# Intel387™ DX MATH COPROCESSOR

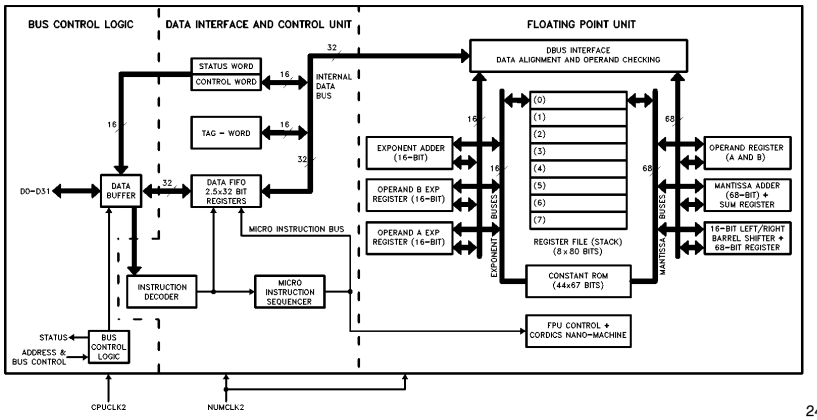
- **High Performance 80-Bit Internal Architecture**
- **Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
- **Expands Intel386™ DX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Directly Extends Intel386™ DX CPU Instruction Set to Include Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data Types**
- **Upward Object-Code Compatible from 8087 and 80287**
- **Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM**
- **Built-In Exception Handling**
- **Operates Independently of Real, Protected and Virtual-8086 Modes of the Intel386™ DX Microprocessor**
- **Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack**
- **Available in 68-Pin PGA Package**
- **One Version Supports 16 MHz–33 MHz Speeds**

(See Packaging Spec: Order #231369)

The Intel387™ DX Math CoProcessor (MCP) is an extension of the Intel386™ microprocessor architecture. The combination of the Intel387 DX MCP with the Intel386™ DX Microprocessor dramatically increases the processing speed of computer application software which utilize mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The Intel387 DX Math CoProcessor adds over seventy mnemonics to the Intel386 DX Microprocessor instruction set. Specific Intel387 DX MCP math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The Intel387 DX MCP supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The Intel387 DX Math CoProcessor is object code compatible with the Intel387 SX MCP, and upward object code compatible from the 80287 and 8087 math coprocessors. Object code for Intel386 DX/Intel387 DX is also compatible with the Intel486™ microprocessor. The Intel387 DX MCP is manufactured on 1 micron, CHMOS IV technology and packaged in a 68-pin PGA package.




240448–1

**Figure 0.1. Intel387™ DX Math CoProcessor Block Diagram**



**March 1992** **Order Number: 240448-005**

# Intel387™ DX Math CoProcessor

## CONTENTS

## CONTENTS

PAGE



**FIGURES**

## CONTENTS

PAGE

**FIGURES** (Continued)



**TABLES**

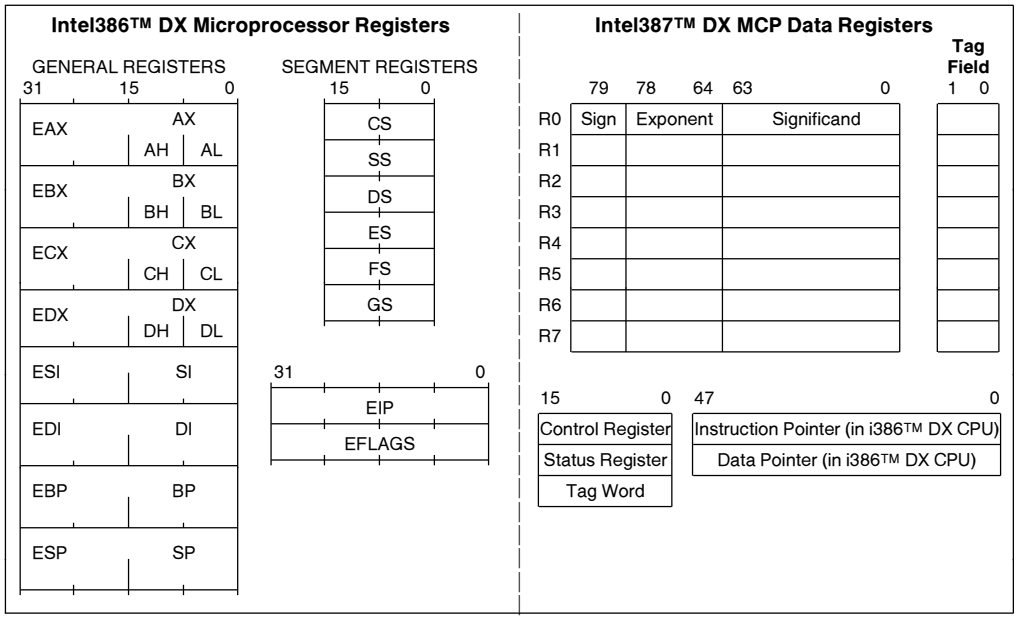



**Figure 1.1. Intel386™ DX Microprocessor and Intel387™ DX Math Coprocessor Register Set**

## 1.0 FUNCTIONAL DESCRIPTION

The Intel387™ DX Math Coprocessor provides arithmetic instructions for a variety of numeric data types in Intel386™ DX Microprocessor systems. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The Intel387 DX MCP effectively extends the register and instruction set of a Intel386 DX Microprocessor system for existing data types and adds several new data types as well. Figure 1.1 shows the model of registers visible to programs. Essentially, the Intel387 DX MCP can be treated as an additional resource or an extension to the Intel386 DX Microprocessor. The Intel386 DX Microprocessor together with a Intel387 DX MCP can be used as a single unified system.

The Intel387 DX MCP works the same whether the Intel386 DX Microprocessor is executing in real-address mode, protected mode, or virtual-8086 mode. All memory access is handled by the Intel386 DX Microprocessor; the Intel387 DX MCP merely operates on instructions and values passed to it by the Intel386 DX Microprocessor. Therefore, the Intel387 DX MCP is not sensitive to the processing mode of the Intel386 DX Microprocessor.

In real-address mode and virtual-8086 mode, the Intel386 DX Microprocessor and Intel387 DX MCP are completely upward compatible with software for 8086/8087, 80286/80287 real-address mode, and Intel386 DX Microprocessor and 80287 Coprocessor real-address mode systems.

In protected mode, the Intel386 DX Microprocessor and Intel387 DX MCP are completely upward compatible with software for 80286/80287 protected mode, and Intel386 DX Microprocessor and 80287 Coprocessor protected mode systems.

The only differences of operation that may appear when 8086/8087 programs are ported to a protected-mode Intel386 DX Microprocessor and Intel387 DX MCP system (*not* using virtual-8086 mode), is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR and FSAVE. These instructions are normally used only by exception handlers and operating systems, not by applications programs.

The Intel387 DX MCP contains three functional units that can operate in parallel to increase system performance. The Intel386 DX Microprocessor can be transferring commands and data to the MCP *bus control logic* for the next instruction while the MCP *floating-point unit* is performing the current numeric instruction.

## 2.0 PROGRAMMING INTERFACE

The MCP adds to the Intel386 DX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the MCP requires no special programming tools, because all new instructions and data types are directly supported by the Intel386 DX CPU assembler and compilers for high-level languages. All 8086/8088 development tools that support the 8087 can also be used to develop software for the Intel386 DX Microprocessor and Intel387 DX Math Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the Intel386 DX Microprocessor and Intel387 DX Math Coprocessor.

All communication between the Intel386 DX Microprocessor and the MCP is transparent to applications software. The CPU automatically controls the MCP whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the MCP. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 6 at the end of this data sheet lists by class the instructions that the MCP adds to the instruction set of the Intel386 DX Microprocessor system.

