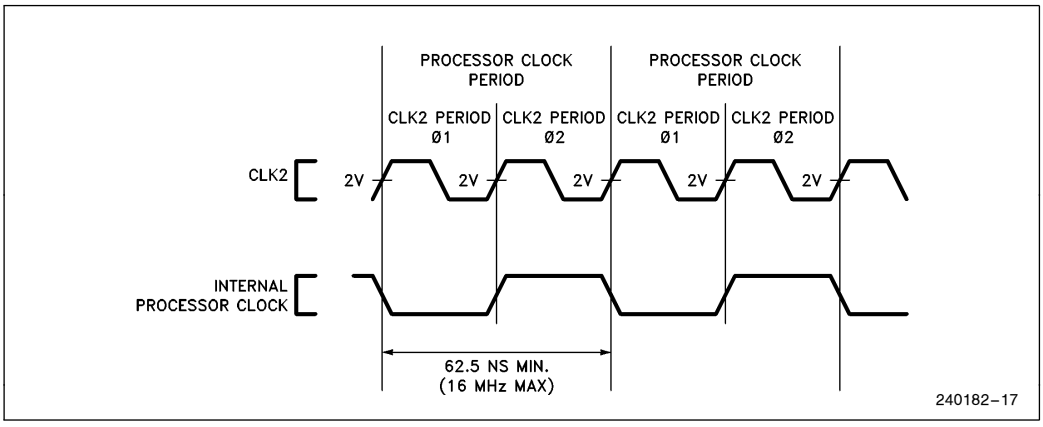


Figure 4.2. CLK2 Signal and Internal Processor Clock

phases, ''phase one'' and ''phase two''. Each CLK2 period is a phase of the internal clock. Figure 4.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26}.

DATA BUS (D_{15}–D_0)

These three-state bidirectional signals provide the general purpose data path between the 80376 and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

ADDRESS BUS ($\overline{BHE}$, $\overline{BLE}$, A_{23}–A_1)

These three-state outputs provide physical memory addresses or I/O port addresses. A_{23}–A_{16} are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A_{22}–A_{16} are driven LOW, and A_{23} is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 80376 for coprocessor commands is 8000F8H, and the I/O address driven by the 80376 processor for coprocessor data is 8000FCH or 8000FEH.

The address bus is capable of addressing 16 Mbytes of physical memory space (000000H through 0FFFFFFH), and 64 Kbytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs $\overline{BHE}$ and $\overline{BLE}$ directly indicate which bytes of the 16-bit data bus are involved with the current transfer. $\overline{BHE}$ applies to D_{15}–D_8 and $\overline{BLE}$ applies to D_7–D_0. If both $\overline{BHE}$ and $\overline{BLE}$ are asserted, then 16 bits of data are being transferred. See Table 4.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

Table 4.1. Byte Enable Definitions

$\overline{BHE}$	$\overline{BLE}$	Function
0	0	Word Transfer
0	1	Byte Transfer on Upper Byte of the Data Bus, D_{15}–D_8
1	0	Byte Transfer on Lower Byte of the Data Bus, D_7–D_0
1	1	Never Occurs

BUS CYCLE DEFINITION SIGNALS (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$)

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

The primary bus cycle definition signals are W/$\overline{R}$, D/$\overline{C}$ and M/$\overline{IO}$, since these are the signals driven valid as $\overline{ADS}$ (Address Status output) becomes active. The $\overline{LOCK}$ signal is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after $\overline{ADS}$ becomes active. Exact bus cycle definitions, as a function of W/$\overline{R}$, D/$\overline{C}$ and M/$\overline{IO}$ are given in Table 4.2.

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

BUS CONTROL SIGNALS ($\overline{ADS}$, $\overline{READY}$, $\overline{NA}$)

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

Address Status ($\overline{ADS}$)

This three-state output indicates that a valid bus cycle definition and address (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{BHE}$, $\overline{BLE}$ and A_{23}–A_1) are being driven at the 80376 pins. $\overline{ADS}$ is an active LOW output. Once $\overline{ADS}$ is driven active, valid address, byte enables, and definition signals will not change. In addition, $\overline{ADS}$ will remain active until its associated bus cycle begins (when $\overline{READY}$ is returned for the previous bus cycle when running pipelined bus cycles). $\overline{ADS}$ will float during bus hold acknowledge. See sections **Non-Pipelined Bus Cycles** and **Pipelined Bus Cycles** for additional information on how $\overline{ADS}$ is asserted for different bus states.

Transfer Acknowledge ($\overline{READY}$)

This input indicates the current bus cycle is complete, and the active bytes indicated by $\overline{BHE}$ and $\overline{BLE}$ are accepted or provided. When $\overline{READY}$ is sampled active during a read cycle or interrupt acknowledge cycle, the 80376 latches the input data and terminates the cycle. When $\overline{READY}$ is sampled active during a write cycle, the processor terminates the bus cycle.

Table 4.2. Bus Cycle Definition

M/$\overline{IO}$	D/$\overline{C}$	W/$\overline{R}$	Bus Cycle Type	Locked?
0	0	0	INTERRUPT ACKNOWLEDGE	Yes
0	0	1	Does Not Occur	—
0	1	0	I/O DATA READ	No
0	1	1	I/O DATA WRITE	No
1	0	0	MEMORY CODE READ	No
1	0	1	HALT: Address = 2, $\overline{BHE}$ = 1, $\overline{BLE}$ = 0 — SHUTDOWN: Address = 0, $\overline{BHE}$ = 1, $\overline{BLE}$ = 0	No
1	1	0	MEMORY DATA READ	Some Cycles
1	1	1	MEMORY DATA WRITE	Some Cycles

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

Next Address Request (NA)

This is used to request pipelining. This input indicates the system is prepared to accept new values of BHE, BLE, A_{23}–A_1, W/R, D/C and M/IO from the 80376 even if the end of the current cycle is not being acknowledged on READY. If this input is active when sampled, the next bus cycle's address and status signals are driven onto the bus, provided the next bus request is already pending internally. NA is ignored in clock cycles in which ADS or READY is activated. This signal is active LOW and must satisfy setup and hold times t_{15} and t_{16} for correct operation. See **Pipelined Bus Cycles** and **Read and Write Cycles** for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

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

Bus Hold Request (HOLD)

This input indicates some device other than the 80376 requires bus mastership. When control is granted, the 80376 floats A_{23}–A_1, BHE, BLE, D_{15}–D_0, LOCK, M/IO, D/C, W/R and ADS, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 80376 will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 80376 floated outputs have internal pull-up resistors. See **Resistor Recommendations** for additional information. HOLD is not recognized while RESET is active but is recognized during the time between the high-to-low transistion of RESET and the first instruction fetch. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high-impedance) state.

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

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 80376 has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 80376. The other output signals or bidirectional signals (D_{15}–D_0, BHE, BLE, A_{23}–A_1, W/R, D/C, M/IO, LOCK and ADS) are in a high-impedance state so the requesting bus master may control them. These pins remain OFF throughout the time that HLDA remains active (see Table 4.3). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** for additional information.

When the HOLD signal is made inactive, the 80376 will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 4.3. Output Pin State during HOLD

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

Hold Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK signal (internal to the CPU) activated by the LOCK prefix, and interrupts. The 80376 will not honor a HOLD request until the current bus operation is complete.

The 80376 breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK signal is not asserted. The 80376 breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again the LOCK signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The 80376 will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that $\overline{\text{READY}}$ returns sufficiently soon.

COPROCESSOR INTERFACE SIGNALS (PEREQ, $\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$)

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 80376 and the 80387SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 80376. In response, the 80376 transfers information between the coprocessor and memory. Because the 80376 has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

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

Coprocessor Busy ($\overline{\text{BUSY}}$)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 80376 encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the $\overline{\text{BUSY}}$ input prevents overrunning the execution of a previous coprocessor instruction.

The F(N)INIT, F(N)CLEX coprocessor instructions are allowed to execute even if $\overline{\text{BUSY}}$ is active, since these instructions are used for coprocessor initialization and exception-clearing.

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

$\overline{\text{BUSY}}$ serves an additional function. If $\overline{\text{BUSY}}$ is sampled LOW at the falling edge of RESET, the 80376 processor performs an internal self-test (see **Bus Activity During and Following Reset.** If $\overline{\text{BUSY}}$ is sampled HIGH, no self-test is performed.

Coprocessor Error ($\overline{\text{ERROR}}$)

When asserted (LOW), this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 80376 when a coprocessor instruction is encountered, and if active, the 80376 generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 80376 generating exception 16 even if $\overline{\text{ERROR}}$ is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

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

INTERRUPT SIGNALS (INTR, NMI, RESET)

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

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 80376 Flag Register IF bit. When the 80376 responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D_7–D_0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the execution of the instruction. If recognized, the 80376 will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI)

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

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

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

Interrupt Latency

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

1. If interrupts are masked, and INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 80376 encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 80376 ***Execution Unit*** except for the following cases:
   - — Repeat string instructions can be interrupted after each iteration.
   - — If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP load. This allows the entire stack pointer to be loaded without interruption.
   - — If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

   The longest latency occurs when the interrupt request arrives while the 80376 processor is executing a long instruction such as multiplication, division or a task-switch.
4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 80376.

RESET

This input signal suspends any operation in progress and places the 80376 in a known reset state. The 80376 is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins except $\overline{FLT}$ are ignored, and all other bus pins are driven to an idle bus state as shown in Table 4.4. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 80376 was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26}, must be met in order to assure proper operation of the 80376.

Table 4.4. Pin State (Bus Idle) during RESET

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

4.2 Bus Transfer Mechanism

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

The 80376 processor address signals are designed to simplify external system hardware. $\overline{BHE}$ and $\overline{BLE}$ provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs $\overline{BHE}$ and $\overline{BLE}$ are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 4.5.

Table 4.5. Byte Enables and Associated Data and Operand Bytes

Byte Enable	Associated Data Bus Signals
$\overline{BHE}$	$D_{15}-D_8$ (Byte 1—Most Significant)
$\overline{BLE}$	D_7-D_0 (Byte 0—Least Significant)

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

4.3 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 4.3, physical memory addresses range from 000000H to 0FFFFFFH (16 Mbytes) and I/O addresses from 000000H to 00FFFFH (64 Kbytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A_{23} and $M/\overline{IO}$ signals.

OPERAND ALIGNMENT

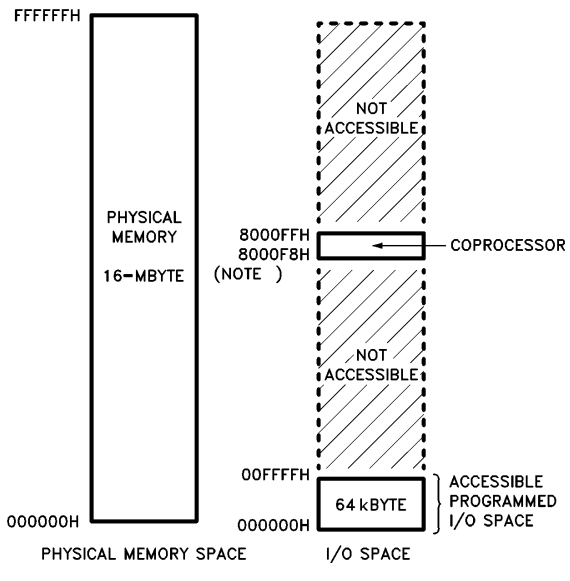
With the flexibility of memory addressing on the 80376, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dword or 16-bit word operands beginning at addresses not evenly divisible by 2.

Operand alignment and size dictate when multiple bus cycles are required. Table 4.6 describes the transfer cycles generated for all combinations of logical operand lengths and alignment.

Table 4.6. 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	b	w	lb, hb	w	hb, l,b	lw, hw	hb, lb, mw	hw, lw	mw, hb, lb

Key: b = byte transfer
w = word transfer
l = low-order portion
m = mid-order portion
x = don't care
h = high-order portion



NOTE:
Since A_{23} is HIGH during automatic communication with coprocessor, A_{23} HIGH and M/$\overline{IO}$ LOW can be used to easily generate a coprocessor select signal.

Figure 4.3. Physical Memory and I/O Spaces

4.4 Bus Functional Description

The 80376 has separate, parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The definition of each bus cycle is given by three signals: M/$\overline{IO}$, W/$\overline{R}$ and D/$\overline{C}$. At the same time, a valid address is present on the byte enable signals, $\overline{BHE}$ and $\overline{BLE}$, and the other address signals A_{23}–A_1. A status signal, $\overline{ADS}$, indicates when the 80376 issues a new bus cycle definition and address.

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

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

Table 4.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** for additonal information.

When the 80376 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 80376 giving no further assertions on its address strobe output ($\overline{ADS}$) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the 80376 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.

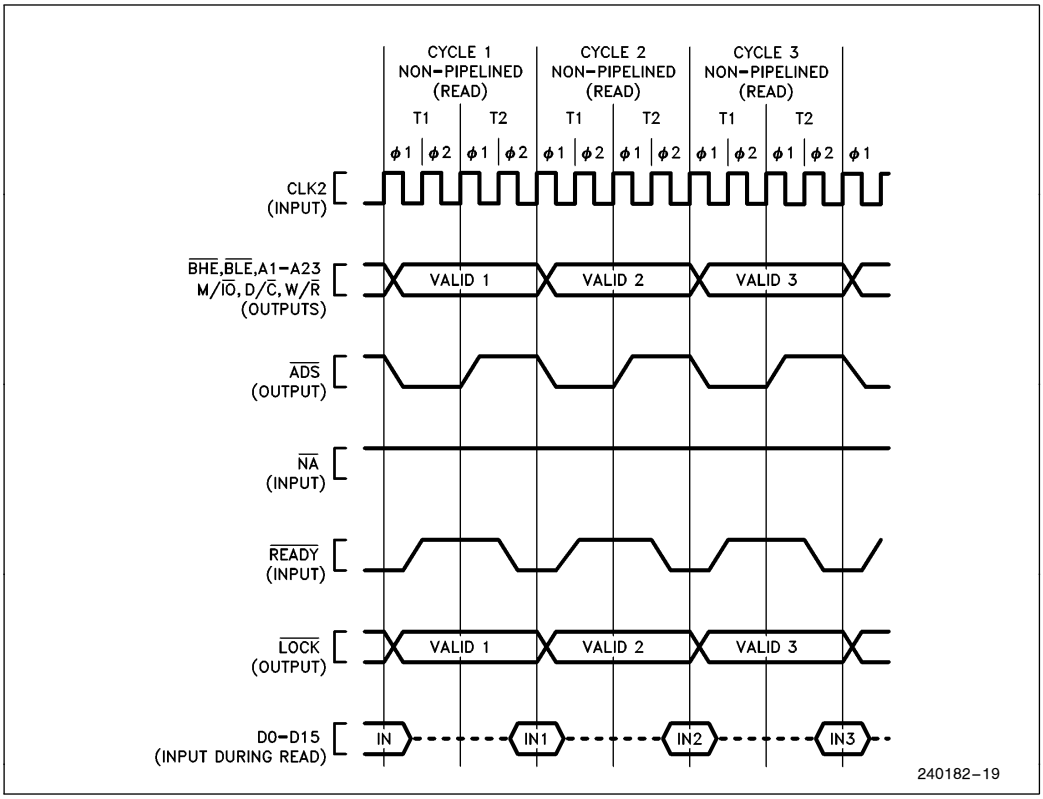


Figure 4.4. Fastest Read Cycles with Non-Pipelined Timing

The fastest 80376 bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 4.4. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 80376 $\overline{\text{READY}}$ input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If $\overline{\text{READY}}$ is not immediately asserted however, T2 states are repeated indefinitely until the $\overline{\text{READY}}$ input is sampled active.

The pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined cycles are selectable on a cycle-by-cycle basis with the Next Address ($\overline{\text{NA}}$) input.

When pipelining is selected the address ($\overline{\text{BHE}}$, $\overline{\text{BLE}}$ and A_{23}–A_1) and definition (W/$\overline{\text{R}}$, D/$\overline{\text{C}}$, M/$\overline{\text{IO}}$ and $\overline{\text{LOCK}}$) of the next cycle are available before the end of the current cycle. To signal their availability, the 80376 address status output ($\overline{\text{ADS}}$) is asserted. Figure 4.5 illustrates the fastest read cycles with pipelined timing.

Note from Figure 4.5 the fastest bus cycles using pipelining require only two bus states, named **T1P** and **T2P**. Therefore pipelined cycles allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.