### 2.1 Data Types

Table 2.1 lists the seven data types that the Intel387 DX MCP supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses evenly divisible by four (doubleword boundaries); operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

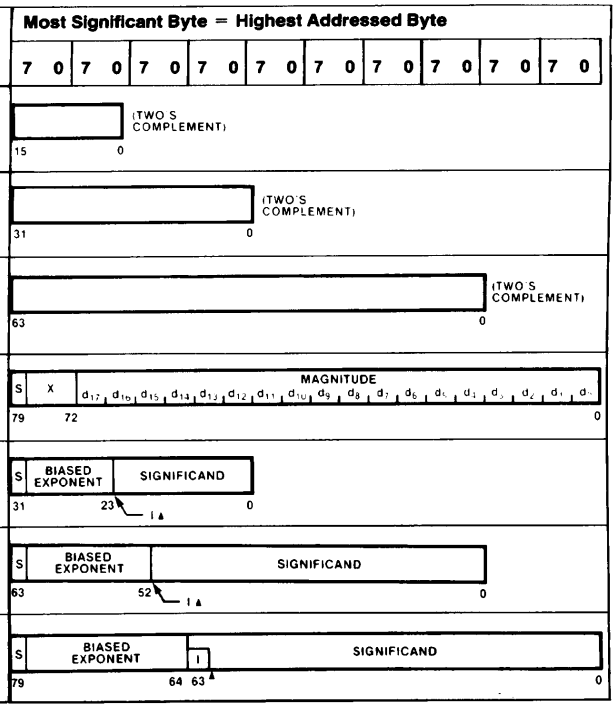
Internally, the Intel387 DX MCP holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

### 2.2 Numeric Operands

A typical MCP instruction accepts one or two operands and produces a single result. In two-operand instructions, one operand is the contents of an MCP register, while the other may be a memory location. The operands of some instructions are predefined; for example FSQRT always takes the square root of the number in the top stack element.

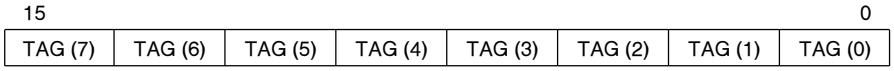
**Table 2.1. Intel387™ DX MCP Data Type Representation in Memory**

| Data Formats | Range | Precision | Most Significant Byte = Highest Addressed Byte |
|---|---|---|---|
| Word Integer | $\pm 10^4$ | 16 Bits | |
| Short Integer | $\pm 10^9$ | 32 Bits | |
| Long Integer | $\pm 10^{18}$ | 64 Bits | |
| Packed BCD | $\pm 10^{\pm 18}$ | 18 Digits | |
| Single Precision | $\pm 10^{\pm 38}$ | 24 Bits | |
| Double Precision | $\pm 10^{\pm 308}$ | 53 Bits | |
| Extended Precision | $\pm 10^{\pm 4932}$ | 64 Bits | |




240448–2

**NOTES:**
(1) S = Sign bit (0 = positive, 1 = negative)
(2) $d_n$ = Decimal digit (two per byte)
(3) X = Bits have no significance; Intel387™ DX MCP ignores when loading, zeros when storing
(4)▲ = Position of implicit binary point
(5) I = Integer bit of significand; stored in temporary real, implicit in single and double precision
(6) Exponent Bias (normalized values):
Single: 127 (7FH)
Double: 1023 (3FFH)
Extended Real: 16383 (3FFFH)
(7) Packed BCD: $(-1)^S$ $(D_{17}...D_0)$
(8) Real: $(-1)^S$ $(2^{E\text{-BIAS}})$ $(F_0\ F_1...)$




**NOTE:**
The index i of tag(i) is **not** top-relative. A program typically uses the ''top'' field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

**Figure 2.1. Intel387™ DX MCP Tag Word**

## 2.3 Register Set

Figure 1.1 shows the Intel387 DX MCP register set. When an MCP is present in a system, programmers may use these registers in addition to the registers normally available on the Intel386 DX CPU.

### 2.3.1 DATA REGISTERS

Intel387 DX MCP computations use the MCP's data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers in the MCP is 80 bits wide and is divided into ''fields'' corresponding to the MCPs extended-precision real data type.

The Intel387 DX MCP register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A ''push'' operation decrements TOP by one and loads a value into the new top register. A ''pop'' operation stores the value from the current top register and then increments TOP by one. Like the Intel386 DX Microprocessor stacks in memory, the MCP register stack grows ''down'' toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to user. This explicit register addressing is also relative to TOP.

### 2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2.1 shows. Each two-bit tag represents one of the eight numerics registers. The principal function of the tag word is to optimize the MCPs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

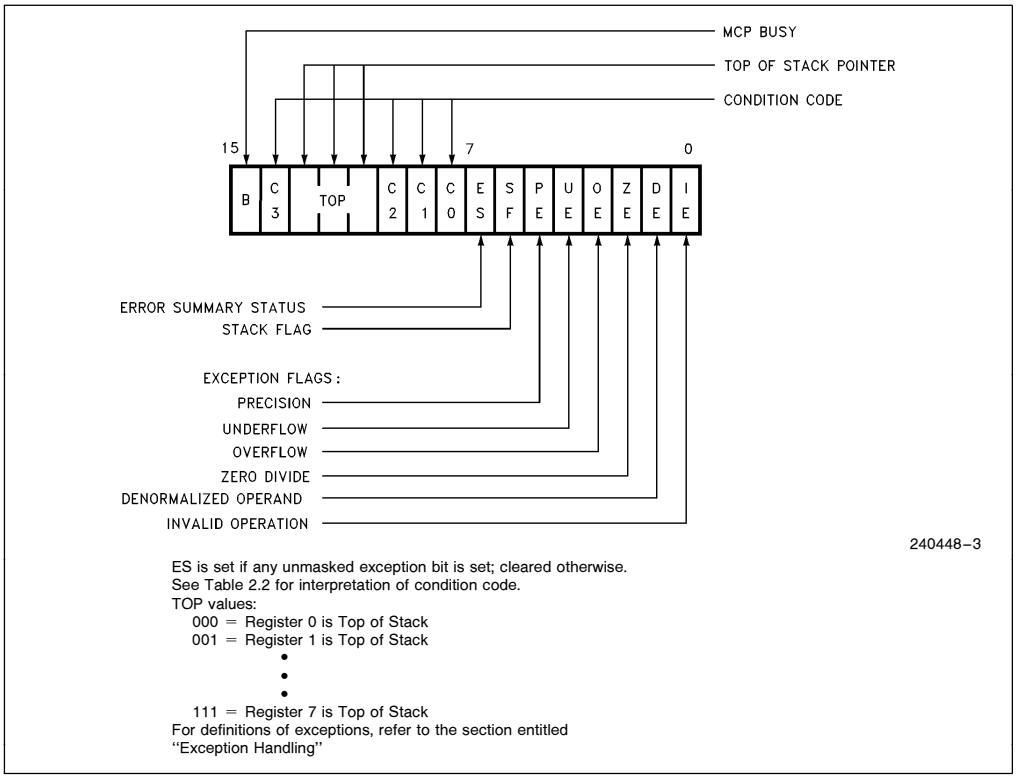



**Figure 2.2. MCP Status Word**

### 2.3.3 STATUS WORD

The 16-bit status word (in the status register) shown in Figure 2.2 reflects the overall state of the MCP. It may be read and inspected by CPU code.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It reflects the contents of the ES bit (bit 7 of the status word), not the status of the BUSY# output of the Intel387 DX MCP.

Bits 13–11 (TOP) point to the Intel387 DX MCP register that is the current top-of-stack.

The four numeric condition code bits ($C_3$–$C_0$) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2.2 through 2.5.

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 ($C_1$) distinguishes between stack overflow ($C_1$ = 1) and underflow ($C_1$ = 0).

Figure 2.2 shows the six exception flags in bits 5–0 of the status word. Bits 5–0 are set to indicate that the MCP has detected an exception while executing an instruction. A later section entitled ''Exception Handling'' explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the MCP is activated immediately.

**Table 2.2. Condition Code Interpretation**

| Instruction | C0 (S) | C3 (Z) | C1 (A) | C2 (C) |
|---|---|---|---|---|
| FPREM, FPREM1 (see Table 2.3) | Three least significant bits of quotient<br>Q2 | Q0 | Q1 or O/U# | Reduction<br>0 = complete<br>1 = incomplete |
| FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP | Result of comparison (see Table 2.4) | | Zero or O/U# | Operand is not comparable (Table 2.4) |
| FXAM | Operand class (see Table 2.5) | | Sign or O/U# | Operand class (Table 2.5) |
| FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real) | UNDEFINED | | Zero or O/U# | UNDEFINED |
| FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1 | UNDEFINED | | Roundup or O/U# | UNDEFINED |
| FPTAN, FSIN FCOS, FSINCOS | UNDEFINED | | Roundup or O/U#, undefined if C2 = 1 | Reduction<br>0 = complete<br>1 = incomplete |
| FLDENV, FRSTOR | Each bit loaded from memory | | | |
| FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE | UNDEFINED | | | |

| | |
|---|---|
| O/U# | When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0). |
| Reduction | If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack. |
| Roundup | When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward. |
| UNDEFINED | Do not rely on finding any specific value in these bits. |