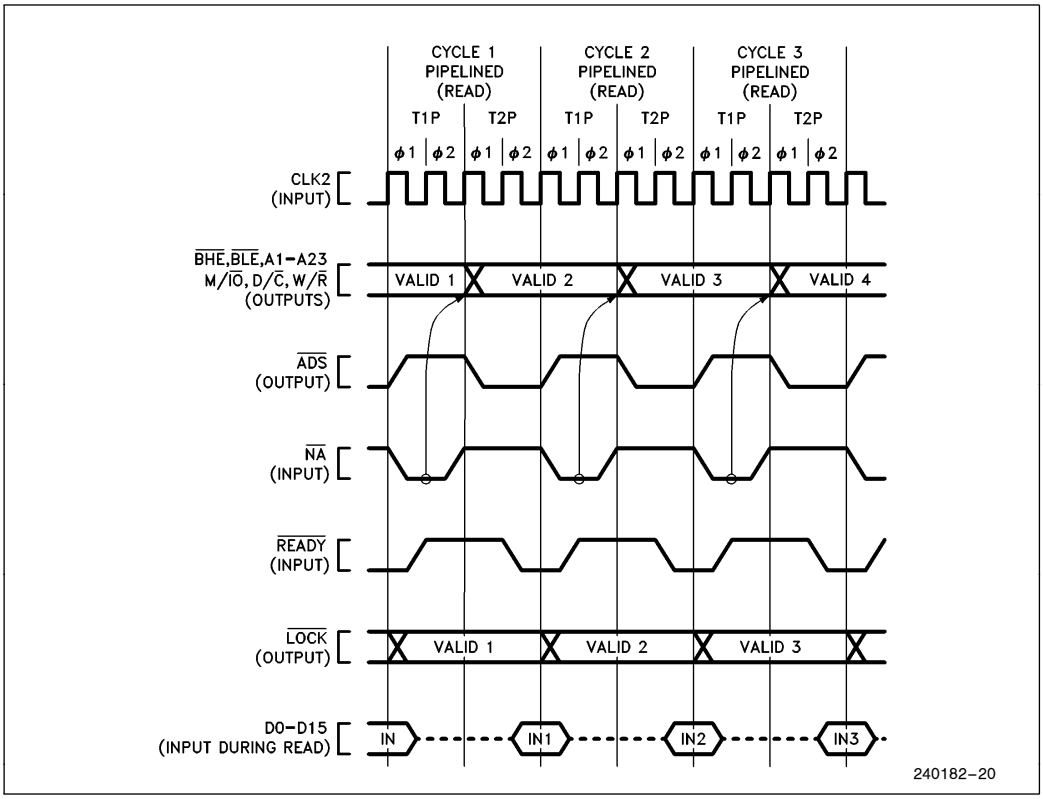


Figure 4.5. Fastest Read Cycles with Pipelined Timing

READ AND WRITE CYCLES

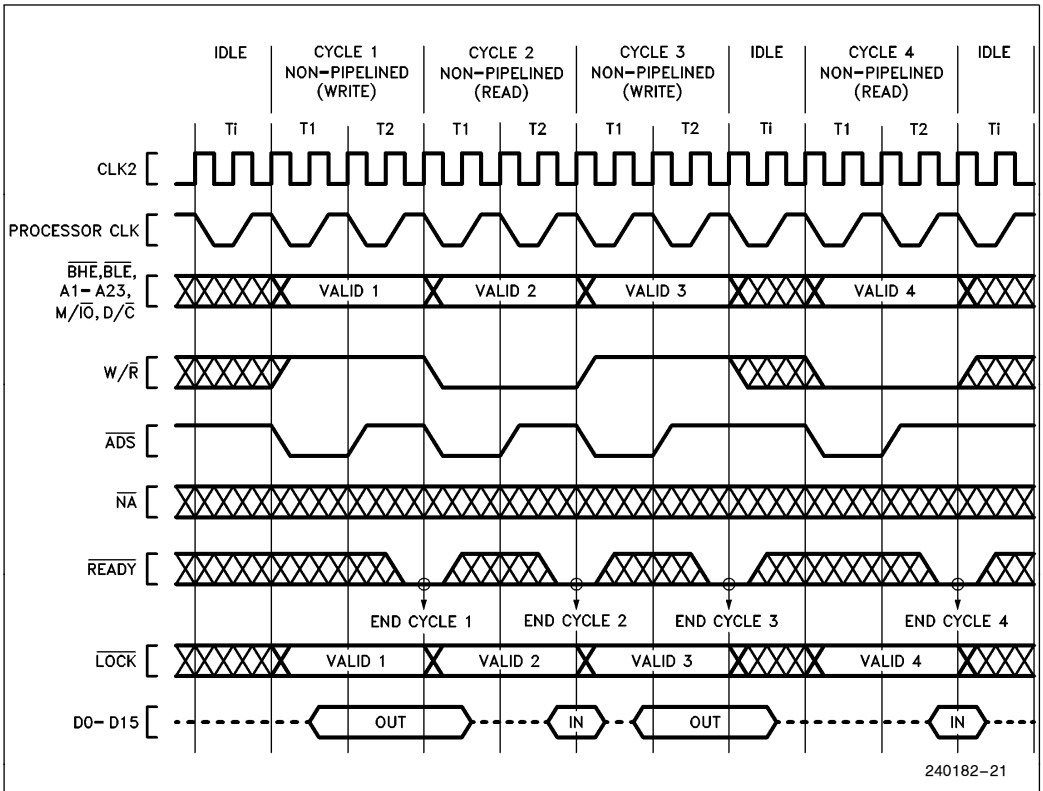
Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.

Two choices of bus cycle timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined timing. However the $\overline{NA}$ (Next Address) input may be asserted to select pipelined timing for the next bus cycle. When pipelining is selected and the 80376 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 $\overline{READY}$.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the $\overline{READY}$ 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 $\overline{READY}$ input at the appropriate time.

At the end of the second bus state within the bus cycle, $\overline{READY}$ is sampled. At that time, if external hardware acknowledges the bus cycle by asserting $\overline{READY}$, the bus cycle terminates as shown in Figure 4.6. If $\overline{READY}$ is negated as in Figure 4.7, the 80376 executes another bus state (a wait state) and $\overline{READY}$ is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by $\overline{READY}$ asserted.

When the current cycle is acknowledged, the 80376 terminates it. When a read cycle is acknowledged, the 80376 latches the information present at its data pins. When a write cycle is acknowledged, the write data of the 80376 remains valid throughout phase one of the next bus state, to provide write data hold time.



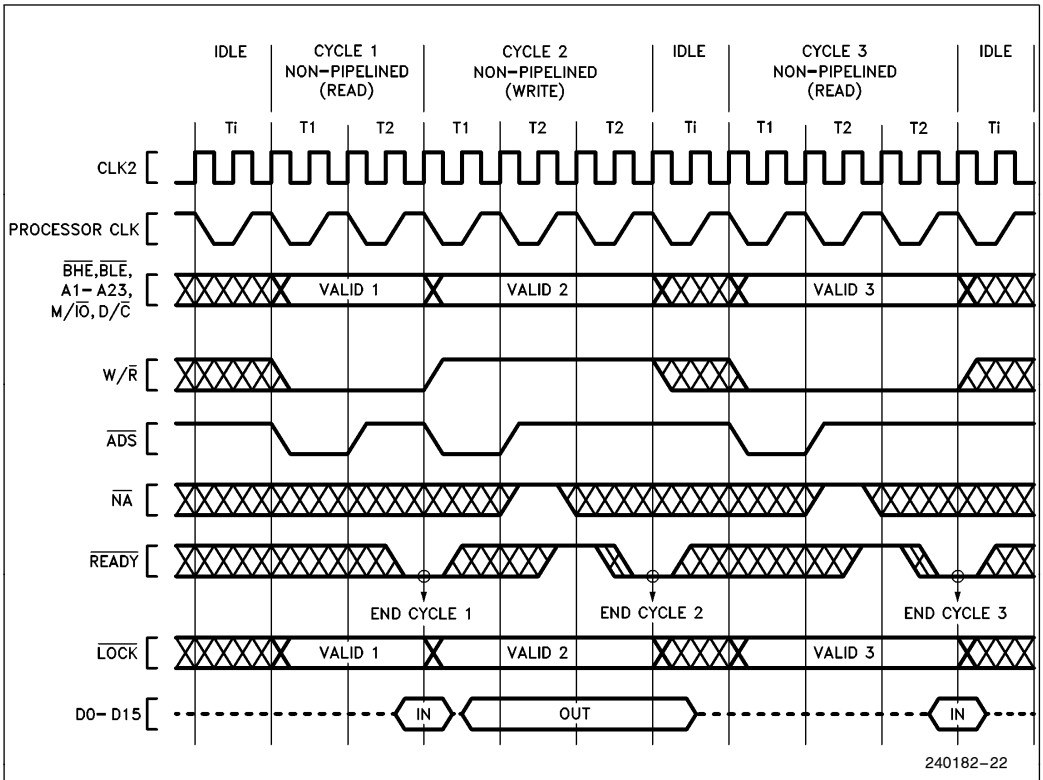
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 4.6. Various Non-Pipelined Bus Cycles (Zero Wait States)

Non-Pipelined Bus Cycles

Any bus cycle may be performed with non-pipelined timing. For example, Figure 4.6 shows a mixture of non-pipelined read and write cycles. Figure 4.6 shows that the fastest possible non-pipelined cycles have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe ($\overline{ADS}$) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 80376 floats its data signals to allow driving by the external device being addressed. **The 80376 requires that all data bus pins be at a valid logic state (HIGH or LOW) at the end of each read cycle, when $\overline{READY}$ is asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 80376 beginning in phase two of T1 until phase one of the bus state following cycle acknowledgement.



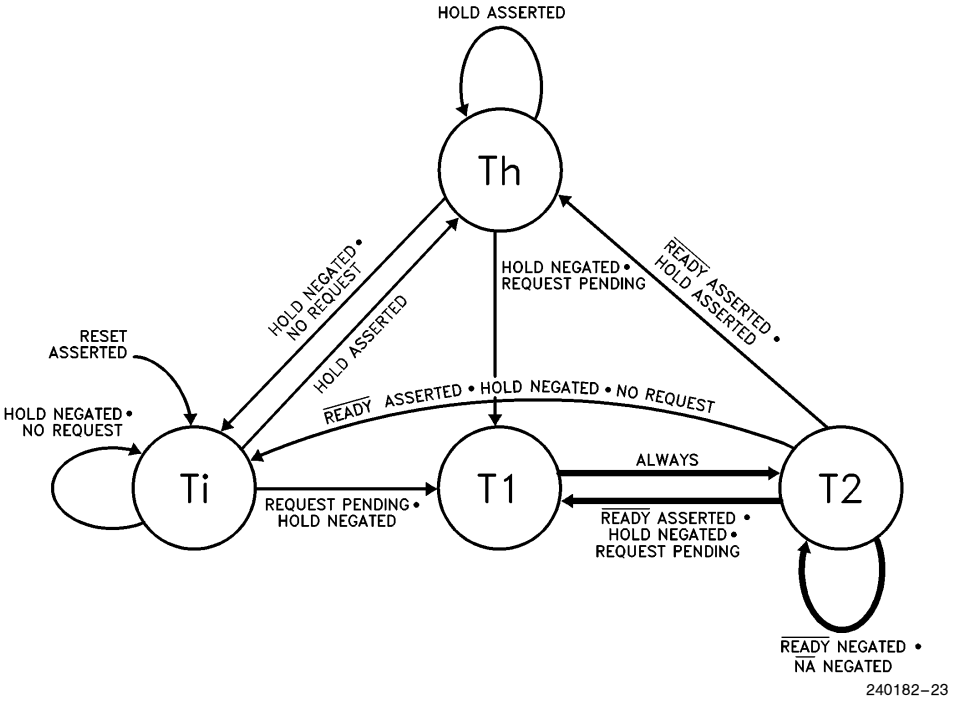
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 4.7. Various Non-Pipelined Bus Cycles (Various Number of Wait States)

Figure 4.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. $\overline{READY}$ is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, $\overline{READY}$ is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined timing, it is necessary to negate $\overline{NA}$ during each T2 state except the last one, as shown in Figure 4.7, Cycles 2 and 3. If $\overline{NA}$ is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 4.8. The bus transitions between four possible states, T1, T2, T_i, and T_h. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, T_i, or in the hold acknowledge state T_h.



Bus States:
T1—first clock of a non-pipelined bus cycle (80376 drives new address and asserts $\overline{ADS}$).
T2—subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
Ti—idle state.
Th—hold acknowledge state (80376 asserts HLDA).
The fastest bus cycle consists of two states: T1 and T2.
Four basic bus states describe bus operation when not using pipelined address.

Figure 4.8. 80376 Bus States (Not Using Pipelined Address)

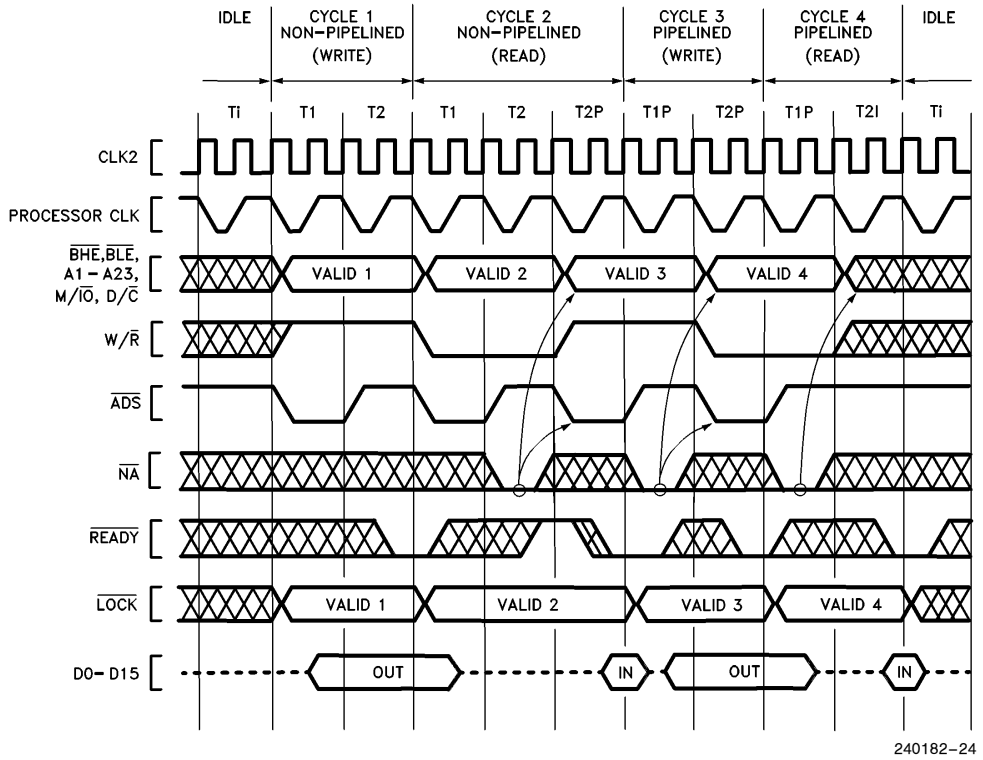
Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and $\overline{NA}$ is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or T_i if there is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

Use of pipelining allows the 80376 to enter three additional bus states not shown in Figure 4.8. Figure 4.12 is the complete bus state diagram, including pipelined cycles.

Pipelined Bus Cycles

Pipelining is the option of requesting the address and the bus cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with $\overline{READY}$ asserted. $\overline{ADS}$ is asserted by the 80376 when the next address is issued. The pipelining option is controlled on a cycle-by-cycle basis with the $\overline{NA}$ input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles $\overline{NA}$ is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 4.9, during which $\overline{NA}$ is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).



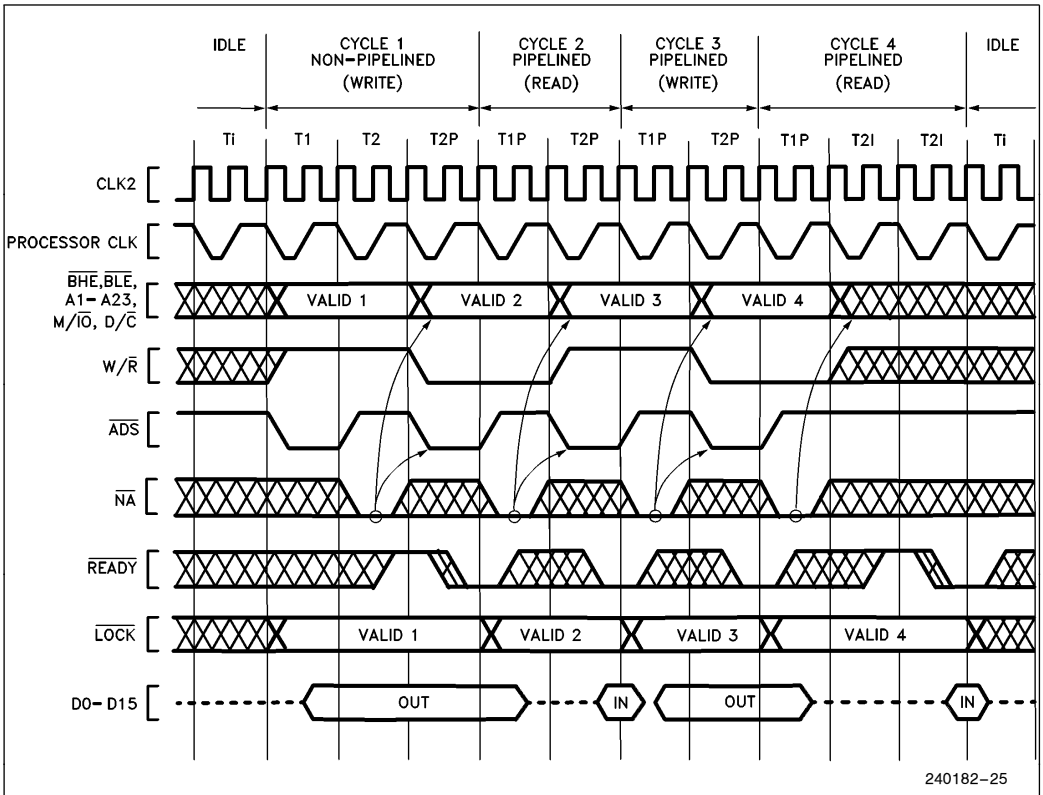
Following any idle bus state (Ti), bus cycles are non-pipelined. Within non-pipelined bus cycles, $\overline{NA}$ is only sampled during wait states. Therefore, to begin pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cylcle 2 above).

Figure 4.9. Transitioning to Pipelining during Burst of Bus Cycles

If $\overline{NA}$ is sampled active, the 80376 is free to drive the address and bus cycle definition of the next bus cycle, and assert $\overline{ADS}$, 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 pipelining, the 80376 has the following characteristics:

1. The next address and status may appear as early as the bus state after $\overline{NA}$ was sampled active (see Figures 4.9 or 4.10). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address and status will not be available immediately after $\overline{NA}$ is asserted and T2I is entered instead of T2P (see Figure 4.11 Cycle 3). Provided the current bus cycle isn't yet acknowledged by $\overline{READY}$ asserted, T2P will be entered as soon as the 80376 does drive the next address and status. External hardware should therefore observe the $\overline{ADS}$ output as confirmation the next address and status are actually being driven on the bus.
2. Any address and status which are validated by a pulse on the 80376 $\overline{ADS}$ output will remain stable on the address pins for at least two processor clock periods. The 80376 cannot produce a new address and status more frequently than every two processor clock periods (see Figures 4.9, 4.10 and 4.11).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 4.11, Cycle 1).



Following any idle bus state (Ti) the bus cycle is always non-pipelined and $\overline{NA}$ 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 4.10. Fastest Transition to Pipelined Bus Cycle Following Idle Bus State

The complete bus state transition diagram, including pipelining is given by Figure 4.12. Note it is a superset of the diagram for non-pipelined only, and the three additional bus states for pipelining are drawn in bold.

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

Initiating and Maintaining Pipelined Bus Cycles

Using the state diagram Figure 4.12, observe the transitions from an idle state, T_i, to the beginning of a pipelined bus cycle T1P. From an idle state, T_i, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided $\overline{NA}$ is asserted and the first bus cycle ends in a T2P state (the address and status for the next bus cycle is driven during T2P). The fastest path from an idle state to a pipelined bus cycle is shown in bold below:

T_i, T_i,	**T1–T2–T2P,**	**T1P–T2P,**
idle states	non-pipelined cycle	pipelined cycle

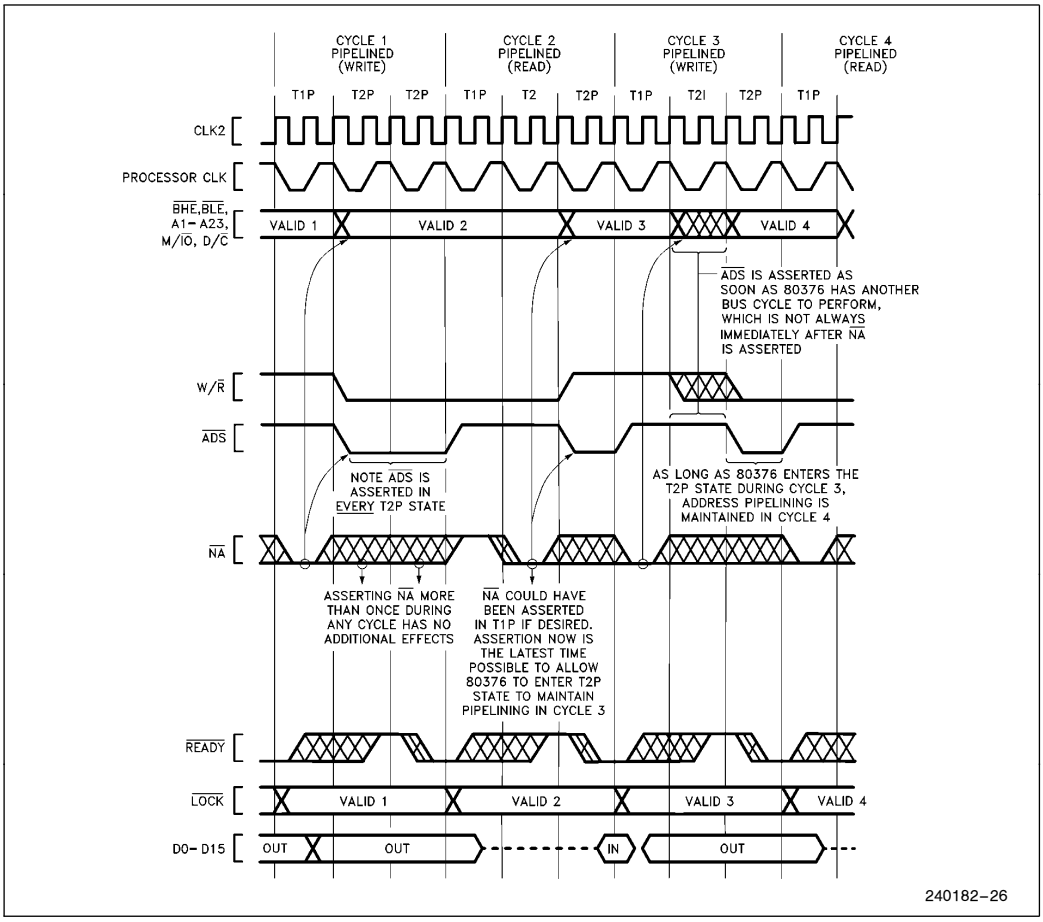


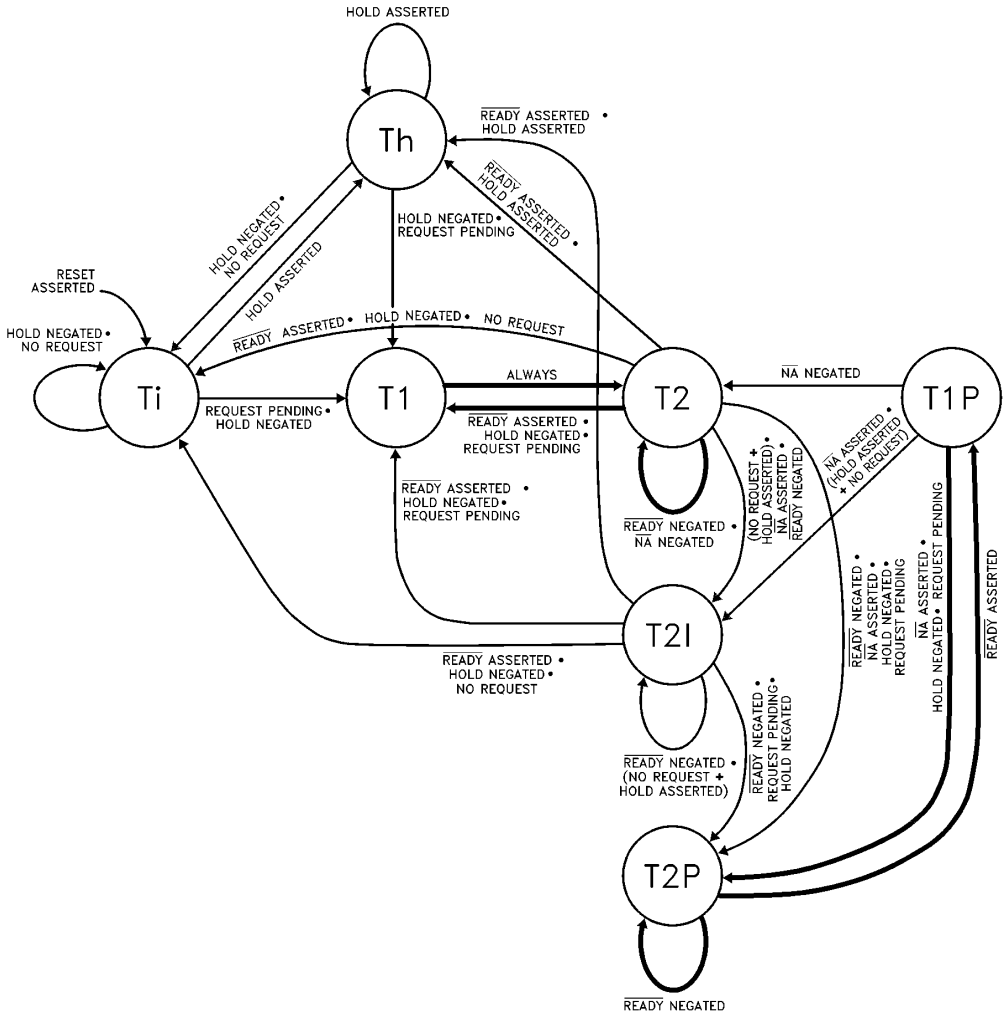
Figure 4.11. Details of Address Pipelining during Cycles with Wait States

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:

T_h, T_h, T_h,	T1–T2–T2P,	T1P–T2P,
hold aknowledge states	non-pipelined cycle	pipelined cycle

The transition to pipelined address is shown functionally by Figure 4.10, Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The $\overline{NA}$ input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address and status has been valid for one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged.



Bus States:
T1—first clock of a non-pipelined bus cycle (80376 drives new address, status and asserts $\overline{ADS}$).
T2—subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
T2I—subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (80376 will not drive new address, status or assert $\overline{ADS}$).
T2P—subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle and there is an internal bus request pending (80376 drives new address, status and asserts $\overline{ADS}$).
T1P—first clock of a pipelined bus cycle.
Ti—idle state.
Th—hold acknowledge state (80376 asserts HLDA).
Asserting $\overline{NA}$ for pipelined bus cycles gives access to three more bus states: T2I, T2P and T1P.
Using pipelining the fastest bus cycle consists of T1P and T2P.

Figure 4.12. 80376 Processor Complete Bus States (Including Pipelining)

Sampling begins in T2 during Cycle 1 in Figure 4.10. Once $\overline{NA}$ is sampled active during the current cycle, the 80376 is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 4.10, Cycle 1 for example, the next address and status is driven during state T2P. Thus Cycle 1 makes the transition to pipelined timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as $\overline{READY}$ asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 4.10, Cycle 1 and Figure 4.9, Cycle 2. Figure 4.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 4.9, Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 ($\overline{NA}$ is asserted at that time), and T2P (provided the 80376 has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined into pipelined timing, for example Figure 4.10, Cycle 1. Figure 4.10, Cycles 2, 3 and 4 show that pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting $\overline{NA}$ and detecting that the 80376 enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of $\overline{ADS}$. Figures 4.9 and 4.10 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 80376 didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

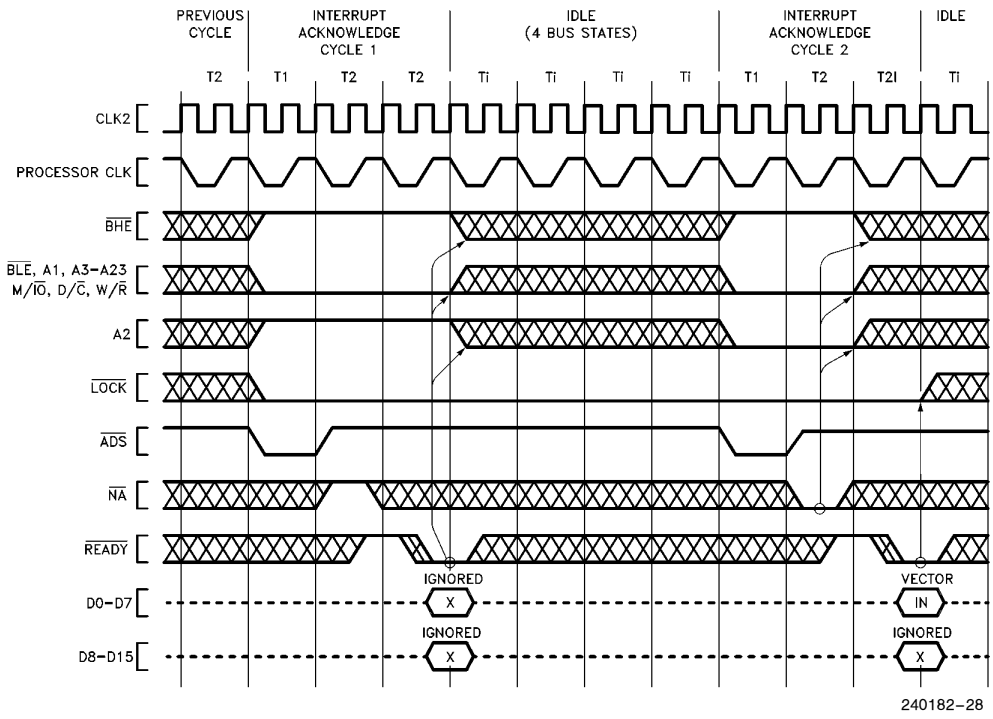
Realistically, pipelining is almost always maintained as long as $\overline{NA}$ is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore pipelining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

INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In repsonse to an interrupt request on the INTR input when interrupts are enabled, the 80376 performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by $\overline{READY}$ sampled active.

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

The $\overline{LOCK}$ output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 80376 between the two interrupt acknowledge cycles for compatibility with the interrupt specification T_{RHRL} of the 8259A Interrupt Controller and the 82370 Integrated Peripheral.



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 $\overline{NA}$ has no practical effect. Choose the approach which is simplest for your system hardware design.

Figure 4.13. Interrupt Acknowledge Cycles

During both interrupt acknowledge cycles, D_{15}–D_0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 80376 will read an external interrupt vector from D_7–D_0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.

HALT INDICATION CYCLE

The 80376 execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals and a byte address of 2. See the **Bus Cycle Definition Signals** section. The halt indication cycle must be acknowledged by $\overline{READY}$ asserted. A halted 80376 resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.

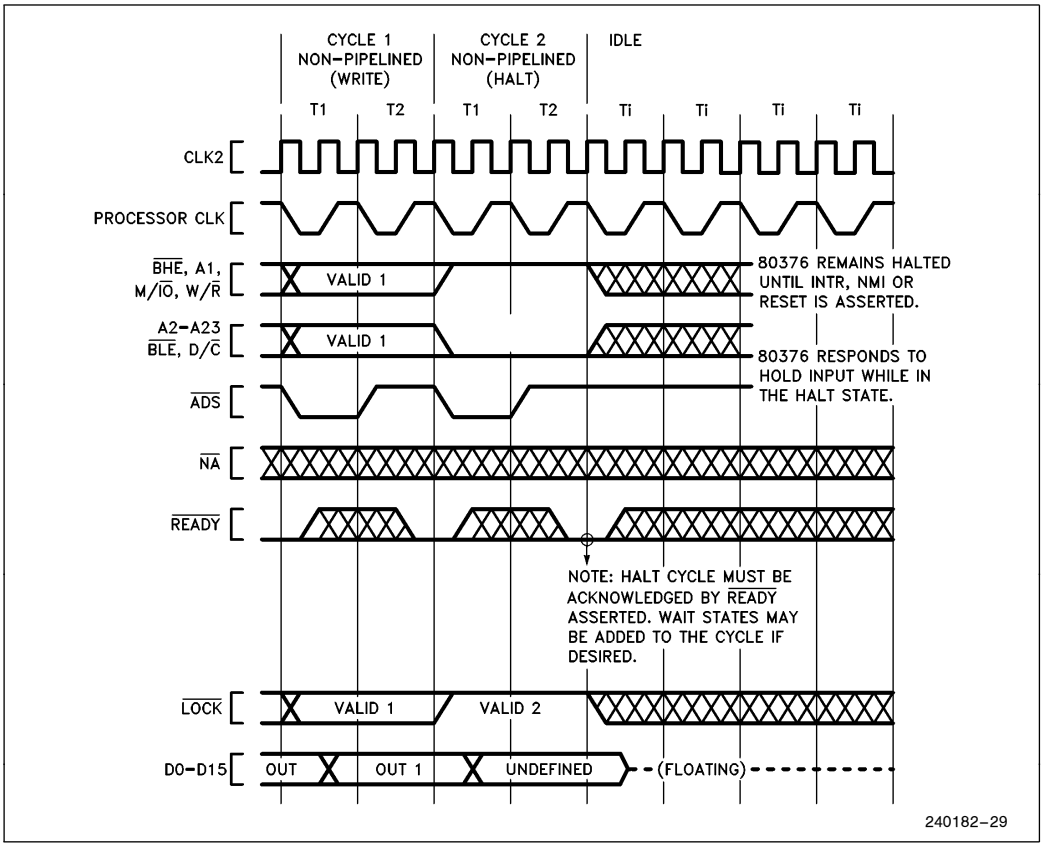


Figure 4.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 80376 shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **Bus Cycle Definition Signals** and a byte address of 0. The shutdown indication cycle must be acknowledged by $\overline{READY}$ asserted. A shutdown 80376 resumes execution when NMI or RESET is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 80376 floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the 80376 remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD and RESET are ignored.