**Table 2.3. Condition Code Interpretation after FPREM and FPREM1 Instructions**

| Condition Code | | | | Interpretation after FPREM and FPREM1 | |
|---|---|---|---|---|---|
| C2 | C3 | C1 | C0 | | |
| 1 | X | X | X | Incomplete Reduction: further interation required for complete reduction | |
| | Q1 | Q0 | Q2 | Q MOD8 | Complete Reduction: C0, C3, C1 contain three least significant bits of quotient |
| 0 | 0 | 0 | 0 | 0 | |
| | 0 | 1 | 0 | 1 | |
| | 1 | 0 | 0 | 2 | |
| | 1 | 1 | 0 | 3 | |
| | 0 | 0 | 1 | 4 | |
| | 0 | 1 | 1 | 5 | |
| | 1 | 0 | 1 | 6 | |
| | 1 | 1 | 1 | 7 | |

**Table 2.4. Condition Code Resulting from Comparison**

| Order | C3 | C2 | C0 |
|---|---|---|---|
| TOP > Operand | 0 | 0 | 0 |
| TOP < Operand | 0 | 0 | 1 |
| TOP = Operand | 1 | 0 | 0 |
| Unordered | 1 | 1 | 1 |

**Table 2.5. Condition Code Defining Operand Class**

| C3 | C2 | C1 | C0 | Value at TOP |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | + Unsupported |
| 0 | 0 | 0 | 1 | + NaN |
| 0 | 0 | 1 | 0 | − Unsupported |
| 0 | 0 | 1 | 1 | − NaN |
| 0 | 1 | 0 | 0 | + Normal |
| 0 | 1 | 0 | 1 | + Infinity |
| 0 | 1 | 1 | 0 | − Normal |
| 0 | 1 | 1 | 1 | − Infinity |
| 1 | 0 | 0 | 0 | + 0 |
| 1 | 0 | 0 | 1 | + Empty |
| 1 | 0 | 1 | 0 | − 0 |
| 1 | 0 | 1 | 1 | − Empty |
| 1 | 1 | 0 | 0 | + Denormal |
| 1 | 1 | 1 | 0 | − Denormal |

### 2.3.4 INSTRUCTION AND DATA POINTERS

Because the MCP operates in parallel with the CPU, any errors detected by the MCP may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the Intel386 DX Microprocessor and Intel387 DX Math CoProcessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written error handlers. These registers are actually located in the Intel386 DX CPU, but appear to be located in the MCP because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. (In the 8086/8087 and 80286/80287, these registers are located in the MCP.) Whenever the Intel386 DX CPU decodes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the Intel386 DX Microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the Intel386 DX Microprocessor is in virtual-8086 mode, the real-address mode formats are used. (See Figures 2.3 through 2.6.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the Intel386 DX Microprocessor registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

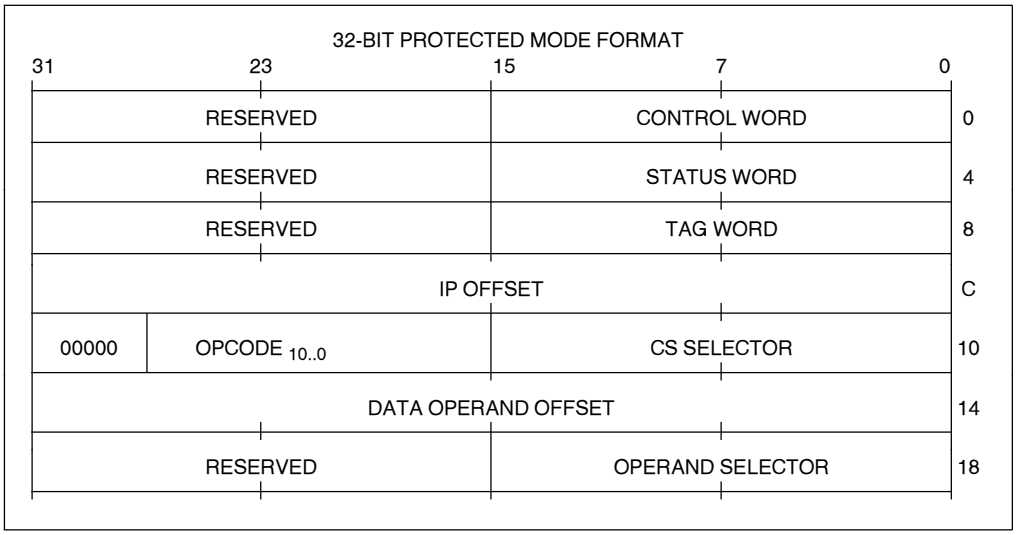



**Figure 2.3. Protected Mode Intel387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format**

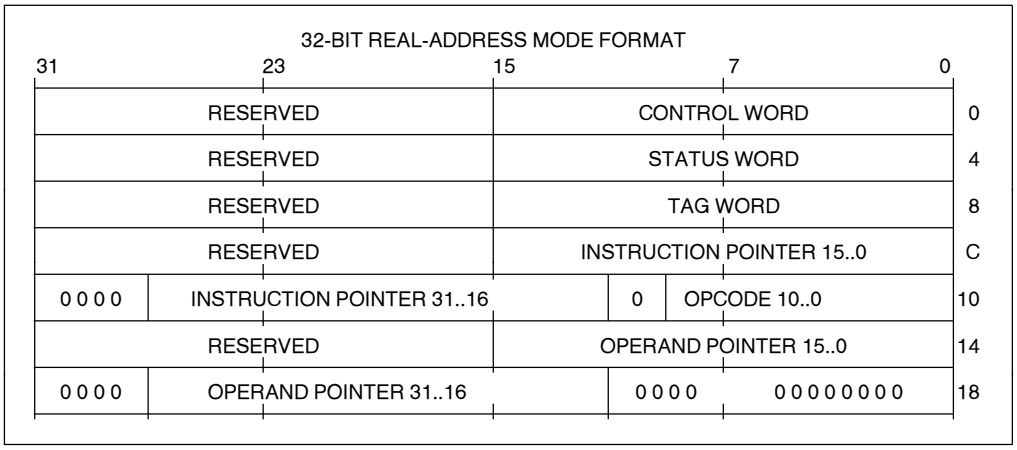



**Figure 2.4. Real Mode Intel387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format**

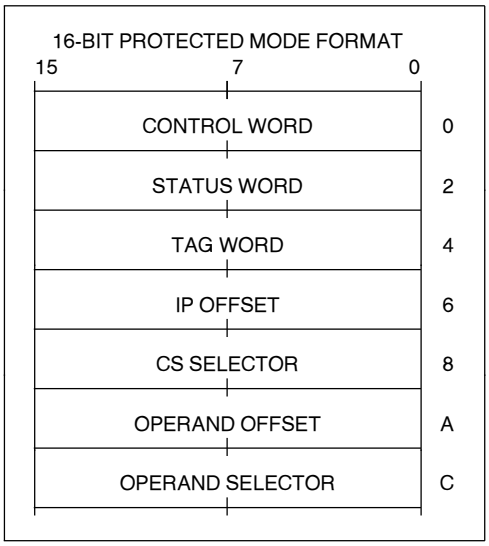



**Figure 2.5. Protected Mode Intel387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format**

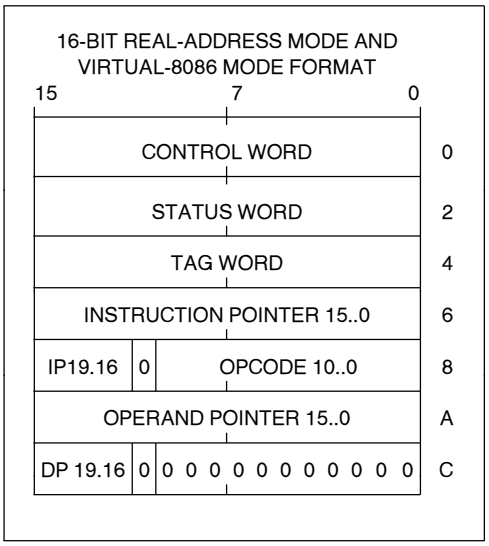



**Figure 2.6. Real Mode Intel387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format**

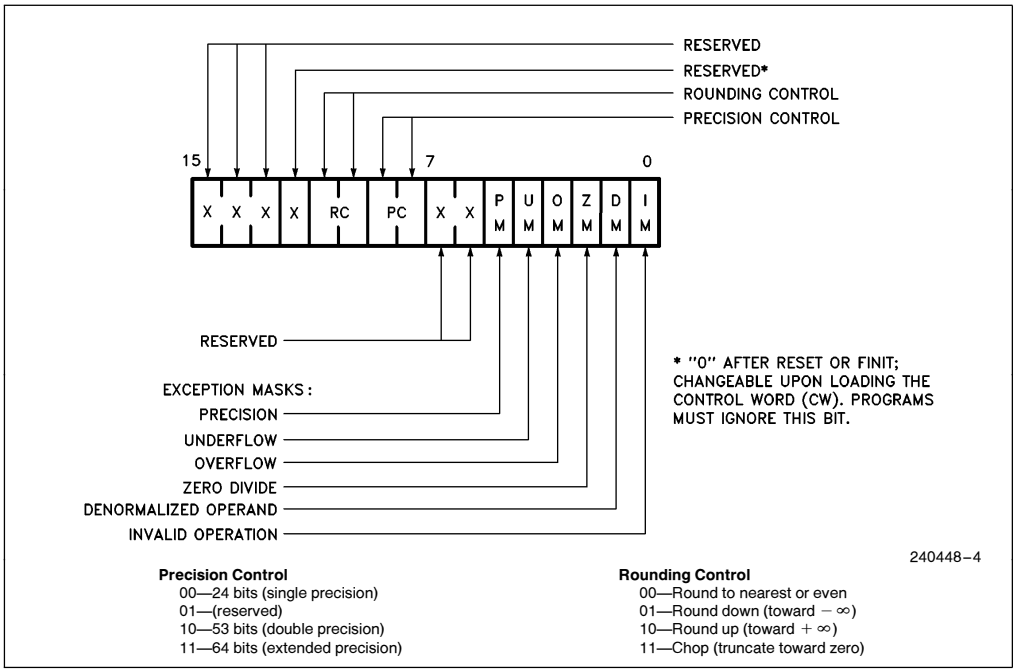



**Figure 2.7. Intel387™ DX MCP Control Word**

### 2.3.5 CONTROL WORD

The MCP provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.7 shows the format and encoding of fields in the control word.

The low-order byte of this control word configures the MCP error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the MCP recognizes.

The high-order byte of the control word configures the MCP operating mode, including precision and rounding.

- Bit 12 no longer defines infinity control and is a reserved bit. Only affine closure is supported for infinity arithmetic. The bit is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.
- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.
- The precision control (PC) bits (bits 9–8) can be used to set the MCP internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

## 2.4 Interrupt Description

Several interrupts of the Intel386 DX CPU are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.6 shows these interrupts and their causes.

**Table 2.6. Intel386™ DX Microprocessor Interrupt Vectors Reserved for MCP**

| Interrupt Number | Cause of Interrupt |
|---|---|
| 7 | An ESC instruction was encountered when EM or TS of the Intel386™ DX CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current MCP context may not belong to the current task. |
| 9 | An operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for small segments, 0FFFFFFFFH for big segments, zero for expand-down segments) and spanned inaccessible addresses(1). The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. As with the 80286/80287, the segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The return address on the stack does not necessarily point to the failing instruction nor to the following instruction. The interrupt can be avoided by never allowing numeric data to start within 108 bytes of the end of a segment. |
| 13 | The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The Intel387™ DX MCP has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction. |
| 16 | The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The Intel386™ DX CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the MCP. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt. |

**1.** An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFC will span addresses FFFC–FFFF and 0000-0003; however addresses FFFE and FFFF are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible pages but intermediate bytes of the operand fall in a not-present page or a page to which the procedure does not have access rights.

## 2.5 Exception Handling

The Intel387 DX MCP detects six different exception conditions that can occur during instruction execution. Table 2.7 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the MCP if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 7 occurs. The exception condition must be resolved via an interrupt service routine. The Intel386 DX Microprocessor saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

## 2.6 Initialization

Intel387 DX MCP initialization software must execute an FNINIT instruction (i.e. an FINIT without a preceding WAIT) to clear ERROR#. After a hardware RESET, the ERROR# output is asserted to indicate that a Intel387 DX MCP is present. To accomplish this, the IE and ES bits of the status word are set, and the IM bit in the control word is reset. After FNINIT, the status word and the control word have the same values as in an 80287 after RESET.

## 2.7 8087 and 80287 Compatibility

This section summarizes the differences between the Intel387 DX MCP and the 80287. Any migration from the 8087 directly to the Intel387 DX MCP must also take into account the differences between the 8087 and the 80287 as listed in Appendix A.

Many changes have been designed into the Intel387 DX MCP to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

### 2.7.1 GENERAL DIFFERENCES

The Intel387 DX MCP supports only affine closure for infinity arithmetic, not projective closure. Bit 12 of the Control Word (CW) no longer defines infinity control. It is a reserved bit; but it is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.

Operands for FSCALE and FPATAN are no longer restricted in range (except for $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.

The results of transcendental operations may be slightly different from those computed by 80287.

In the case of FPTAN, the Intel387 DX MCP supplies a true tangent result in ST(1), and (always) a floating point 1 in ST.

Rounding control is in effect for FLD *constant*.

Software cannot change entries of the tag word to values (other than empty) that do not reflect the actual register contents.

After reset, FINIT, and incomplete FPREM, the Intel387 DX MCP resets to zero the condition code bits $C_3$–$C_0$ of the status word.

In conformance with the IEEE standard, the Intel387 DX MCP does not support the special data formats: pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

**Table 2.7. Exceptions**

| Exception | Cause | Default Action (if exception is masked) |
|---|---|---|
| Invalid Operation | Operation on a signaling NaN, unsupported format, indeterminate form ($0*\infty$, 0/0, $(+\infty)+(-\infty)$), etc.), or stack overflow/underflow (SF is also set). | Result is a quiet NaN, integer indefinite, or BCD indefinite |
| Denormalized Operand | At least one of the operands is denormalized, i.e. it has the smallest exponent but a nonzero significand. | Normal processing continues |
| Zero Divisor | The divisor is zero while the dividend is a noninfinite, nonzero number. | Result is $\infty$ |
| Overflow | The result is too large in magnitude to fit in the specified format. | Result is largest finite value or $\infty$ |
| Underflow | The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy. | Result is denormalized or zero |
| Inexact Result (Precision) | The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode. | Normal processing continues |

### 2.7.2 EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the Intel387 DX MCP:

1. When the overflow or underflow exception is masked, the Intel387 DX MCP differs from the 80287 in rounding when overflow or underflow occurs. The Intel387 DX MCP produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the Intel387 DX MCP sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the Intel387 DX MCP automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signaling NaN, FLD *single/double precision* signals an invalid-operand exception.
12. The Intel387 DX MCP only generates quiet NaNs (as on the 80287); however, the Intel387 DX MCP distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.
14. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1)) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
    - FSCALE(0, $\infty$) generates the invalid operation exception.
    - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE(finite, $+\infty$) generates $\infty$ with the same sign as the scaled operand.

    The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The Intel387 DX MCP returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

## 3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

## 3.1 Signal Description

In the following signal descriptions, the Intel387 DX Math Coprocessor pins are grouped by function as follows:

1. Execution control—CPUCLK2, NUMCLK2, CKM, RESETIN
2. MCP handshake—PEREQ, BUSY#, ERROR#
3. Bus interface pins—D31–D0, W/R#, ADS#, READY#, READYO#
4. Chip/Port Select—STEN, NPS1#, NPS2, CMD0#
5. Power supplies—$V_{CC}$, $V_{SS}$

Table 3.1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D31–D0 are also tristate; they leave floating state only in read cycles when the MCP is selected (i.e. when STEN, NPS1#, and NPS2 are all active).

Figure 3.1 and Table 3.2 together show the location of every pin in the pin grid array.