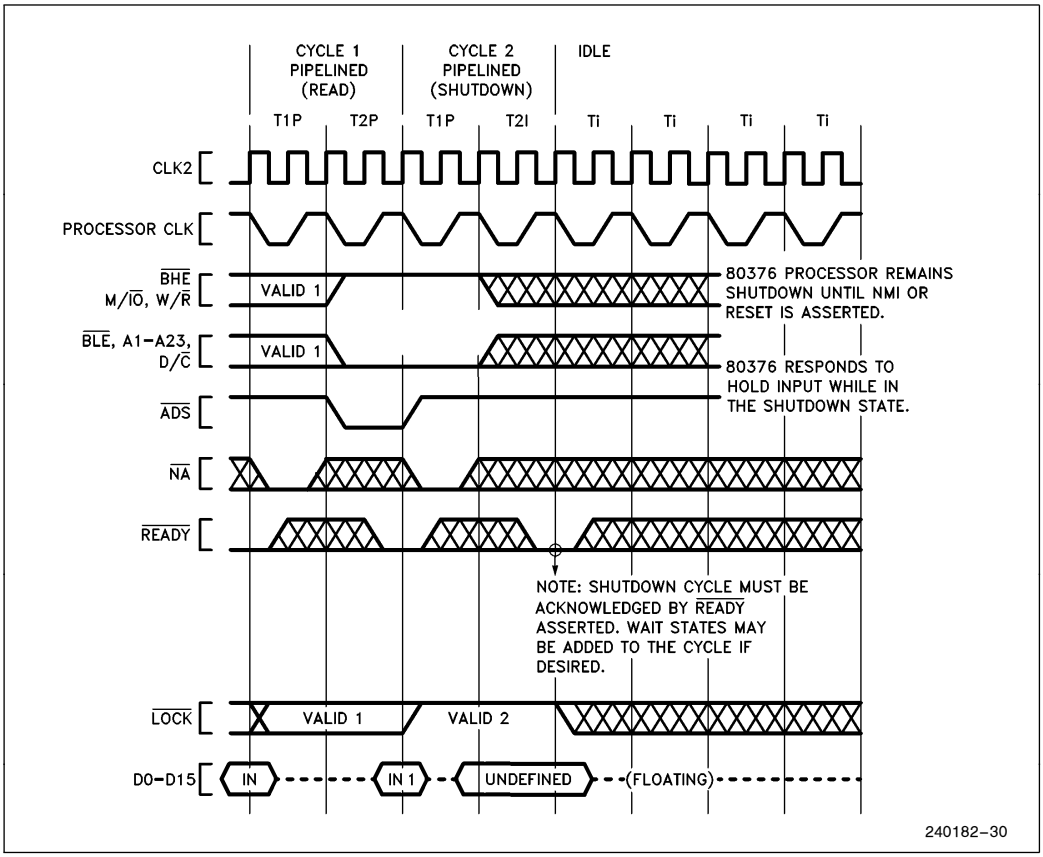
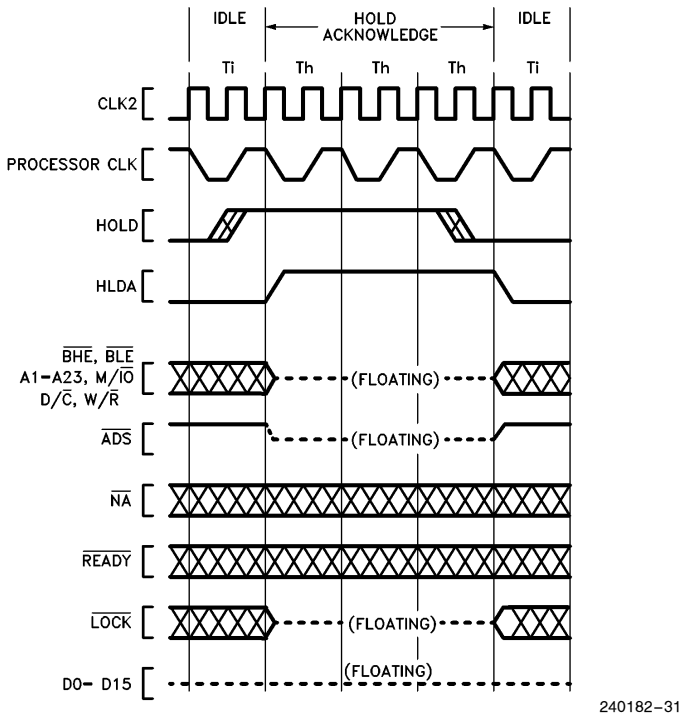


Figure 4.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle

T_h may be entered from a bus idle state as in Figure 4.16 or after the acknowledgement of the current physical bus cycle if the $\overline{LOCK}$ signal is not asserted, as in Figures 4.17 and 4.18.

T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 4.16 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 4.17 and 4.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h, the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 80376 is reset before T_h is exited.



NOTE:
For maximum design flexibility the 80376 has no internal pull-up resistors on its outputs. Your design may require an external pullup on ADS and other 80376 outputs to keep them negated during float periods.

Figure 4.16. Requesting Hold from Idle Bus

RESET DURING HOLD ACKNOWLEDGE

RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the 80376 drives its pins to defined states during reset, as in **Table 4.5, Pin State During Reset,** and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 80376 enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 80376 processor would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is inactive, the BUSY input is still sampled as usual to determine whether a self test is being requested.

FLOAT

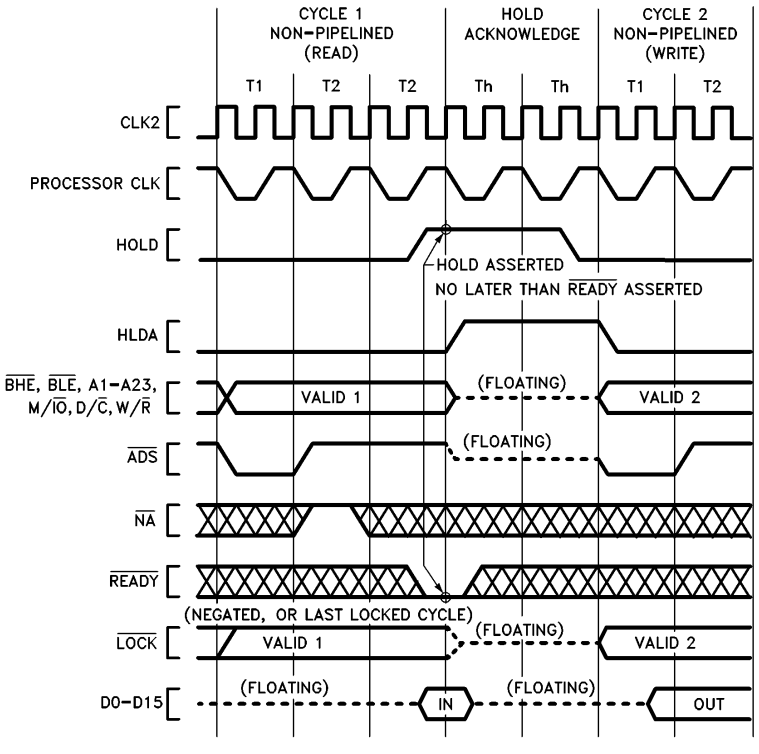
Activating the FLT input floats all 80376 bidirectional and output signals, including HLDA. Asserting FLT isolates the 80376 from the surrounding circuitry.

When an 80376 in a PQFP surface-mount package is used without a socket, it cannot be removed from the printed circuit board. The FLT input allows the 80376 to be electrically isolated to allow testing of external circuitry. This technique is known as ONCE for ''ON-Circuit Emulation''.

ENTERING AND EXITING FLOAT

FLT is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the 80376 (Figure 4.20). FLT must be held low for a minimum of 16 CLK2 cycles. Reset should be asserted and held asserted until after FLT is deasserted. This will ensure that the 80376 will exit float in a valid state.

Asserting the FLT input unconditionally aborts the current bus cycle and forces the 80376 into the FLOAT mode. Since activating FLT unconditionally forces the 80376 into FLOAT mode, the 80376 is not



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 4.17. Requesting Hold from Active Bus ($\overline{NA}$ Inactive)

guaranteed to enter FLOAT in a valid state. After deactivating $\overline{FLT}$, the 80376 is not guaranteed to exit FLOAT mode in a valid state. This is not a problem as the $\overline{FLT}$ pin is meant to be used only during ONCE. After exiting FLOAT, the 80376 must be reset to return it to a valid state. Reset should be asserted before $\overline{FLT}$ is deasserted. This will ensure that the 80376 will exit float in a valid state.

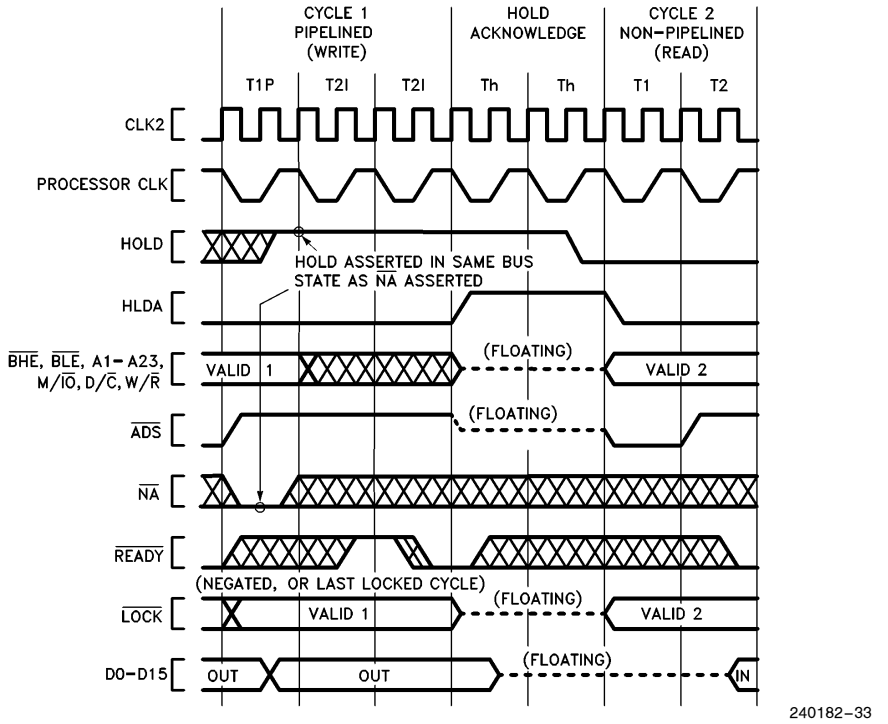
$\overline{FLT}$ has an internal pull-up resistor, and if it is not used it should be unconnected.

BUS ACTIVITY DURING AND FOLLOWING RESET

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

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 80376, and at least 80 CLK2 periods if a 80376 self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26}, the internal processor clock phase is defined at that time as illustrated by Figure 4.19 and Figure 6.7.

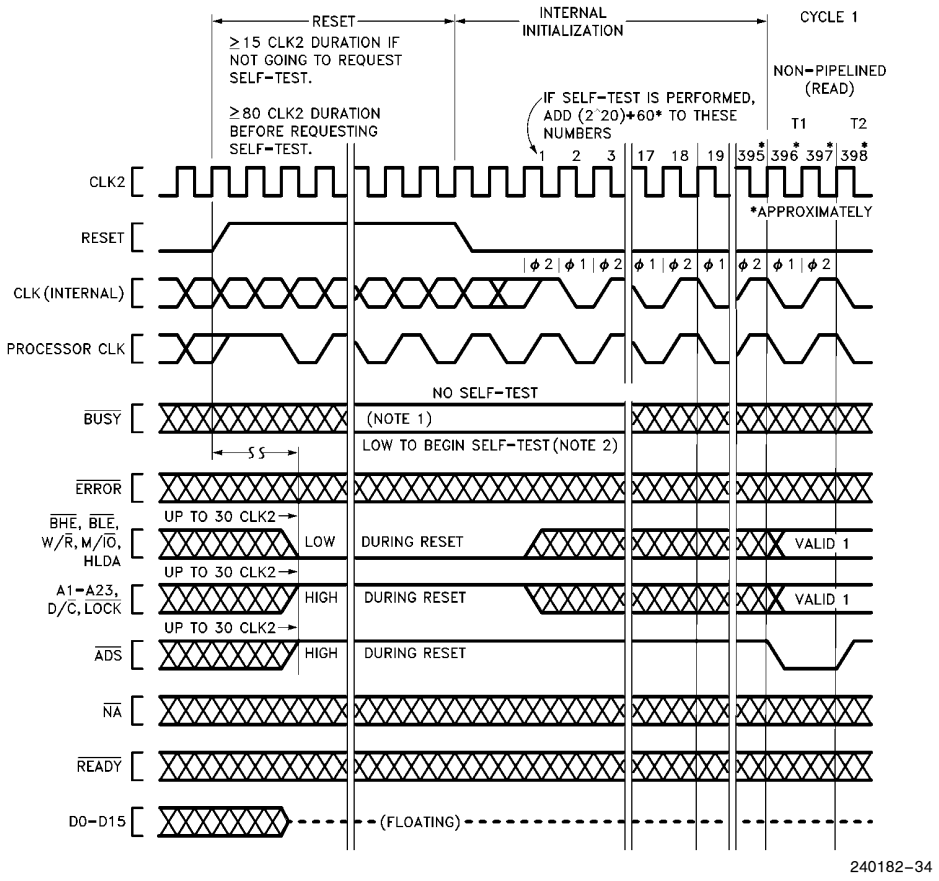


NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 4.18. Requesting Hold from Idle Bus ($\overline{NA}$ Active)

An 80376 self-test may be requested at the time RESET goes inactive by having the $\overline{BUSY}$ input at a LOW level as shown in Figure 4.19. The self-test requires (2^{20} + approximately 60) CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 80376 attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 80376 performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.



NOTES:
1. BUSY should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested, the 80376 outputs remain in their reset state as shown here.

Figure 4.19. Bus Activity from Reset until First Code Fetch

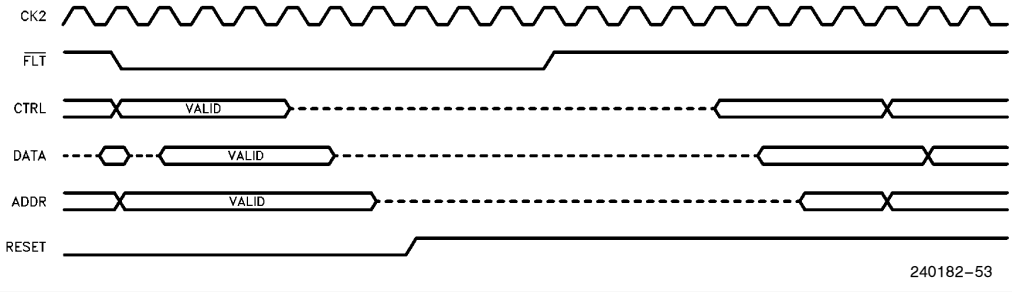


Figure 4.20. Entering and Exiting FLOAT

4.5 Self-Test Signature

Upon completion of self-test (if self-test was requested by driving BUSY LOW at the falling edge of RESET) the EAX register will contain a signature of 00000000H indicating the 80376 passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all 80376 revision levels. Any non-zero signature indicates the 80376 unit is faulty.

4.6 Component and Revision Identifiers

To assist 80376 users, the 80376 after reset holds a component identifier and revision identifier in its DX register. The upper 8 bits of DX hold 33H as identification of the 80376 component. (The lower nibble, 03H, refers to the Intel386™ architecture. The upper nibble, 30H, refers to the third member of the Intel386 family). The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 80376 revision identifier will track that of the 80386 where possible.

The revision identifier is intended to assist 80376 users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 4.7. Component and Revision Identifier History

80376 Stepping Name	Revision Identifier
A0	05H
B	08H

4.7 Coprocessor Interfacing

The 80376 provides an automatic interface for the Intel 80387SX numeric floating-point coprocessor. The 80387SX coprocessor uses an I/O mapped interface driven automatically by the 80376 and assisted by three dedicated signals: BUSY, ERROR and PEREQ.

As the 80376 begins supporting a coprocessor instruction, it tests the BUSY and ERROR signals to determine if the coprocessor can accept its next instruction. Thus, the BUSY and ERROR inputs eliminate the need for any ''preamble'' bus cycles for communication between processor and coprocessor. The 80387SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 80376 WAIT opcode (9BH) for 80387SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 80376 based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol ''primitives''. Instead, memory-mapped or I/O-mapped interfaces may use all applicable 80376 instructions for high-speed coprocessor communication. The BUSY and ERROR inputs of the 80376 may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 80376 WAIT opcode (9BH). The WAIT instruction will wait until the BUSY input is inactive (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the ERROR pin is active when the BUSY goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the segmentation mechanism of the 80376. If the custom interface is I/O-mapped, protection of the interface can be provided with the 80376 IOPL (I/O Privilege Level) mechanism.

The 80387SX numeric coprocessor interface is I/O mapped as shown in Table 4.8. Note that the 80387SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 80376 supports the 80387SX coprocessor, the 80376 automatically generates bus cycles to the coprocessor interface addresses.

Table 4.8 Numeric Coprocessor Port Addresses

Address in 80376 I/O Space	80387SX Coprocessor Register
8000F8H	Opcode Register
8000FCH	Operand Register
8000FEH	Operand Register

SOFTWARE TESTING FOR COPROCESSOR PRESENCE

When software is used to test coprocessor (80387SX) presence, it should use only the following coprocessor opcodes: FNINIT, FNSTCW and FNSTSW. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 80376 CR0 register.

5.0 PACKAGE THERMAL SPECIFICATIONS

The Intel 80376 embedded processor is specified for operation when case temperature is within the range of 0°C–115°C for both the ceramic 88-pin PGA package and the plastic 100-pin PQFP package. The case temperature may be measured in any environment, to determine whether the 80376 is within specified operating range. The case temperature should be measured at the center of the top surface.

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

$$T_J = T_c + P^*\theta_{jc}$$

$$T_A = T_j - P^*\theta_{ja}$$

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

Values for θ_{ja} and θ_{jc} are given in Table 5.1 for the 100-lead fine pitch. θ_{ja} is given at various airflows. Table 5.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching ''fins'' or a ''heat sink'' to the package. P is calculated using the maximum ***cold*** I_{cc} of 305 mA and the maximum V_{CC} of 5.5V for both packages.