**Table 3.1. Intel387™ DX MCP Pin Summary**

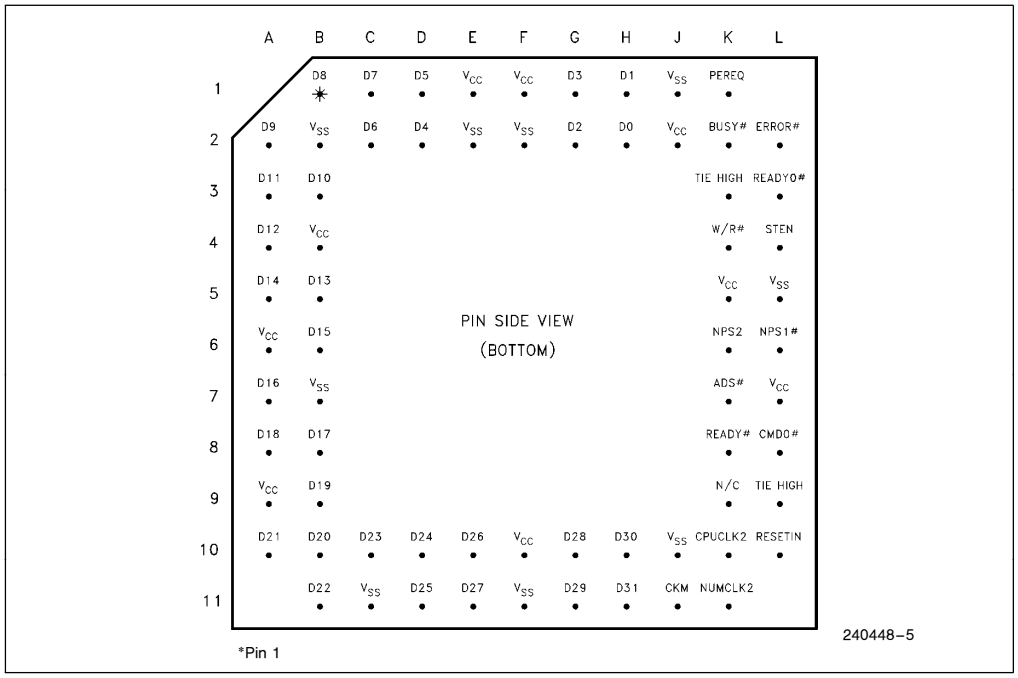
| Pin Name | Function | Active State | Input/ Output | Referenced To |
|---|---|---|---|---|
| CPUCLK2 | Intel386™ DX CPU CLocK 2 | | I | |
| NUMCLK2 | Intel387™ DX MCP CLocK 2 | | I | |
| CKM | Intel387™ DX MCP CLocKing Mode | | I | |
| RESETIN | System reset | High | I | CPUCLK2 |
| PEREQ | Processor Extension REQuest | High | O | CPUCLK2/STEN |
| BUSY# | Busy status | Low | O | CPUCLK2/STEN |
| ERROR# | Error status | Low | O | NUMCLK2/STEN |
| D31–D0 | Data pins | High | I/O | CPUCLK2 |
| W/R# | Write/Read bus cycle | Hi/Lo | I | CPUCLK2 |
| ADS# | ADdress Strobe | Low | I | CPUCLK2 |
| READY# | Bus ready input | Low | I | CPUCLK2 |
| READYO# | Ready output | Low | O | CPUCLK2/STEN |
| STEN | STatus ENable | High | I | CPUCLK2 |
| NPS1# | MCP select #1 | Low | I | CPUCLK2 |
| NPS2 | MCP select #2 | High | I | CPUCLK2 |
| CMD0# | CoMmanD | Low | I | CPUCLK2 |
| $V_{CC}$ | | | I | |
| $V_{SS}$ | | | I | |

**NOTE:**
STEN is referenced to only when getting the output pins into or out of tristate mode.

**Table 3.2. Intel387™ DX MCP Pin Cross-Reference**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ADS# | — | K7 | D18 | — | A8 | STEN | — | L4 |
| BUSY# | — | K2 | D19 | — | B9 | W/R# | — | K4 |
| CKM | — | J11 | D20 | — | B10 | | | |
| CPUCLK24 | — | K10 | D21 | — | A10 | $V_{CC}$ | — | A6, A9, B4, |
| CMD0# | — | L8 | D22 | — | B11 | | | E1, F1, F10, |
| D0 | — | H2 | D23 | — | C10 | | | J2, K5, |
| D1 | — | H1 | D24 | — | D10 | | | L7 |
| D2 | — | G2 | D25 | — | D11 | | | |
| D3 | — | G1 | D26 | — | E10 | $V_{SS}$ | — | B2, B7, C11, |
| D4 | — | D2 | D27 | — | E11 | | | E2, F2, F11, |
| D5 | — | D1 | D28 | — | G10 | | | J1, J10, L5 |
| D6 | — | C2 | D29 | — | G11 | | | |
| D7 | — | C1 | D30 | — | H10 | NO CONNECT | — | K9 |
| D8 | — | B1 | D31 | — | H11 | TIE HIGH | — | K3, L9* |
| D9 | — | A2 | ERROR# | — | L2 | | | |
| D10 | — | B3 | NPS1# | — | L6 | | | |
| D11 | — | A3 | NPS2 | — | K6 | | | |
| D12 | — | A4 | NUMCLK2 | — | K11 | | | |
| D13 | — | B5 | PEREQ | — | K1 | | | |
| D14 | — | A5 | READY# | — | K8 | | | |
| D15 | — | B6 | READYO# | — | L3 | | | |
| D16 | — | A7 | RESETIN | — | L10 | | | |
| D17 | — | B8 | | | | | | |

*Tie high pins may either be tied high with a pullup resistor or connected to $V_{CC}$.



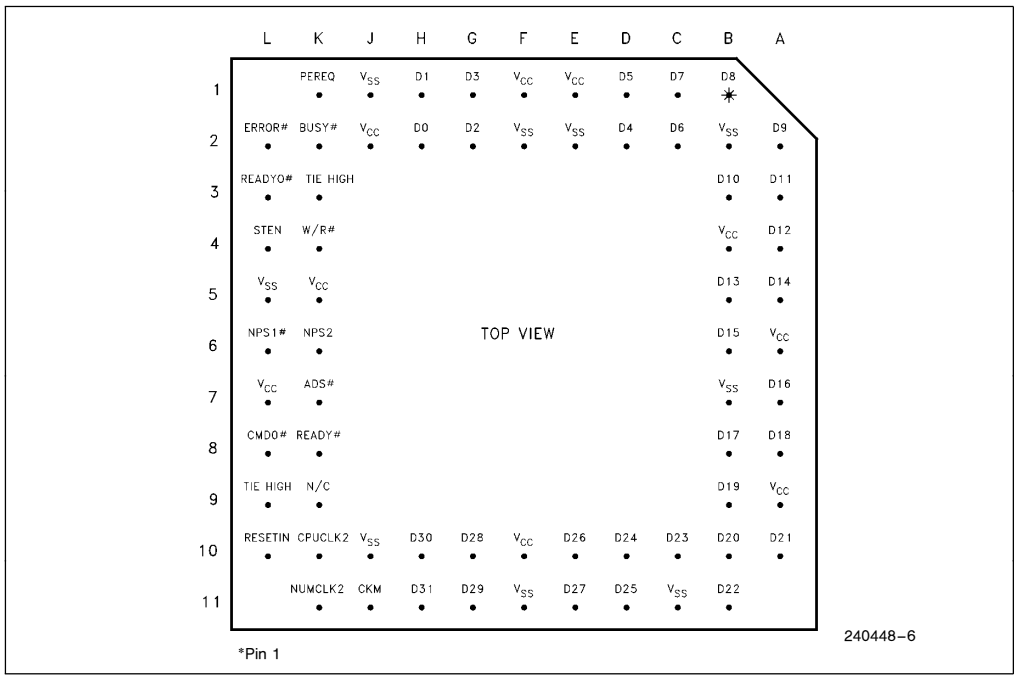




**Figure 3.1. Intel387™ DX MCP Pin Configuration**

### 3.1.1 Intel386™ DX CPU CLOCK 2 (CPUCLK2)

This input uses the Intel386 DX CPU CLK2 signal to time the bus control logic. Several other MCP signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating-point unit of the MCP. This pin requires MOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

### 3.1.2 Intel387™ DX MCP CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode) this pin provides the clock for the data interface and control unit and the floating-point unit of the MCP. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. When CKM = 1 (synchronous mode) this pin is ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires TTL-level input.

### 3.1.3 Intel387™ DX MCP CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to $V_{CC}$, the MCP operates in synchronous mode; when strapped to $V_{SS}$, the MCP operates in asynchronous mode. These modes relate to clocking of the data interface and control unit and the floating-point unit only; the bus control logic always operates synchronously with respect to the Intel386 DX Microprocessor.

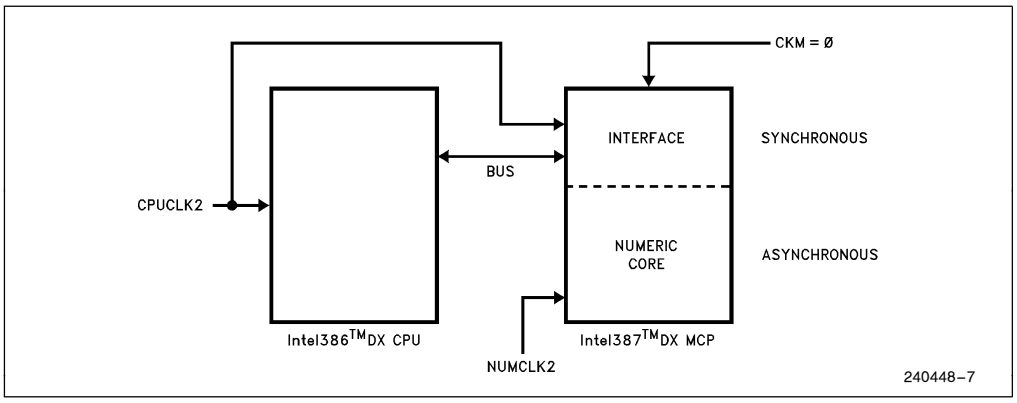



**Figure 3.2. Asynchronous Operation**

### 3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the MCP to terminate its present activity and to enter a dormant state. RESETIN must remain HIGH for at least 40 NUMCLK2 periods. The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by 2) is the same as the phase of the internal clock of the Intel386 DX CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first MCP instruction is written into the Intel387 DX MCP. This pin should be connected to the Intel386 DX CPU RESET pin. Table 3.3 shows the status of other pins after a reset.