Table 5.1. 80376 Package Thermal Characteristics Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

Package	θ_{jc}	θ_{ja} Versus Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7.5	34.5	29.5	25.5	22.5	21.5	21.0
88-Pin PGA	2.5	29.0	22.5	17.0	14.5	12.5	12.0

Table 5.2. 80376 Maximum Allowable Ambient Temperature at Various Airflows

Package	θ_{jc}	T_A(°C) vs Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7.5	70	78	85	90	92	93
88-Pin PGA	2.5	70	81	90	95	98	99

6.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 80376, and its electrical specifications.

6.1 Power and Grounding

The 80376 is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 V_{CC} and 18 V_{SS} pins separately feed functional units of the 80376.

Power and ground connections must be made to all external V_{CC} and GND pins of the 80376. On the circuit board, all V_{CC} pins should be connected on a V_{CC} plane and all V_{SS} pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 80376. The 80376 driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 80376 and decoupling capacitors as much as possible.

RESISTOR RECOMMENDATIONS

The $\overline{\text{ERROR}}$, $\overline{\text{FLT}}$ and $\overline{\text{BUSY}}$ inputs have internal pull-up resistors of approximately 20 KΩ and the PEREQ input has an internal pull-down resistor of approximately 20 KΩ built into the 80376 to keep these signals inactive when the 80387SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 6.1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

Table 6.1. Recommended Resistor Pull-Ups to V_{CC}

Pin	Signal	Pull-Up Value	Purpose
16	$\overline{ADS}$	20 KΩ ± 10%	Lightly Pull $\overline{ADS}$ Inactive during 80376 Hold Acknowledge States
26	$\overline{LOCK}$	20 KΩ ± 10%	Lightly Pull $\overline{LOCK}$ Inactive during 80376 Hold Acknowledge States

OTHER CONNECTION RECOMMENDATIONS

For reliable operation, always connect unused inputs to an appropriate signal level. N/C pins should always remain **unconnected. Connection of N/C pins to V_{CC} or V_{SS} will result in incompatibility with future steppings of the 80376.**

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

—INTR
—NMI
—HOLD

If not using address pipelining connect the $\overline{NA}$ pin to a pull-up resistor in the range of 20 KΩ to V_{CC}.

6.2 Absolute Maximum Ratings

Table 6.2. Maximum Ratings

Parameter	Maximum Rating
Storage Temperature	−65°C to +150°C
Case Temperature under Bias	−65°C to +120°C
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V
Voltage on Other Pins	−0.5V to (V_{CC} + 0.5)V

Table 6.2 gives a stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **Section 6.3, D.C. Specifications,** and **Section 6.4, A.C. Specifications.**

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 80376 contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

6.3 D.C. Specifications

ADVANCE INFORMATION SUBJECT TO CHANGE

Table 6.3: 80376 D.C. Characteristics

Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

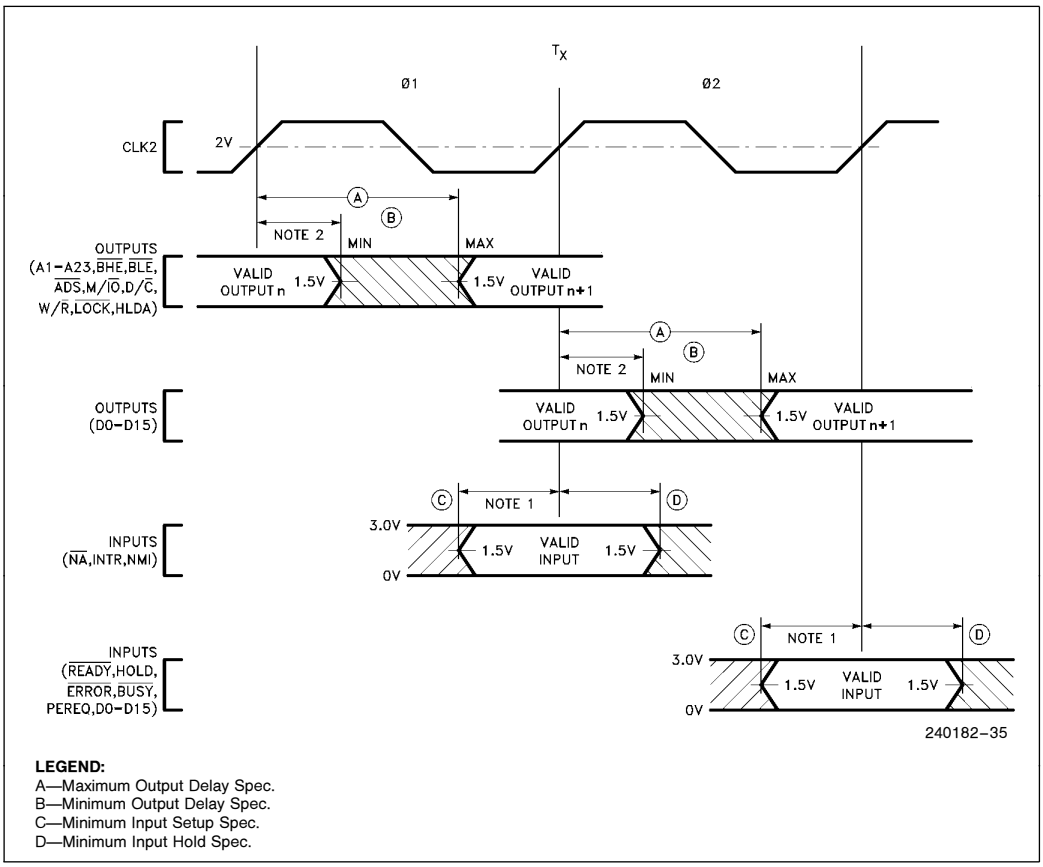
Symbol	Parameter	Min	Max	Unit
V_{IL}	Input LOW Voltage	−0.3	+0.8	V(1)
V_{IH}	Input HIGH Voltage	2.0	V_{CC} +0.3	V(1)
V_{ILC}	CLK2 Input LOW Voltage	−0.3	+0.8	V(1)
V_{IHC}	CLK2 Input HIGH Voltage	V_{CC} − 0.8	V_{CC} + 0.3	V(1)
V_{OL}	Output LOW Voltage			
I_{OL} = 4 mA:	A_{23}–A_1, D_{15}–D_0		0.45	V(1)
I_{OL} = 5 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA		0.45	V(1)
V_{OH}	Output High Voltage			
I_{OH} = −1 mA:	A_{23}–A_1, D_{15}–D_0	2.4		V(1)
I_{OH} = −0.2 mA:	A_{23}–A_1, D_{15}–D_0	V_{CC} − 0.5		V(1)
I_{OH} = −0.9 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA	2.4		V(1)
I_{OH} = −0.18 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$ $\overline{ADS}$, HLDA	V_{CC} − 0.5		V(1)
I_{LI}	Input Leakage Current (For All Pins except PEREQ, $\overline{BUSY}$, $\overline{FLT}$ and $\overline{ERROR}$)		±15	μA, 0V ≤ V_{IN} ≤ V_{CC}(1)
I_{IH}	Input Leakage Current (PEREQ Pin)		200	μA, V_{IH} = 2.4V(1, 2)
I_{IL}	Input Leakage Current ($\overline{BUSY}$ and $\overline{ERROR}$ Pins)		−400	μA, V_{IL} = 0.45V(3)
I_{LO}	Output Leakage Current		±15	μA, 0.45V ≤ V_{OUT} ≤ V_{CC}(1)
I_{CC}	Supply Current CLK2 = 32 MHz CLK2 = 40 MHz		 275 305	 mA, I_{CC} typ = 175 mA(4) mA, I_{CC} typ = 200 mA(4)
C_{IN}	Input Capacitance		10	pF, F_C = 1 MHz(5)
C_{OUT}	Output or I/O Capacitance		12	pF, F_C = 1 MHz(5)
C_{CLK}	CLK2 Capacitance		20	pF, F_C = 1 MHz(5)

NOTES:
1. Tested at the minimum operating frequency of the device.
2. PEREQ input has an internal pull-down resistor.
3. $\overline{BUSY}$, $\overline{FLT}$ and $\overline{ERROR}$ inputs each have an internal pull-up resistor.
4. I_{CC} max measurement at worse case load, V_{CC} and temperature (0°C).
5. Not 100% tested.

The A.C. specifications given in Table 6.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. specification measurement is defined by Figure 6.1. Inputs must be driven to the voltage levels indicated by Figure 6.1 when A.C. specifications are measured. 80376 output delays are specified with minimum and maximum limits measured as shown. The minimum 80376 delay times are hold times provided to external circuitry. 80376 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 80376 processor operation.

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



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

Figure 6.1. Drive Levels and Measurement Points for A.C. Specifications

6.4 A.C. Specifications

Table 6.4. 80376 A.C. Characteristics at 16 MHz

Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	16	MHz		Half CLK2 Freq
t_1	CLK2 Period	31	125	ns	6.3	
t_{2a}	CLK2 HIGH Time	9		ns	6.3	At 2(3)
t_{2b}	CLK2 HIGH Time	5		ns	6.3	At (V_{CC} − 0.8)V(3)
t_{3a}	CLK2 LOW Time	9		ns	6.3	At 2V(3)
t_{3b}	CLK2 LOW Time	7		ns	6.3	At 0.8V(3)
t_4	CLK2 Fall Time		8	ns	6.3	(V_{CC}−0.8)V to 0.8V(3)
t_5	CLK2 Rise Time		8	ns	6.3	0.8V to (V_{CC}−0.8)(3)
t_6	A_{23}–A_1 Valid Delay	4	36	ns	6.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4	40	ns	6.6	(1)
t_8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	4	36	ns	6.5	C_L = 75 pF(4)
t_9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	4	40	ns	6.6	(1)
t_{10}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Valid Delay	6	33	ns	6.5	C_L = 75 pF(4)
t_{11}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	6	35	ns	6.6	(1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	40	ns	6.5	C_L = 120 pF(4)
t_{13}	D_{15}–D_0 Write Data Float Delay	4	35	ns	6.6	(1)
t_{14}	HLDA Valid Delay	4	33	ns	6.6	C_L = 75 pF(4)
t_{15}	$\overline{NA}$ Setup Time	5		ns	6.4	
t_{16}	$\overline{NA}$ Hold Time	21		ns	6.6	
t_{19}	$\overline{READY}$ Setup Time	19		ns	6.4	
t_{20}	$\overline{READY}$ Hold Time	4		ns	6.4	
t_{21}	Setup Time D_{15}–D_0 Read Data	9		ns	6.4	
t_{22}	Hold Time D_{15}–D_0 Read Data	6		ns	6.4	
t_{23}	HOLD Setup Time	26		ns	6.4	
t_{24}	HOLD Hold Time	5		ns	6.4	
t_{25}	RESET Setup Time	13		ns	6.7	
t_{26}	RESET Hold Time	4		ns	6.7	

Table 6.4. 80376 A.C. Characteristics at 16 MHz (Continued)
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{27}	NMI, INTR Setup Time	16		ns	6.4	(2)
t_{28}	NMI, INTR Hold Time	16		ns	6.4	(2)
t_{29}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup Time	16		ns	6.4	(2)
t_{30}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold Time	5		ns	6.4	(2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set to 50 pF and derated to support the indicated distributed capacitive load. See Figures 6.8 through 6.10 for capacitive derating curves.
5. The 80376 does not have t_{17} or t_{18} timing specifications.

Table 6.5. 80376 A.C. Characteristics at 20 MHz
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	20	MHz		Half CLK2 Frequency
t_1	CLK2 Period	25	125	ns	6.3	
t_{2a}	CLK2 HIGH Time	8		ns	6.3	At 2V(3)
t_{2b}	CLK2 HIGH Time	5		ns	6.3	At (V_{CC} − 0.8)V(3)
t_{3a}	CLK2 LOW Time	8		ns	6.3	At 2V(3)
t_{3b}	CLK2 LOW Time	6		ns	6.3	At 0.8V(3)
t_4	CLK2 Fall Time		8	ns	6.3	(V_{CC}−0.8V) to 0.8V(3)
t_5	CLK2 Rise Time		8	ns	6.3	0.8V to (V_{CC}−0.8)(3)
t_6	A_{23}–A_1 Valid Delay	4	30	ns	6.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4		ns	6.6	(1)
t_8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	4	30	ns	6.5	C_L = 75 pF(4)
t_9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	4	32	ns	6.6	(1)
t_{10a}	M/$\overline{IO}$, D/$\overline{C}$ Valid Delay	6	28	ns	6.5	C_L = 75 pF(4)
t_{10b}	W/$\overline{R}$, $\overline{ADS}$ Valid Delay	6	26	ns	6.5	C_L = 75 pF(4)
t_{11}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	6	30	ns	6.6	(1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	38	ns	6.5	C_L = 120 pF
t_{13}	D_{15}–D_0 Write Data Float Delay	4	27	ns	6.6	(1)

Table 6.5. 80376 A.C. Characteristics at 20 MHz (Continued)
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{14}	HLDA Valid Delay	4	28	ns	6.5	C_L = 75 pF(4)
t_{15}	$\overline{NA}$ Setup Time	5		ns	6.4	
t_{16}	$\overline{NA}$ Hold Time	12		ns	6.4	
t_{19}	$\overline{READY}$ Setup Time	12		ns	6.4	
t_{20}	$\overline{READY}$ Hold Time	4		ns	6.4	
t_{21}	D_{15}–D_0 Read Data Setup Time	9		ns	6.4	
t_{22}	D_{15}–D_0 Read Data Hold Time	6		ns	6.4	
t_{23}	HOLD Setup Time	17		ns	6.4	
t_{24}	HOLD Hold Time	5		ns	6.4	
t_{25}	RESET Setup Time	12		ns	6.7	
t_{26}	RESET Hold Time	4		ns	6.7	
t_{27}	NMI, INTR Setup Time	16		ns	6.4	(2)
t_{28}	NMI, INTR Hold Time	16		ns	6.4	(2)
t_{29}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup Time	14		ns	6.4	(2)
t_{30}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold Time	5		ns	6.4	(2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set to 50 pF and derated to support the indicated distributed capacitive load. See Figures 6.8 through 6.10 for capacitive derating curves.
5. The 80376 does not have t_{17} or t_{18} timing specifications.