**Table 3.3. Output Pin Status During Reset**

| Pin Value | Pin Name |
|---|---|
| HIGH | READYO#, BUSY# |
| LOW | PEREQ, ERROR# |
| Tri-State OFF | D31–D0 |

### 3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the Intel386 DX CPU that the MCP is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is referenced to CPUCLK2. It should be connected to the Intel386 DX CPU PEREQ input.

### 3.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the Intel386 DX CPU that the MCP is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the Intel386 DX CPU BUSY# pin.

### 3.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bits of the status register. When active, it indicates that an unmasked exception has occurred (except that, immediately after a reset, it indicates to the Intel386 DX Microprocessor that a Intel387 DX MCP is present in the system). This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, and FNSAVE. This signal is referenced to NUMCLK2. It should be connected to the Intel386 DX CPU ERROR# pin.

### 3.1.8 DATA PINS (D31–D0)

These bidirectional pins are used to transfer data and opcodes between the Intel386 DX CPU and Intel387 DX MCP. They are normally connected directly to the corresponding Intel386 DX CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to CPUCLK2.

### 3.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the MCP whether the Intel386 DX CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the Intel386 DX CPU W/R# pin. HIGH indicates a write cycle; LOW, a read cycle. This input is ignored if any of the signals STEN, NPS1#, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

### 3.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input indicates when the MCP bus-control logic may sample W/R# and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the Intel386 DX CPU ADS# pin.

#### 3.1.11 BUS READY INPUT (READY#)

This input indicates to the MCP when a Intel386 DX CPU bus cycle is to be terminated. It is used by the bus-control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the Intel386 DX CPU READY# input. Setup and hold times are referenced to CPUCLK2.

#### 3.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the Intel386 DX CPU READY# input. Refer to section 3.4 ''Bus Operation'' for details. This pin is activated only during bus cycles that select the MCP. This signal is referenced to CPUCLK2.

#### 3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the MCP. When inactive, this pin forces BUSY#, PEREQ, ERROR#, and READYO# outputs into floating state. D31–D0 are normally floating and leave floating state only if STEN is active and additional conditions are met. STEN also causes the chip to recognize its other chip-select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the MCP. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing, STEN should be connected to $V_{CC}$. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e. if STEN changes state during a Intel387 DX MCP bus cycle, it should change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

#### 3.1.14 MCP Select #1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a Intel386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the MCP. This pin should be connected directly to the Intel386 DX CPU M/IO# pin, so that the MCP is selected only when the Intel386 DX CPU performs I/O cycles. Setup and hold times are referenced to CPUCLK2.

#### 3.1.15 MCP SELECT #2 (NPS2)

When active (along with STEN and NPS1#) in the first period of an Intel386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the MCP. This pin should be connected directly to the Intel386 DX CPU A31 pin, so that the MCP is selected only when the Intel386 DX CPU uses one of the I/O addresses reserved for the MCP (800000F8 or 800000FC). Setup and hold times are referenced to CPUCLK2.

#### 3.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active) or data (CMD0# inactive) is being sent to the MCP. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0# inactive) is being read. CMD0# should be connected directly to the A2 output of the Intel386 DX Microprocessor. Setup and hold times are referenced to CPUCLK2.

## 3.2 Processor Architecture

As shown by the block diagram on the front page, the MCP is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numerics instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for the Intel386 DX CPU bus tracking and interface. The BCL is the only unit in the Intel387 DX MCP that must run synchronously with the Intel386 DX CPU; the rest of the MCP can run asynchronously with respect to the Intel386 DX Microprocessor.

### 3.2.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the MCP and transferring outputs from the MCP to memory.

### 3.2.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, or FSTCW, the control executes it independently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY#, PEREQ and ERROR# signals that synchronize Intel387 DX MCP activities with the Intel386 DX CPU. It also supports the FPU in all operations that it cannot perform alone (e.g. exceptions handling, transcendental operations, etc.).

### 3.2.3 FLOATING POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

## 3.3 System Configuration

As an extension to the Intel386 DX Microprocessor, the Intel387 DX Math Coprocessor can be connected to the CPU as shown by Figure 3.3. A dedicated

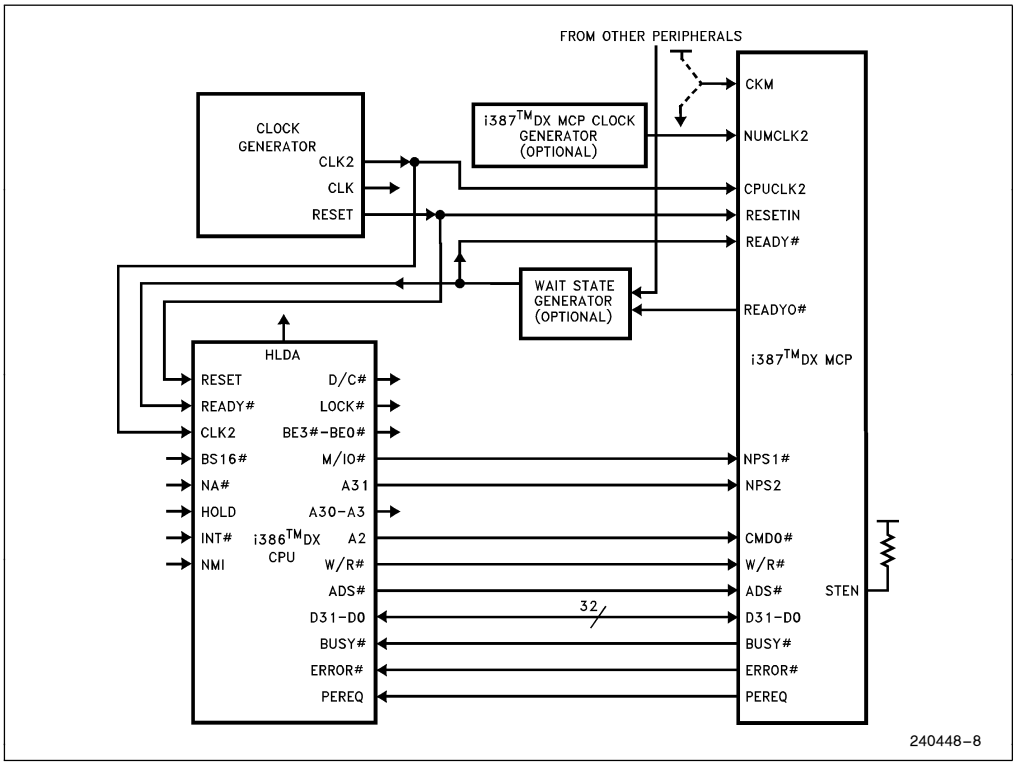



**Figure 3.3. Intel386™ DX Microprocessor and Intel387™ DX Math Coprocessor System Configuration**

**Table 3.4. Bus Cycles Definition**

| STEN | NPS1# | NPS2 | CMD0# | W/R# | Bus Cycle Type |
|---|---|---|---|---|---|
| 0 | x | x | x | x | MCP not selected and all outputs in floating state |
| 1 | 1 | x | x | x | MCP not selected |
| 1 | x | 0 | x | x | MCP not selected |
| 1 | 0 | 1 | 0 | 0 | CW or SW read from MCP |
| 1 | 0 | 1 | 0 | 1 | Opcode write to MCP |
| 1 | 0 | 1 | 1 | 0 | Data read from MCP |
| 1 | 0 | 1 | 1 | 1 | Data write to MCP |

communication protocol makes possible high-speed transfer of opcodes and operands between the Intel386 DX CPU and Intel387 DX MCP. The Intel387 DX MCP is designed so that no additional components are required for interface with the Intel386 DX CPU. The Intel387 DX MCP shares the 32-bit wide local bus of the Intel386 DX CPU and most control pins of the Intel387 DX MCP are connected directly to pins of the Intel386 DX Microprocessor.