A.C. TEST LOADS

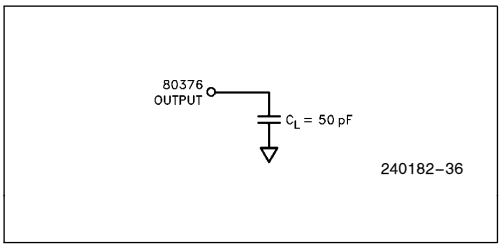


Figure 6.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

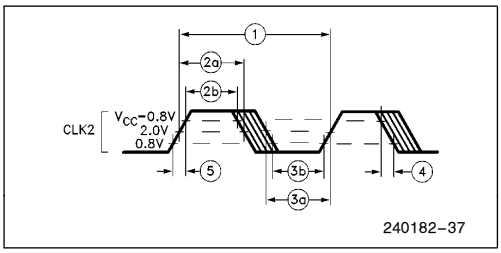


Figure 6.3. CLK2 Waveform

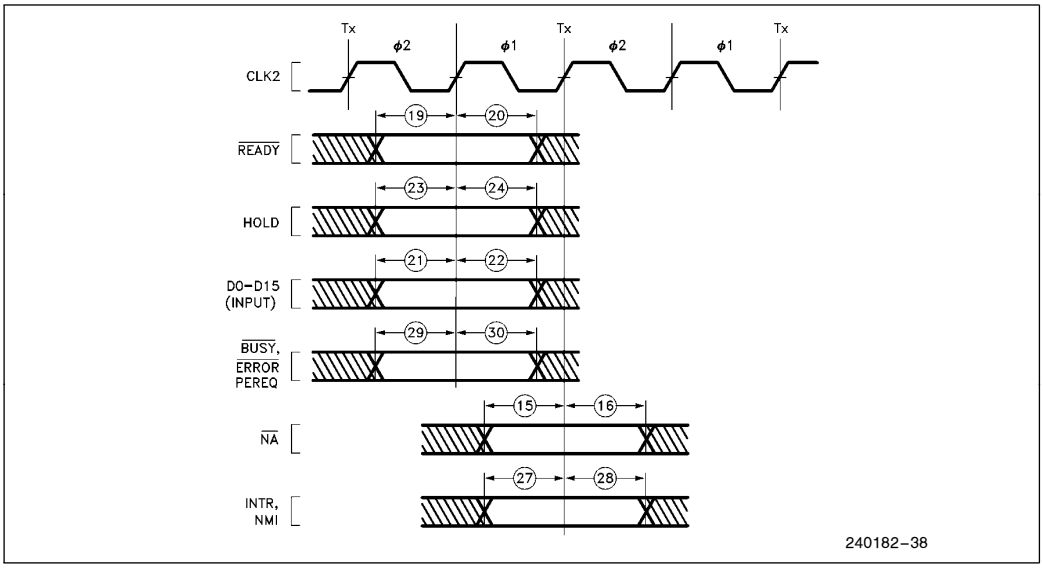


Figure 6.4. A.C. Timing Waveforms—Input Setup and Hold Timing

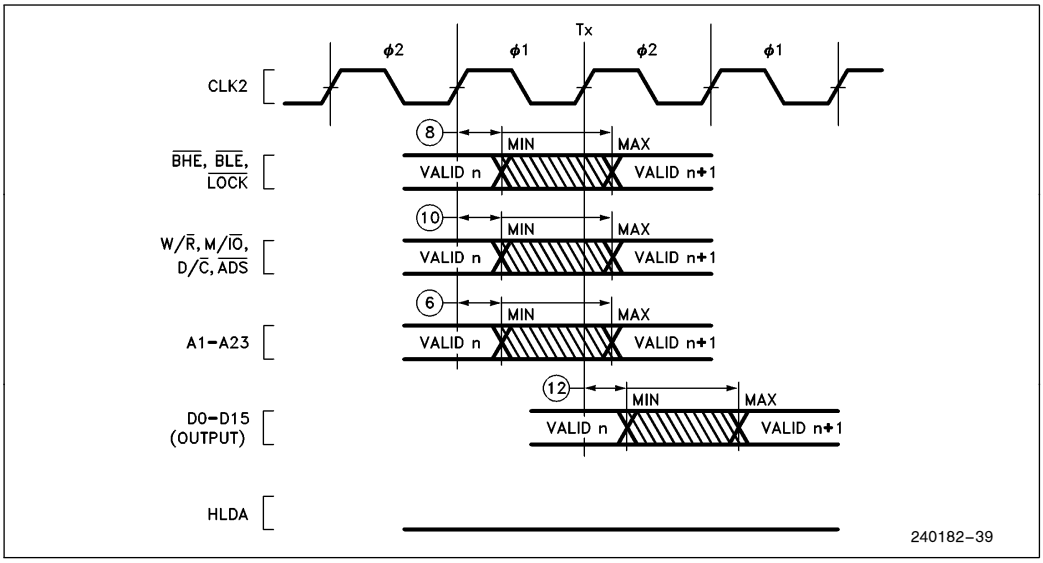


Figure 6.5. A.C. Timing Waveforms—Output Valid Delay Timing

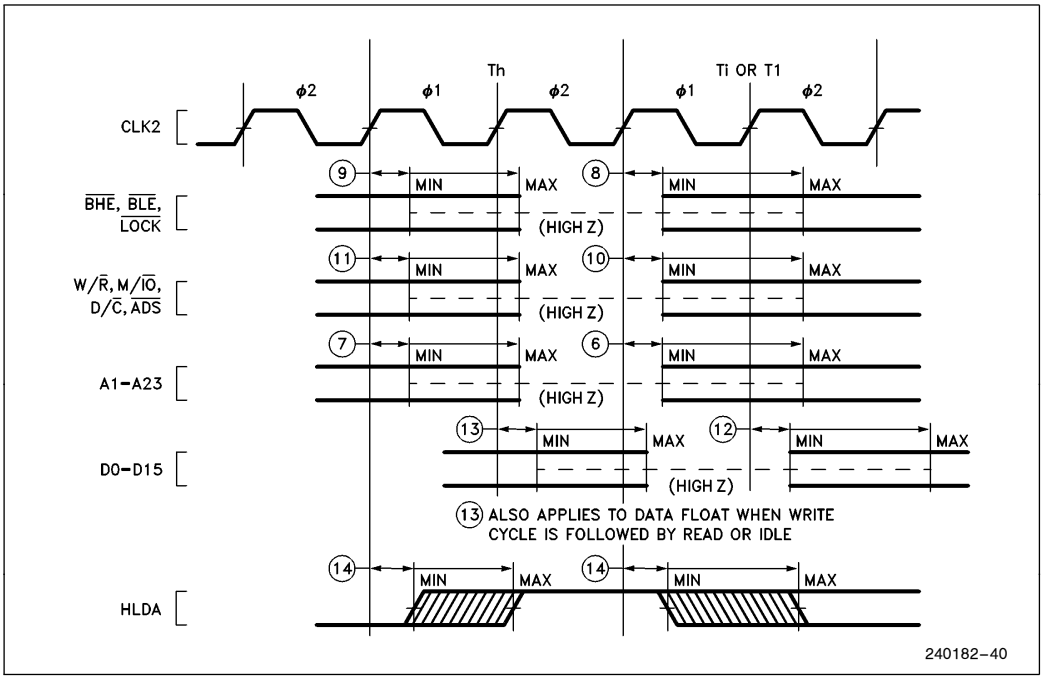
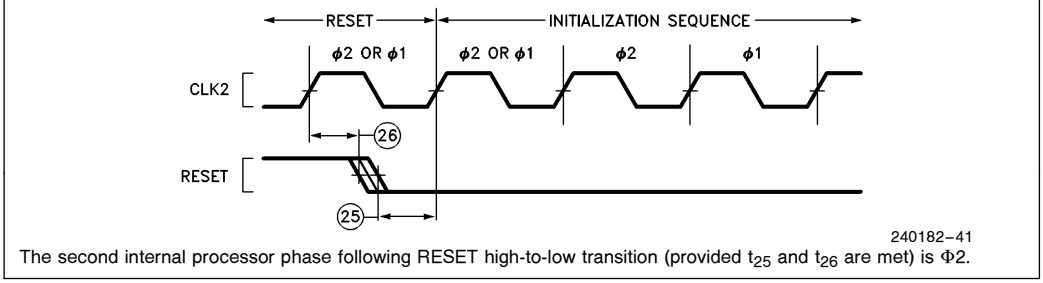


Figure 6.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing



The second internal processor phase following RESET high-to-low transition (provided t_{25} and t_{26} are met) is Φ2.

Figure 6.7. A.C. Timing Waveforms—RESET Setup and Hold Timing, and Internal Phase

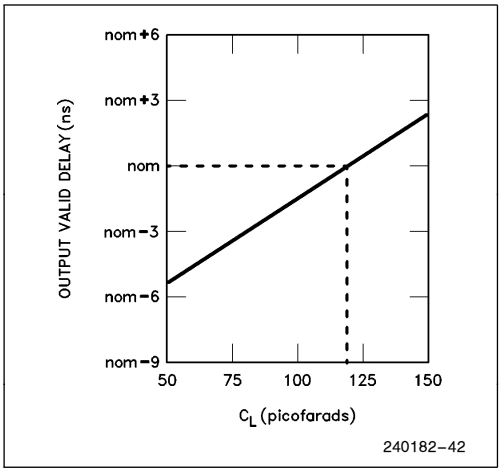


Figure 6.8. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 120 pF)

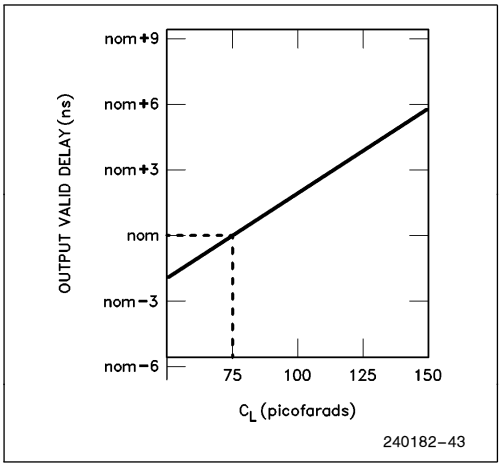


Figure 6.9. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 75 pF)

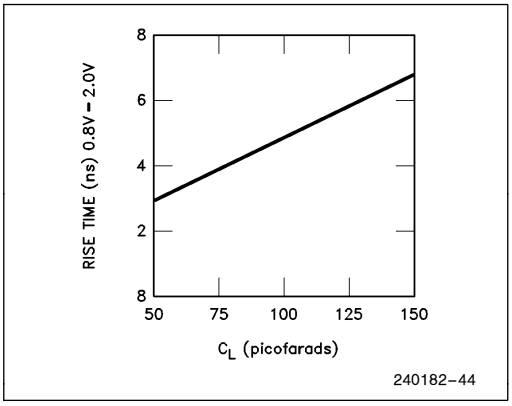


Figure 6.10. Typical Output Rise Time versus Load Capacitance at Maximum Operating Temperature

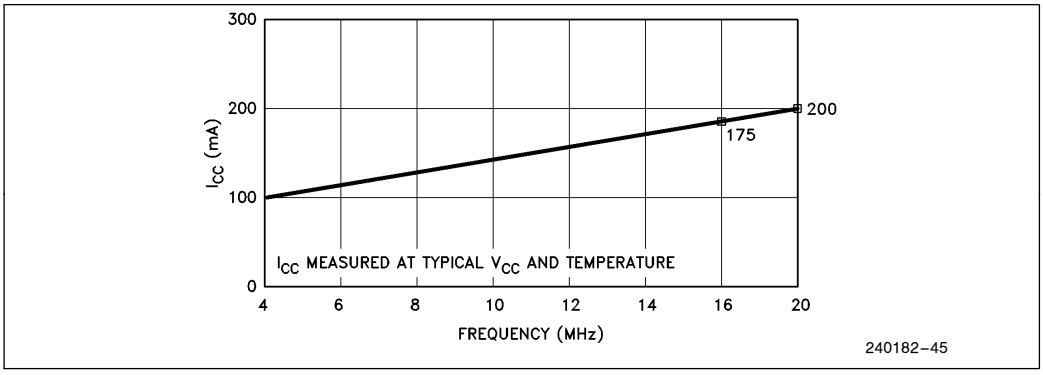


Figure 6.11. Typical I_{CC} vs Frequency

6.5 Designing for the ICE™-376 Emulator

The 376 embedded processor in-circuit emulator product is the ICE-376 emulator. Use of the emulator requires the target system to provide a socket that is compatible with the ICE-376 emulator. The 80376 offers two different probes for emulating user systems: an 88-pin PGA probe and a 100-pin fine pitch flat-pack probe. The 100-pin fine pitch flat-pack probe requires a socket, called the 100-pin PQFP, which is available from 3-M Textool (part number 2-0100-07243-000). The ICE-376 emulator probe attaches to the target system via an adapter which replaces the 80376 component in the target system. Because of the high operating frequency of 80376 systems and of the ICE-376 emulator, there is no buffering between the 80376 emulation processor in the ICE-376 emulator probe and the target system. A direct result of the non-buffered interconnect is that the ICE-376 emulator shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE emulator hardware. In order for the ICE-376 emulator to be functional in the user's system without the Optional Isolation Board (OIB) the designer must be aware of the following conditions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 80376, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an 80376 would, and responds to RESET later. Correct phase of the response is guaranteed.

In addition to the above considerations, the ICE-376 emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 80376 system.

Capacitive Loading: ICE-376 adds up to 27 pF to each 80376 signal.

Drive Requirements: ICE-376 adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 80376 emulation processor, which has standard drive and loading capability listed in Tables 6.3 and 6.4.

Power Requirements: For noise immunity and CMOS latch-up protection the ICE-376 emulator processor module is powered by the user system. The circuitry on the processor module draws up to 1.4A including the maximum 80376 I_{CC} from the user 80376 socket.

80376 Location and Orientation: The ICE-376 emulator processor module may require lateral clearance. Figure 6.12 shows the clearance requirements of the iMP adapter and Figure 6.13 shows the clearance requirements of the 88-pin PGA adapter. The

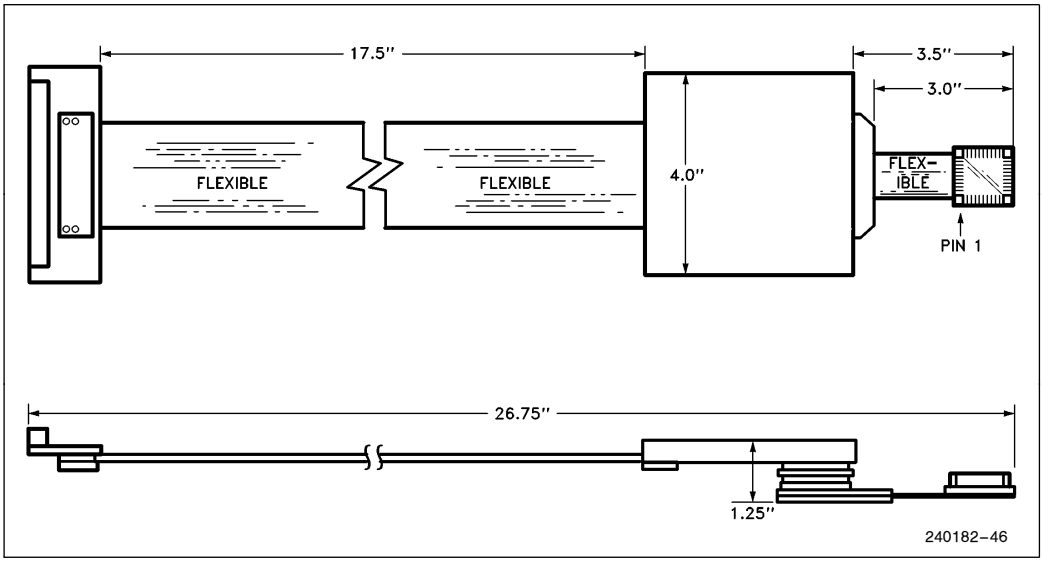


Figure 6.12. Preliminary ICE™-376 Emulator User Cable with PQFP Adapter

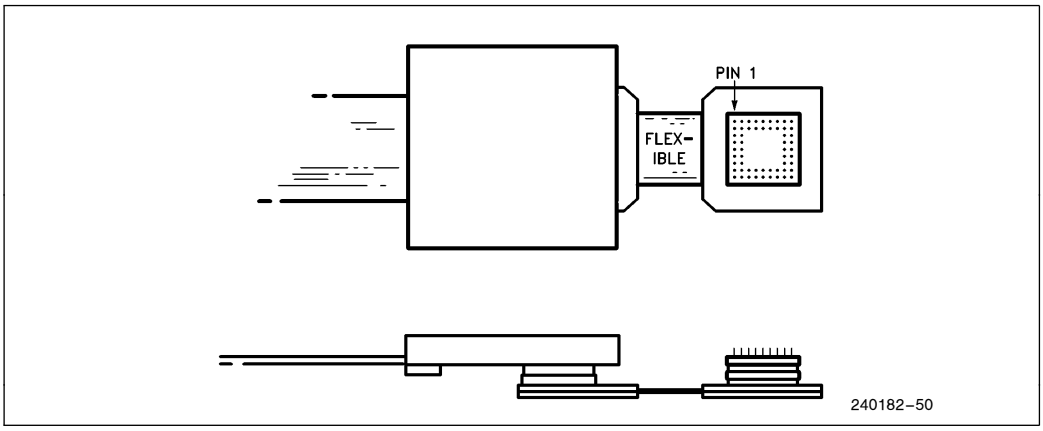


Figure 6.13. ICE™-376 Emulator User Cable with 88-Pin PGA Adapter

optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figures 6.12 and 6.13, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 6.14.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-376 emulator is susceptible to errors on the user's bus. The OIB allows the ICE-376 emulator to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 20 MHz.

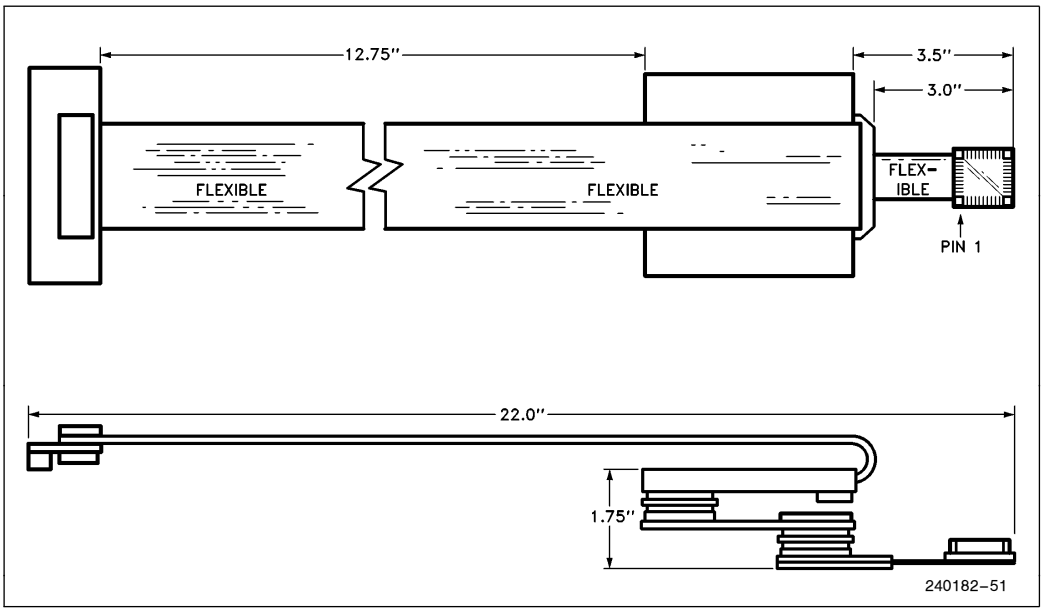


Figure 6.14. ICE™-376 Emulator User Cable with OIB and PQFP Adapter

7.0 DIFFERENCES BETWEEN THE 80376 AND THE 80386

The following are the major differences between the 80376 and the 80386.

1. The 80376 generates byte selects on $\overline{BHE}$ and $\overline{BLE}$ (like the 8086 and 80286 microprocessors) to distinguish the upper and lower bytes on its 16-bit data bus. The 80386 uses four-byte selects, $\overline{BE0}$–$\overline{BE3}$, to distinguish between the different bytes on its 32-bit bus.
2. The 80376 has no bus sizing option. The 80386 can select between either a 32-bit bus or a 16-bit bus by use of the $\overline{BS16}$ input. The 80376 has a 16-bit bus size.
3. The $\overline{NA}$ pin operation in the 80376 is identical to that of the $\overline{NA}$ pin on the 80386 with one exception: the $\overline{NA}$ pin of the 80386 cannot be activated on 16-bit bus cycles (where $\overline{BS16}$ is LOW in the 80386 case), whereas $\overline{NA}$ can be activated on any 80376 bus cycle.
4. The contents of all 80376 registers at reset are identical to the contents of the 80386 registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

   in 80386, after reset DH = 03H indicates 80386
   DL = revision number;
   in 80376, after reset DH = 33H indicates 80376
   DL = revision number.

5. The 80386 uses A_{31} and M/$\overline{IO}$ as a select for numerics coprocessor. The 80376 uses the A_{23} and M/$\overline{IO}$ to select its numerics coprocessor.
6. The 80386 prefetch unit fetches code in four-byte units. The 80376 prefetch unit reads two bytes as one unit (like the 80286 microprocessor). In $\overline{BS16}$ mode, the 80386 takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 80386 will fetch all four bytes before addressing the new request.
7. The 80376 has no paging mechanism.
8. The 80376 starts executing code in what corresponds to the 80386 protected mode. The 80386 starts execution in real mode, which is then used to enter protected mode.
9. The 80386 has a virtual-86 mode that allows the execution of a real mode 8086 program as a task in protected mode. The 80376 has no virtual-86 mode.
10. The 80386 maps a 48-bit logical address into a 32-bit physical address by segmentation and paging. The 80376 maps its 48-bit logical address into a 24-bit physical address by segmentation only.
11. The 80376 uses the 80387SX numerics coprocessor for floating point operations, while the 80386 uses the 80387 coprocessor.
12. The 80386 can execute from 16-bit code segments. The 80376 can **only** execute from 32-bit code Segments.
13. The 80376 has an input called $\overline{FLT}$ which three-states all bidirectional and output pins, including HLDA, when asserted. It is used with ON Circuit Emulation (ONCE).

8.0 INSTRUCTION SET

This section describes the 376 embedded processor instruction set. Table 8.1 lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 80376 instructions.

8.1 80376 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. 50 ns for an 80376 operating at 20 MHz). The actual clock count of an 80376 program will average 10% more

than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions:

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor acess to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts showns. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.
6. Memory reference instruction accesses byte or aligned 16-bit operands.

Instruction Clock Count Notation

— If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.

—n = number of times repeated.

—m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses:

— If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:

2* clocks for read or write.

4** clocks for read and write.

— If instructions accesses a 32-bit operand on odd address add:

4* clocks for read or write.

8** clocks for read and write.

Wait States:

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

Table 8.1. 80376 Instruction Set Clock Count Summary

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
GENERAL DATA TRANSFER				
MOV = Move:				
Register to Register/Memory	1 0 0 0 1 0 0 w | mod reg r/m	2/2*	0/1*	a
Register/Memory to Register	1 0 0 0 1 0 1 w | mod reg r/m	2/4*	0/1*	a
Immediate to Register/Memory	1 1 0 0 0 1 1 w | mod 0 0 0 r/m | immediate data	2/2*	0/1*	a
Immediate to Register (Short Form)	1 0 1 1 w reg | immediate data	2	2	
Memory to Accumulator (Short Form)	1 0 1 0 0 0 0 w | full displacement	4*	1*	a
Accumulator to Memory (Short Form)	1 0 1 0 0 0 1 w | full displacement	2*	1*	a
Register/Memory to Segment Register	1 0 0 0 1 1 1 0 | mod sreg3 r/m	22/23*	0/6*	a,b,c
Segment Register to Register/Memory	1 0 0 0 1 1 0 0 | mod sreg3 r/m	2/2*	0/1*	a
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*	0/1*	a
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*	0/1*	a
PUSH = Push:				
Register/Memory	1 1 1 1 1 1 1 1 | mod 1 1 0 r/m	7/9*	2/4*	a
Register (Short Form)	0 1 0 1 0 reg	4	2	a
Segment Register (ES, CS, SS or DS)	0 0 0 sreg2 1 1 0	4	2	a
Segment Register (FS or GS)	0 0 0 0 1 1 1 1 | 1 0 sreg3 0 0 0	4	2	a
Immediate	0 1 1 0 1 0 s 0 | immediate data	4	2	a
PUSHA = Push All	0 1 1 0 0 0 0 0	34	16	a
POP = Pop				
Register/Memory	1 0 0 0 1 1 1 1 | mod 0 0 0 r/m	7/9*	2/4*	a
Register (Short Form)	0 1 0 1 1 reg	6	2	a
Segment Register (ES, SS or DS)	0 0 0 sreg 2 1 1 1	25	6	a, b, c
Segment Register (FS or GS)	0 0 0 0 1 1 1 1 | 1 0 sreg 3 0 0 1	25	6	a, b, c
POPA = Pop All	0 1 1 0 0 0 0 1	40	16	a
XCHG = Exchange				
Register/Memory with Register	1 0 0 0 0 1 1 w | mod reg r/m	3/5**	0/2**	a, m
Register with Accumulator (Short Form)	1 0 0 1 0 reg	3	0	
IN = Input from:				
Fixed Port	1 1 1 0 0 1 0 w | port number	6*	1*	f,k
		26*	1*	f,l
Variable Port	1 1 1 0 1 1 0 w	7*	1*	f,k
		27*	1*	f,l
OUT = Output to:				
Fixed Port	1 1 1 0 0 1 1 w | port number	4*	1*	f,k
		24*	1*	f,l
Variable Port	1 1 1 0 1 1 1 w	5*	1*	f,k
		26*	1*	f,l
LEA = Load EA to Register	1 0 0 0 1 1 0 1 | mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
SEGMENT CONTROL				
LDS = Load Pointer to DS	11000101 | mod reg r/m	26*	6*	a, b, c
LES = Load Pointer to ES	11000100 | mod reg r/m	26*	6*	a, b, c
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	29*	6*	a, b, c
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	29*	6*	a, b, c
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	26*	6*	a, b, c
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
CLD = Clear Direction Flag	11111100	2		
CLI = Clear Interrupt Enable Flag	11111010	8		f
CLTS = Clear Task Switched Flag	00001111 | 00000110	5		e
CMC = Complement Carry Flag	11110101	2		
LAHF = Load AH into Flag	10011111	2		
POPF = Pop Flags	10011101	7		a, g
PUSHF = Push Flags	10011100	4		a
SAHF = Store AH into Flags	10011110	3		
STC = Set Carry Flag	11111001	2		
STD = Set Direction Flag	11111101	2		
STI = Set Interrupt Enable Flag	11111011	8		f
ARITHMETIC				
ADD = Add				
Register to Register	000000dw | mod reg r/m	2		
Register to Memory	0000000w | mod reg r/m	7**	2**	a
Memory to Register	0000001w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	100000sw | mod 000 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0000010w | immediate data	2		
ADC = Add with Carry				
Register to Register	000100dw | mod reg r/m	2		
Register to Memory	0001000w | mod reg r/m	7**	2**	a
Memory to Register	0001001w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	100000sw | mod 010 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0001010w | immediate data	2		
INC = Increment				
Register/Memory	1111111w | mod 000 r/m	2/6**	0/2**	a
Register (Short Form)	01000 reg	2		
SUB = Subtract				
Register from Register	001010dw | mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
ARITHMETIC (Continued)				
Register from Memory	0 0 1 0 1 0 0 w | mod reg r/m	7**	2**	a
Memory from Register	0 0 1 0 1 0 1 w | mod reg r/m	6*	1	a
Immediate from Register/Memory	1 0 0 0 0 0 s w | mod 1 0 1 r/m | immediate data	2/7**	0/1**	a
Immediate from Accumulator (Short Form)	0 0 1 0 1 1 0 w | immediate data	2		
SBB = Subtract with Borrow				
Register from Register	0 0 0 1 1 0 d w | mod reg r/m	2		
Register from Memory	0 0 0 1 1 0 0 w | mod reg r/m	7**	2**	a
Memory from Register	0 0 0 1 1 0 1 w | mod reg r/m	6*	1*	a
Immediate from Register/Memory	1 0 0 0 0 0 s w | mod 0 1 1 r/m | immediate data	2/7**	0/2**	a
Immediate from Accumulator (Short Form)	0 0 0 1 1 1 0 w | immediate data	2		
DEC = Decrement				
Register/Memory	1 1 1 1 1 1 1 w | reg 0 0 1 r/m	2/6**	0/2**	a
Register (Short Form)	0 1 0 0 1 reg	2		
CMP = Compare				
Register with Register	0 0 1 1 1 0 d w | mod reg r/m	2		
Memory with Register	0 0 1 1 1 0 0 w | mod reg r/m	5*	1*	a
Register with Memory	0 0 1 1 1 0 1 w | mod reg r/m	6**	2**	a
Immediate with Register/Memory	1 0 0 0 0 0 s w | mod 1 1 1 r/m | immediate data	2/5*	0/1*	a
Immediate with Accumulator (Short Form)	0 0 1 1 1 1 0 w | immediate data	2		
NEG = Change Sign	1 1 1 1 0 1 1 w | mod 0 1 1 r/m	2/6*	0/2*	a
AAA = ASCII Adjust for Add	0 0 1 1 0 1 1 1	4		
AAS = ASCII Adjust for Subtract	0 0 1 1 1 1 1 1	4		
DAA = Decimal Adjust for Add	0 0 1 0 0 1 1 1	4		
DAS = Decimal Adjust for Subtract	0 0 1 0 1 1 1 1	4		
MUL = Multiply (Unsigned)				
Accumulator with Register/Memory	1 1 1 1 0 1 1 w | mod 1 0 0 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
IMUL = Integer Multiply (Signed)				
Accumulator with Register/Memory	1 1 1 1 0 1 1 w | mod 1 0 1 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register with Register/Memory	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 1 1 | mod reg r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register/Memory with Immediate to Register	0 1 1 0 1 0 s 1 | mod reg r/m | immediate data			
—Word		13–26/14–27*	0/1*	a,n
—Doubleword		13–42/16–45*	0/2*	a,n

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
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	0/1	a, o
—Word		22/25*	0/1*	a, o
—Doubleword		38/43*	0/2*	a, o
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	0/1	a, o
—Word		27/30*	0/1	a, o
—Doubleword		43/48*	0/2*	a, o
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	19		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	17		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0	3		
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1	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**	0/2**	a
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	3/7**	0/2**	a
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	3/7**	0/2**	a
Through Carry **(RCL** and **RCR)**				
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	9/10**	0/2**	a
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	9/10**	10/2**	a
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	9/10**	0/2**	a
	TTT Instruction 000 ROL 001 ROR 010 RCL 011 RCR 100 SHL/SAL 101 SHR 111 SAR			
SHLD = Shift Left Double				
Register/Memory by Immediate	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**	0/2**	
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**	0/2**	
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**	0/2**	
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**	0/2**	
AND = And				
Register to Register	0 0 1 0 0 0 d w | mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
LOGIC (Continued)				
Register to Memory	0 0 1 0 0 0 0 w | mod reg r/m	7**	2**	a
Memory to Register	0 0 1 0 0 0 1 w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 1 0 0 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0 0 1 0 0 1 0 w | immediate data	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*	0/1*	a
Immediate Data and Register/Memory	1 1 1 1 0 1 1 w | mod 0 0 0 r/m | immediate data	2/5*	0/1*	a
Immediate Data and Accumulator (Short Form)	1 0 1 0 1 0 0 w | immediate data	2		
OR = Or				
Register to Register	0 0 0 0 1 0 d w | mod reg r/m	2		
Register to Memory	0 0 0 0 1 0 0 w | mod reg r/m	7**	2**	a
Memory to Register	0 0 0 0 1 0 1 w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 0 0 1 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0 0 0 0 1 1 0 w | immediate data	2		
XOR = Exclusive Or				
Register to Register	0 0 1 1 0 0 d w | mod reg r/m	2		
Register to Memory	0 0 1 1 0 0 0 w | mod reg r/m	7**	2**	a
Memory to Register	0 0 1 1 0 0 1 w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	1 0 0 0 0 0 0 w | mod 1 1 0 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0 0 1 1 0 1 0 w | immediate data	2		
NOT = Invert Register/Memory	1 1 1 1 0 1 1 w | mod 0 1 0 r/m	2/6**	0/2**	a
STRING MANIPULATION				
CMPS = Compare Byte Word	1 0 1 0 0 1 1 w	10*	2*	a
INS = Input Byte/Word from DX Port	0 1 1 0 1 1 0 w	9** 29**	1** 1**	a,f,k a,f,l
LODS = Load Byte/Word to AL/AX/EAX	1 0 1 0 1 1 0 w	5*	1*	a
MOVS = Move Byte Word	1 0 1 0 0 1 0 w	7**	2**	a
OUTS = Output Byte/Word to DX Port	0 1 1 0 1 1 1 w	8** 28**	1** 1**	a,f,k a,f,l
SCAS = Scan Byte Word	1 0 1 0 1 1 1 w	7*	1*	a
STOS = Store Byte/Word from AL/AX/EX	1 0 1 0 1 0 1 w	4*	1*	a
XLAT = Translate String	1 1 0 1 0 1 1 1	5*	1*	a
REPEATED STRING MANIPULATION Repeated by Count in CX or ECX				
REPE CMPS = Compare String (Find Non-Match)	1 1 1 1 0 0 1 1 | 1 0 1 0 0 1 1 w	5 + 9n**	2n**	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
REPEATED STRING MANIPULATION (Continued)				
REPNE CMPS = Compare String				
(Find Match)	11110010 | 1010011w	5 + 9n**	2n**	a
REP INS = Input String	11110011 | 0110110w	7 + 6n* 27 + 6n*	1n* 1n*	a,f,k a,f,l
REP LODS = Load String	11110011 | 1010110w	5 + 6n*	1n*	a
REP MOVS = Move String	11110011 | 1010010w	7 + 4n**	2n**	a
REP OUTS = Output String	11110011 | 0110111w	6 + 5n* 26 + 5n*	1n* 1n*	a,f,k a,f,l
REPE SCAS = Scan String				
(Find Non-AL/AX/EAX)	11110011 | 1010111w	5 + 8n*	1n*	a
REPNE SCAS = Scan String				
(Find AL/AX/EAX)	11110010 | 1010111w	5 + 8n*	1n*	a
REP STOS = Store String	11110011 | 1010101w	5 + 5n*	1n*	a
BIT MANIPULATION				
BSF = Scan Bit Forward	00001111 | 10111100 | mod reg r/m	10 + 3n**	2n**	a
BSR = Scan Bit Reverse	00001111 | 10111101 | mod reg r/m	10 + 3n**	2n**	a
BT = Test Bit				
Register/Memory, Immediate	00001111 | 10111010 | mod 100 r/m | immed 8-bit data	3/6*	0/1*	a
Register/Memory, Register	00001111 | 10100011 | mod reg r/m	3/12*	0/1*	a
BTC = Test Bit and Complement				
Register/Memory, Immediate	00001111 | 10111010 | mod 111 r/m | immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 | 10111011 | mod reg r/m	6/13*	0/2*	a
BTR = Test Bit and Reset				
Register/Memory, Immediate	00001111 | 10111010 | mod 110 r/m | immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 | 10110011 | mod reg r/m	6/13*	0/2*	a
BTS = Test Bit and Set				
Register/Memory, Immediate	00001111 | 10111010 | mod 101 r/m | immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 | 10101011 | mod reg r/m	6/13*	0/2*	a
CONTROL TRANSFER				
CALL = Call				
Direct within Segment	11101000 | full displacement	9 + m*	2	j
Register/Memory				
Indirect within Segment	11111111 | mod 010 r/m	9 + m/12 + m	2/3	a, j
Direct Intersegment	10011010 | unsigned full offset, selector	42 + m	9	c, d, j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)				
(Direct Intersegment)				
Via Call Gate to Same Privilege Level		64 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		98 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		106 + 8x + m	13 + 4x	a,c,d,j
From 386 Task to 386 TSS		392	124	a,c,d,j
Indirect Intersegment	11111111 | mod 011 r/m	46 + m	10	a,c,d,j
Via Call Gate to Same Privilege Level		68 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		102 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		110 + 8x + m	14 + 4x	a,c,d,j
From 386 Task to 386 TSS		399	130	a,c,d,j
JMP = Unconditional Jump				
Short	11101011 | 8-bit displacement	7 + m		j
Direct within Segment	11101001 | full displacement	7 + m		j
Register/Memory Indirect within Segment	11111111 | mod 100 r/m	9 + m/14 + m	2/4	a,j
Direct Intersegment	11101010 | unsigned full offset, selector	37 + m	5	c,d,j
Via Call Gate to Same Privilege Level		53 + m	9	a,c,d,j
From 386 Task to 386 TSS		395	124	a,c,d,j
Indirect Intersegment	11111111 | mod 101 r/m	37 + m	9	a,c,d,j
Via Call Gate to Same Privilege Level		59 + m	13	a,c,d,j
From 386 Task to 386 TSS		401	124	a,c,d,j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format			Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)						
RET = Return from CALL:						
Within Segment	11000011			12 + m	2	a,j,p
Within Segment Adding Immediate to SP	11000010	16-bit displ		12 + m	2	a,j,p
Intersegment	11001011			36 + m	4	a,c,d,j,p
Intersegment Adding Immediate to SP	11001010	16-bit displ		36 + m	4	a,c,d,j,p
to Different Privilege Level						
Intersegment				80	4	c,d,j,p
Intersegment Adding Immediate to SP				80	4	c,d,j,p
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		j
Full Displacement	00001111	10000000	full displacement	7 + m or 3		j
JNO = Jump on Not Overflow						
8-Bit Displacement	01110001	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000001	full displacement	7 + m or 3		j
JB/JNAE = Jump on Below/Not Above or Equal						
8-Bit Displacement	01110010	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000010	full displacement	7 + m or 3		j
JNB/JAE = Jump on Not Below/Above or Equal						
8-Bit Displacement	01110011	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000011	full displacement	7 + m or 3		j
JE/JZ = Jump on Equal/Zero						
8-Bit Displacement	01110100	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000100	full displacement	7 + m or 3		j
JNE/JNZ = Jump on Not Equal/Not Zero						
8-Bit Displacement	01110101	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000101	full displacement	7 + m or 3		j
JBE/JNA = Jump on Below or Equal/Not Above						
8-Bit Displacement	01110110	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000110	full displacement	7 + m or 3		j
JNBE/JA = Jump on Not Below or Equal/Above						
8-Bit Displacement	01110111	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10000111	full displacement	7 + m or 3		j
JS = Jump on Sign						
8-Bit Displacement	01111000	8-bit displ		7 + m or 3		j
Full Displacement	00001111	10001000	full displacement	7 + m or 3		j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONDITIONAL JUMPS (Continued)				
JNS = Jump on Not Sign				
8-Bit Displacement	01111001 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001001 | full displacement	7 + m or 3		j
JP/JPE = Jump on Parity/Parity Even				
8-Bit Displacement	01111010 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001010 | full displacement	7 + m or 3		j
JNP/JPO = Jump on Not Parity/Parity Odd				
8-Bit Displacement	01111011 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001011 | full displacement	7 + m or 3		j
JL/JNGE = Jump on Less/Not Greater or Equal				
8-Bit Displacement	01111100 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001100 | full displacement	7 + m or 3		j
JNL/JGE = Jump on Not Less/Greater or Equal				
8-Bit Displacement	01111101 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001101 | full displacement	7 + m or 3		j
JLE/JNG = Jump on Less or Equal/Not Greater				
8-Bit Displacement	01111110 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001110 | full displacement	7 + m or 3		j
JNLE/JG = Jump on Not Less or Equal/Greater				
8-Bit Displacement	01111111 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001111 | full displacement	7 + m or 3		j
JECXZ = Jump on ECX Zero	11100011 | 8-bit displ	9 + m or 5		j
(Address Size Prefix Differentiates JCXZ from JECXZ)				
LOOP = Loop ECX Times	11100010 | 8-bit displ	11 + m		j
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displ	11 + m		j
LOOPNZ/LOOPNE = Loop While Not Zero	11100000 | 8-bit displ	11 + m		j
CONDITIONAL BYTE SET				
NOTE: Times Are Register/Memory				
SETO = Set Byte on Overflow				
To Register/Memory	00001111 | 10010000 | mod 000 r/m	4/5*	0/1*	a
SETNO = Set Byte on Not Overflow				
To Register/Memory	00001111 | 10010001 | mod 000 r/m	4/5*	0/1*	a
SETB/SETNAE = Set Byte on Below/Not Above or Equal				
To Register/Memory	00001111 | 10010010 | mod 000 r/m	4/5*	0/1*	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format			Clock Counts	Number of Data Cycles	Notes
CONDITIONAL BYTE SET (Continued)						
SETNB = Set Byte on Not Below/Above or Equal						
To Register/Memory	00001111	10010011	mod000 r/m	4/5*	0/1*	a
SETE/SETZ = Set Byte on Equal/Zero						
To Register/Memory	00001111	10010100	mod000 r/m	4/5*	0/1*	a
SETNE/SETNZ = Set Byte on Not Equal/Not Zero						
To Register/Memory	00001111	10010101	mod000 r/m	4/5*	0/1*	a
SETBE/SETNA = Set Byte on Below or Equal/Not Above						
To Register/Memory	00001111	10010110	mod000 r/m	4/5*	0/1*	a
SETNBE/SETA = Set Byte on Not Below or Equal/Above						
To Register/Memory	00001111	10010111	mod000 r/m	4/5*	0/1*	a
SETS = Set Byte on Sign						
To Register/Memory	00001111	10011000	mod000 r/m	4/5*	0/1*	a
SETNS = Set Byte on Not Sign						
To Register/Memory	00001111	10011001	mod000 r/m	4/5*	0/1*	a
SETP/SETPE = Set Byte on Parity/Parity Even						
To Register/Memory	00001111	10011010	mod000 r/m	4/5*	0/1*	a
SETNP/SETPO = Set Byte on Not Parity/Parity Odd						
To Register/Memory	00001111	10011011	mod000 r/m	4/5*	0/1*	a
SETL/SETNGE = Set Byte on Less/Not Greater or Equal						
To Register/Memory	00001111	10011100	mod000 r/m	4/5*	0/1*	a
SETNL/SETGE = Set Byte on Not Less/Greater or Equal						
To Register/Memory	00001111	01111101	mod000 r/m	4/5*	0/1*	a
SETLE/SETNG = Set Byte on Less or Equal/Not Greater						
To Register/Memory	00001111	10011110	mod000 r/m	4/5*	0/1*	a
SETNLE/SETG = Set Byte on Not Less or Equal/Greater						
To Register/Memory	00001111	10011111	mod000 r/m	4/5*	0/1*	a
ENTER = Enter Procedure	11001000	16-bit displacement, 8-bit level				
L = 0				10		a
L = 1				14	1	a
L > 1				17 + 8(n − 1)	4(n − 1)	a
LEAVE = Leave Procedure	11001001			6		a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
INTERRUPT INSTRUCTIONS				
INT = Interrupt:				
Type Specified	11001101 | type			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		467	140	c,d,j,p
Type 3	11001100			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		308	138	c,d,j,p
INTO = Interrupt 4 if Overflow Flag Set	11001110			
If OF = 1:		3		
If OF = 0				
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		413	138	c,d,j,p