### 3.3.1 BUS CYCLE TRACKING

The ADS# and READY# signals allow the MCP to track the beginning and end of the Intel386 DX CPU bus cycles, respectively. When ADS# is asserted at the same time as the MCP chip-select inputs, the bus cycle is intended for the MCP. To signal the end of a bus cycle for the MCP, READY# may be asserted directly or indirectly by the MCP or by other bus-control logic. Refer to Table 3.4 for definition of the types of MCP bus cycles.

### 3.3.2 MCP ADDRESSING

The NPS1#, NPS2 and STEN signals allow the MCP to identify which bus cycles are intended for the MCP. The MCP responds only to I/O cycles when bit 31 of the I/O address is set. In other words, the MCP acts as an I/O device in a reserved I/O address space.

Because $A_{31}$ is used to select the MCP for data transfers, it is not possible for a program running on the Intel386 DX CPU to address the MCP with an I/O instruction. Only ESC instructions cause the Intel386 DX Microprocessor to communicate with the MCP. The Intel386 DX CPU BS16# input must be inactive during I/O cycles when $A_{31}$ is active.

### 3.3.3 FUNCTION SELECT

The CMD0# and W/R# signals identify the four kinds of bus cycle: control or status register read, data read, opcode write, data write.

### 3.3.4 CPU/MCP Synchronization

The pin pairs BUSY#, PEREQ, and ERROR# are used for various aspects of synchronization between the CPU and the MCP.

BUSY# is used to synchronize instruction transfer from the Intel386 DX CPU to the MCP. When the MCP recognizes an ESC instruction, it asserts BUSY#. For most ESC instructions, the Intel386 DX CPU waits for the MCP to deassert BUSY# before sending the new opcode.

The MCP uses the PEREQ pin of the Intel386 DX CPU to signal that the MCP is ready for data transfer to or from its data FIFO. The MCP does not directly access memory; rather, the Intel386 DX Microprocessor provides memory access services for the MCP. Thus, memory access on behalf of the MCP always obeys the rules applicable to the mode of the Intel386 DX CPU, whether the Intel386 DX CPU be in real-address mode or protected mode.

Once the Intel386 DX CPU initiates an MCP instruction that has operands, the Intel386 DX CPU waits for PEREQ signals that indicate when the MCP is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the Intel386 DX CPU continues program execution while the MCP executes the ESC instruction.

In 8086/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In 80286/80287, Intel386 DX Microprocessor and Intel387 DX Math Coprocessor systems, WAIT instructions are required only for operand synchronization; namely, after MCP stores to memory (except FSTSW and FSTCW) or loads from memory. Used this way, WAIT ensures that the value has already been written or read by the MCP before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the Intel386 DX CPU, the MCP can process the instruction in parallel with and independent of the host CPU. When the MCP detects an exception, it asserts the ERROR# signal, which causes a Intel386 DX CPU interrupt.

#### 3.3.5 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the Intel387 DX MCP (the FPU) can either operate directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the CKM pin. In either case, the bus control logic (BCL) of the MCP is synchronized with the CPU clock. Use of asynchronous mode allows the Intel386 DX CPU and the FPU section of the MCP to run at different speeds. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design.

#### 3.3.6 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, READYO# can be used to drive the Intel386 DX CPU READY# input. If this pin is used, it should be connected to the logic that ORs all READY outputs from peripherals on the Intel386 DX CPU bus. READYO# is asserted by the MCP only during I/O cycles that select the MCP. Refer to section 3.4 "Bus Operation" for details.

## 3.4 Bus Operation

With respect to the bus interface, the Intel387 DX MCP is fully synchronous with the Intel386 DX Microprocessor. Both operate at the same rate, because each generates its internal CLK signal by dividing CPUCLK2 by two.

The Intel386 DX CPU initiates a new bus cycle by activating ADS#. The MCP recognizes a bus cycle, if, during the cycle in which ADS# is activated, STEN, NPS1#, and NPS2 are all activated. Proper operation is achieved if NPS1# is connected to the M/IO# output of the Intel386 DX CPU, and NPS2 to the A31 output. The Intel386 DX CPU's A31 output is guaranteed to be inactive in all bus cycles that do not address the MCP (i.e. I/O cycles to other devices, interrupt acknowledge, and reserved types of bus cycles). System logic must not signal a 16-bit bus cycle via the Intel386 DX CPU BS16# input during I/O cycles when A31 is active.

During the CLK period in which ADS# is activated, the MCP also examines the W/R# input signal to determine whether the cycle is a read or a write cycle and examines the CMD0# input to determine whether an opcode, operand, or control/status register transfer is to occur.

The Intel387 DX MCP supports both pipelined and nonpipelined bus cycles. A nonpipelined cycle is one for which the Intel386 DX CPU asserts ADS# when no other MCP bus cycle is in progress. A pipelined bus cycle is one for which the Intel386 DX CPU asserts ADS# and provides valid next-address and control signals as soon as in the second CLK period after the ADS# assertion for the previous Intel386 DX CPU bus cycle. Pipelining increases the availability of the bus by at least one CLK period. The MCP supports pipelined bus cycles in order to optimize address pipelining by the Intel386 DX CPU for memory cycles.

Bus operation is described in terms of an abstract *state machine*. Figure 3.4 illustrates the states and state transitions for MCP bus cycles:

- $T_I$ is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after evey nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- $T_{RS}$ is the READY# sensitive state. Different types of bus cycle may require a minimum of one or two successive $T_{RS}$ states. The bus logic remains in $T_{RS}$ state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive $T_{RS}$ states.
- $T_P$ is the first state for every pipelined bus cycle.

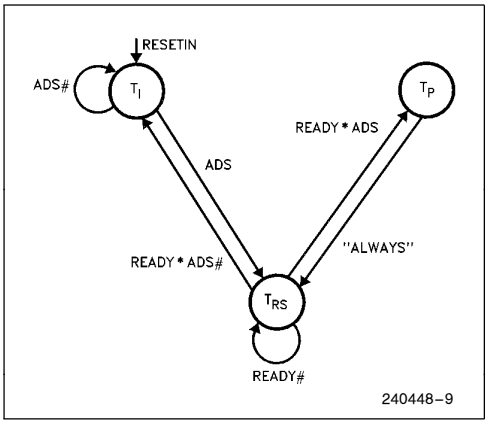



**Figure 3.4. Bus State Diagram**

The READYO# output of the Intel387 DX MCP indicates when a bus cycle for the MCP may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READYO# is always asserted in the first $T_{RS}$ state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and FRSTOR, READYO# is always asserted in the second $T_{RS}$ state, regardless of the number of wait states. These rules apply to both pipelined and nonpipelined cycles. Systems designers must use READYO# in one of the following ways:

1. Connect it (directly or through logic that ORs READY signals from other devices) to the READY# inputs of the Intel386 DX CPU and Intel387 DX MCP.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of MCP bus cycles.

Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus-cycle diagrams show memory cycles between MCP operand-transfer cycles. Note however that, during the instructions FLDENV, FSTENV, FSAVE, and FRSTOR, some consecutive accesses to the MCP do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see Figure 3.8.

### 3.4.1 NONPIPELINED BUS CYCLES

Figure 3.5 illustrates bus activity for consecutive nonpipelined bus cycles.

#### 3.4.1.1 Write Cycle

At the second clock of the bus cycle, the Intel387 DX MCP enters the $T_{RS}$ (READY#-sensitive) state. During this state, the Intel387 DX MCP samples the READY# input and stays in this state as long as READY# is inactive.

In write cycles, the MCP drives the READYO# signal for one CLK period beginning with the second CLK of the bus cycle; therefore, the fastest write cycle takes two CLK cycles (see cycle 2 of Figure 3.5). For the instructions FLDENV and FRSTOR, however, the MCP forces a wait state by delaying the activation of READYO# to the second $T_{RS}$ cycle (not shown in Figure 3.5).

When READY# is asserted the MCP returns to the idle state, in which ADS# could be asserted again by the Intel386 DX Microprocessor for the next cycle.