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
INTERRUPT INSTRUCTIONS (Continued)				
Bound = Out of Range Interrupt 5 if Detect Value	01100010 | mod reg r/m			
if in Range		10	0	a,c,d,j,o,p
if Out of Range:				
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		398	138	c,d,j,p
INTERRUPT RETURN				
IRET = Interrupt Return	11001111			
To the Same Privilege Level (within Task)		42	5	a,c,d,j,p
To Different Privilege Level (within Task)		86	5	a,c,d,j,p
From 386 Task to 386 TSS		328	138	c,d,j,p
PROCESSOR CONTROL				
HLT = HALT	11110100	5		b
MOV = Move to and from Control/Debug/Test Registers				
CR0 from register	00001111 | 00100010 | 11 eee reg	10		b
Register from CR0	00001111 | 00100000 | 11 eee reg	6		b
DR0–3 from Register	00001111 | 00100011 | 11 eee reg	22		b
DR6–7 from Register	00001111 | 00100011 | 11 eee reg	16		b
Register from DR6–7	00001111 | 00100001 | 11 eee reg	14		b
Register from DR0–3	00001111 | 00100001 | 11 eee reg	22		b
NOP = No Operation	10010000	3		
WAIT = Wait until $\overline{\text{BUSY}}$ Pin is Negated	10011011	6		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROCESSOR EXTENSION INSTRUCTIONS				
Processor Extension Escape	11011TTT | mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 80387SX Data Sheet		a
PREFIX BYTES				
Address Size Prefix	01100111	0		
LOCK = Bus Lock Prefix	11110000	0		f
Operand Size Prefix	01100110	0		
Segment Override Prefix				
CS:	00101110	0		
DS:	00111110	0		
ES:	00100110	0		
FS:	01100100	0		
GS:	01100101	0		
SS:	00110110	0		
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From Register/Memory	01100011 | mod reg r/m	20/21**	2**	a
LAR = Load Access Rights				
From Register/Memory	00001111 | 00000010 | mod reg r/m	17/18*	1*	a,c,i,p
LGDT = Load Global Descriptor				
Table Register	00001111 | 00000001 | mod 010 r/m	13**	3*	a,e
LIDT = Load Interrupt Descriptor				
Table Register	00001111 | 00000001 | mod 011 r/m	13**	3*	a,e
LLDT = Load Local Descriptor				
Table Register to Register/Memory	00001111 | 00000000 | mod 010 r/m	24/28*	5*	a,c,e,p
LMSW = Load Machine Status Word				
From Register/Memory	00001111 | 00000001 | mod 110 r/m	10/13*	1*	a,e
LSL = Load Segment Limit				
From Register/Memory	00001111 | 00000011 | mod reg r/m			
Byte-Granular Limit		24/27*	2*	a,c,i,p
Page-Granular Limit		29/32*	2*	a,c,i,p
LTR = Load Task Register				
From Register/Memory	00001111 | 00000000 | mod 001 r/m	27/31*	4*	a,c,e,p
SGDT = Store Global Descriptor				
Table Register	00001111 | 00000001 | mod 000 r/m	11*	3*	a
SIDT = Store Interrupt Descriptor				
Table Register	00001111 | 00000001 | mod 001 r/m	11*	3*	a
SLDT = Store Local Descriptor Table Register				
To Register/Memory	00001111 | 00000000 | mod 000 r/m	2/2*	4*	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROTECTION CONTROL (Continued)				
SMSW = Store Machine Status Word	00001111 | 00000001 | mod100 r/m	2/2*	1*	a, c
STR = Store Task Register				
To Register/Memory	00001111 | 00000000 | mod001 r/m	2/2*	1*	a
VERR = Verify Read Accesss				
Register/Memory	00001111 | 00000000 | mod100 r/m	10/11**	2**	a,c,i,p
VERW = Verify Write Accesss	00001111 | 00000000 | mod101 r/m	15/16**	2**	a,c,i,p

NOTES:
a. Exception 13 fault (general violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, and exception 12 (stack segment limit violation or not present) occurs.
b. For segment load operations, the CPL, RPL and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segments's descriptor must indicate ''present'' or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present occurs).
c. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{LOCK}$ to maintain descriptor integrity in multiprocessor systems.
d. 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 volated.
e. An exception 13 fault occurs if CPL is greater than 0.
f. An exception 13 fault occurs if CPL is greater than IOPL.
g. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL field of the flag register is updated only if CPL = 0.
h. Any violation of privelege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.
i. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or no present) will occur if the stack limit is violated by the operand's starting address.
j. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.
k. If CPL $\leq$ IOPL
l. If CPL $>$ IOPL
m. $\overline{LOCK}$ is automatically asserted, regardless of the presence or absence of the $\overline{LOCK}$ prefix.
n. The 80376 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) + 9 clocks:
if m = 0 then 12 clocks (where m is the multiplier)

o. An exception may occur, depending on the value of the operand.
p. $\overline{LOCK}$ is asserted during descriptor table accesses.

8.2 INSTRUCTION ENCODING

Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the ''mod r/m'' byte and ''scaled index'' byte, a displacement if required, and an immediate data field if required.

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

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

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

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

Figure 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 80376 instruction set. Further ahead, following Table 8.2, are detailed tables for each field.

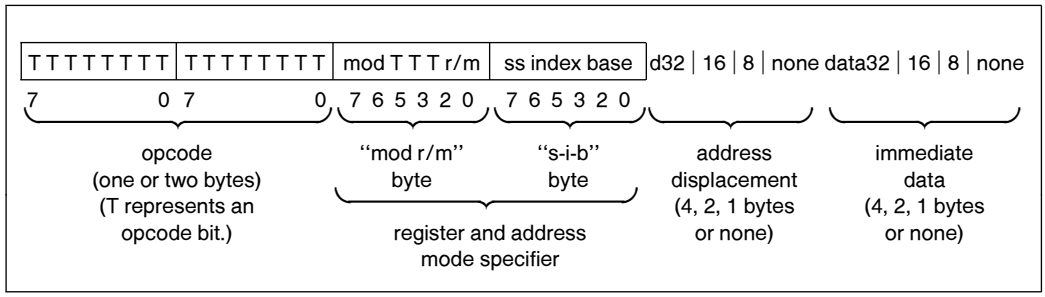


Figure 8.1. General Instruction Format

Table 8.2. Fields within 80376 Instructions

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

Note: Table 8.1 shows encoding of individual instructions.

16-Bit Extensions of the Instruction Set

Two prefixes, the operand size prefix (66H) and the effective address size prefix (67H), 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 (66H) and the effective address prefix will allow 16-bit data operation and 16-bit effective address calculations.

For instructions with more than one prefix, the order of prefixes is unimportant.

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

Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on.

ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction will execute as a 32-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 with 66H Prefix	Normal Operand Size
0	8 Bits	8 Bits
1	16 Bits	32 Bits

ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary 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 with 66H Prefix	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field with 66H Prefix		
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 without 66H Prefix		
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

ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the CS, DS, ES or SS segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the FS and GS segment registers to be specified also.

2-Bit sreg2 Field

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

3-Bit sreg3 Field

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

ENCODING OF ADDRESS MODE

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

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

The primary addressing byte, the ''mod r/m'' byte, also contains three bits (shown as TTT in Figure 8.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

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

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

Encoding of Normal Address Mode with "mod r/m" byte (no "s-i-b" byte present):

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

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

Register Specified by reg or r/m during Normal Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Register Specified by reg or r/m during 16-Bit Data Operations: (66H Prefix)		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Encoding of 16-bit Address Mode with "mod r/m" Byte Using 67H Prefix

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	SS:[BP + d8]
01 111	DS:[BX + d8]

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

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

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

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

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

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

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

ENCODING OF OPERATION DIRECTION (d) FIELD

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

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

ENCODING OF SIGN-EXTEND (s) FIELD

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

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

ENCODING OF CONTROL OR DEBUG REGISTER (eee) FIELD

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

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	Reserved
011	Reserved
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	

9.0 REVISION HISTORY

The sections significantly revised since version -003 are:

Section 1.0 Added $\overline{FLT}$ pin.

Section 4.4 Added description of FLOAT operation and ONCE Mode. Figure 4.20 is new.

Section 4.6 Added revision identifier information for change to CHMOS IV manufacturing process.

Section 5.0 Both packages now specified for 0°C–115°C case temperature operation. Thermal resistance values changed.

Section 6.3 I_{CC} Max. specifications changed from 400 mA (cold) and 360 mA (hot) to 275 mA (cold, 16 MHz) and 305 mA (cold, 20 MHz).

Section 6.4 HLDA Valid Delay, t_{14}, min. changed from 6 ns to 4 ns. Added 20 MHz A.C. specifications in Table 6.5. Replaced Capacitive Derating Curves in Figures 6.8–6.10 to reflect new manufacturing process. Replaced I_{CC} vs. Frequency data (Figure 6.11) to reflect new specifications.

The sections significantly revised since version -002 are:

Section 1.0 Modified table 1.1. to list pins in alphabetical order.

The sections significantly revised since version -001 are:

Section 2.0 Figure 2.0 was updated to show the 16-bit registers SI, DI, BP and SP.

Section 2.1 Figure 2.2 was updated to show the correct bit polarity for bit 4 in the CR0 register.

Section 2.1 Tables 2.1 and 2.2 were updated to include additional information on the EFLAGs and CR0 registers.

Section 2.3 Figure 2.3 was updated to more accurately reflect the addressing mechanism of the 80376.

Section 2.6 In the subsection **Maskable Interrupt** a paragraph was added to describe the effect of interrupt gates on the IF EFLAGs bit.

Section 2.8 Table 2.7 was updated to reflect the correct power up condition of the CR0 register.

Section 2.10 Figure 2.6 was updated to show the correct bit positions of the BT, BS and BD bits in the DR6 register.

Section 3.0 Figure 3.1 was updated to clearly show the address calculation process.

Section 3.2 The subsection **DESCRIPTORS** was elaborated upon to clearly define the relationship between the linear address space and physical address space of the 80376.

Section 3.2 Figures 3.3 and 3.4 were updated to show the AVL bit field.

Section 3.3 The last sentence in the first paragraph of subsection **PROTECTION AND I/O PERMISSION BIT MAP** was deleted. This was an incorrect statement.

Section 4.1 In the Subsection **ADDRESS BUS ($\overline{BHE}$, $\overline{BLE}$, A_{23}–A_1** last sentence in the first paragraph was updated to reflect the numerics operand addresses as 8000FCH and 8000FEH. Because the 80376 sometimes does a double word I/O access a second access to 8000FEH can be seen.

Section 4.1 The Subsection **Hold Lantencies** was updated to describe how 32-bit and unaligned accesses are internally locked but do not assert the $\overline{LOCK}$ signal.

Section 4.2 Table 4.6 was updated to show the correct active data bits during a $\overline{BLE}$ assertion.

9.0 REVISION HISTORY (Continued)

Section 4.4 This section was updated to correctly reflect the pipelining of the address and status of the 80376 as opposed to ''Address Pipelining'' which occurs on processors such as the 80286.

Section 4.6 Table 4.7 was updated to show the correct Revision number, 05H.

Section 4.7 Table 4.8 was updated to show the numerics operand register 8000FEH. This address is seen when the 80376 does a DWORD operation to the port address 8000FCH.

Section 5.0 In the first paragraph the case temperatures were updated to reflect the 0°C–115°C for the ceramic package and 0°C–110°C for the plastic package.

Section 6.2 Table 6.2 was updated to reflect the Case Temperature under Bias specification of −65°C–120°C.

Section 6.4 Figure 6.8 vertical axis was updated to reflect ''Output Valid Delay (ns)''.

Section 6.4 Figure 6.11 was updated to show typical I_{CC} vs Frequency for the 80376.

Section 8.1 The clock counts and opcodes for various instructions were updated to their correct values.

Section 8.2 The section **INSTRUCTION ENCODING** was appended to the data sheet.