#### 3.4.1.2 Read Cycle

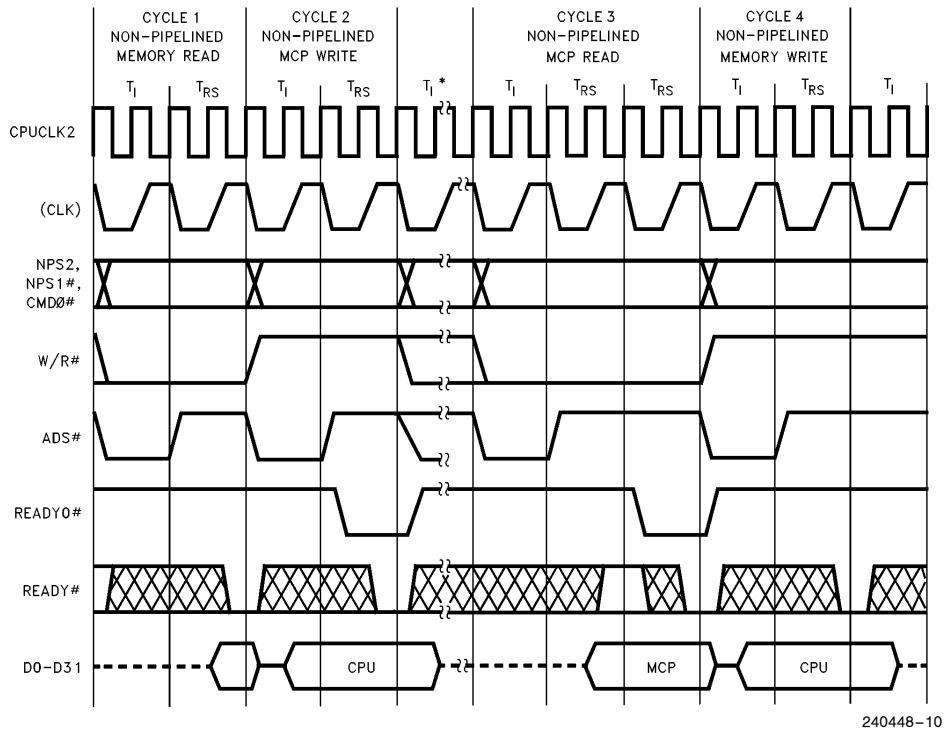
At the second clock of the bus cycle, the MCP enters the $T_{RS}$ state. See Figure 3.5. In this state, the MCP samples the READY# input and stays in this state as long as READY# is inactive.

At the rising edge of CLK in the second clock period of the cycle, the MCP starts to drive the D31–D0 outputs and continues to drive them as long as it stays in $T_{RS}$ state.

In read cycles that address the MCP, at least one wait state must be inserted to insure that the Intel386 DX CPU latches the correct data. Since the MCP starts driving the system data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the Intel386 DX CPU at the falling edge of the same clock period. The MCP drives the READYO# signal for one CLK period in the third CLK of the bus cycle. Therefore, if the READYO# output is used to drive the Intel386 DX CPU READY# input, one wait state is inserted automatically.

Because one wait state is required for MCP reads, the minimum is three CLK cycles per read, as cycle 3 of Figure 3.5 shows.

When READY# is asserted the MCP returns to the idle state, in which ADS# could be asserted again by the Intel386 DX CPU for the next cycle. The transition from $T_{RS}$ state to idle state causes the MCP to put the tristate D31–D0 outputs into the floating state, allowing another device to drive the system data bus.




Cycles 1 & 2 represent part of the operand transfer cycle for instructions involving either 4-byte or 8-byte operand loads.
Cycles 3 & 4 represent part of the operand transfer cycle for a store operation.
*Cycles 1 & 2 could repeat here or $T_I$ states for various non-operand transfer cycles and overhead.

**Figure 3.5. Nonpipelined Read and Write Cycles**

### 3.4.2 PIPELINED BUS CYCLES

Because all the activities of the Intel387 DX MCP bus interface occur either during the $T_{RS}$ state or during the transitions to or from that state, the only difference between a pipelined and a nonpipelined cycle is the manner of changing from one state to another. The exact activities in each state are detailed in the previous section ''Nonpipelined Bus Cycles''.

When the Intel386 DX CPU asserts ADS# before the end of a bus cycle, both ADS# and READY# are active during a $T_{RS}$ state. This condition causes the MCP to change to a different state named $T_P$. The MCP activities in the transition from a $T_{RS}$ state to a $T_P$ state are exactly the same as those in the transition from a $T_{RS}$ state to a $T_I$ state in nonpipelined cycles.

$T_P$ state is metastable; therefore, one clock period later the MCP returns to $T_{RS}$ state. In consecutive pipelined cycles, the MCP bus logic uses only $T_{RS}$ and $T_P$ states.

Figure 3.6 shows the fastest transition into and out of the pipelined bus cycles. Cycle 1 in this figure represents a nonpipelined cycle. (Nonpipelined write cycles with only one $T_{RS}$ state (i.e. no wait states) are always followed by another nonpipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

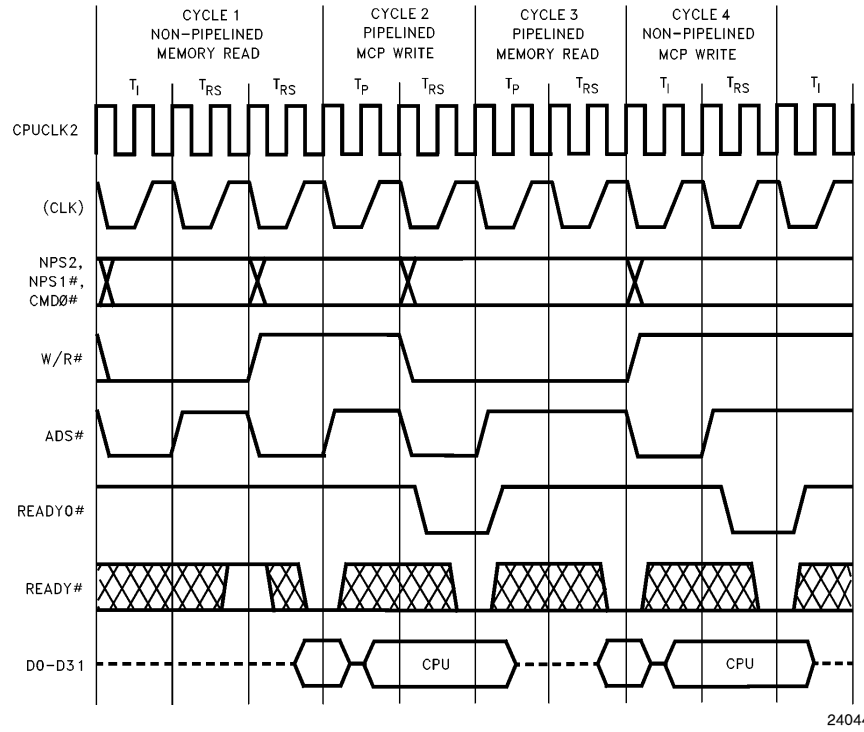
Figure 3.7 shows the pipelined write and read cycles with one additional $T_{RS}$ states beyond the minimum required. To delay the assertion of READY# requires external logic.

### 3.4.3 BUS CYCLES OF MIXED TYPE

When the Intel387 DX MCP bus logic is in the $T_{RS}$ state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of ADS# and READY#. In a nonpipelined cycle, only READY# is activated, and the transition is from $T_{RS}$ to idle state. In a pipelined cycle, both READY# and ADS# are active and the transition is first from $T_{RS}$ state to $T_P$ state then, after one clock period, back to $T_{RS}$ state.

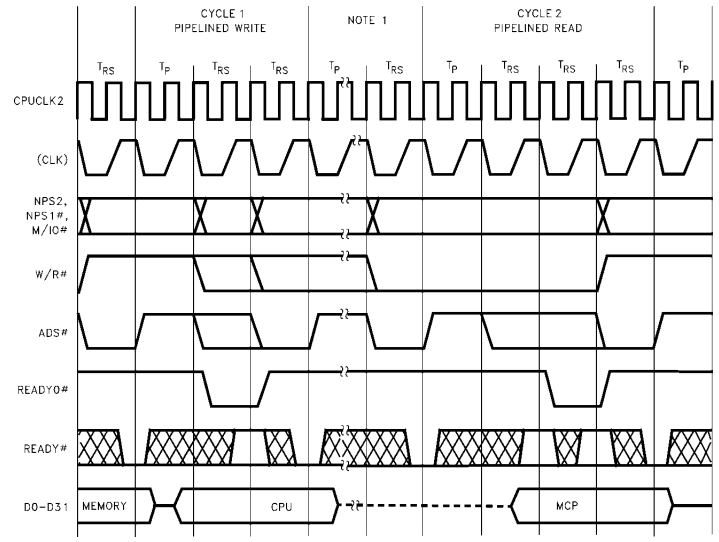
### 3.4.4 BUSY# AND PEREQ TIMING RELATIONSHIP

Figure 3.8 shows the activation of BUSY# at the beginning of instruction execution and its deactivation after execution of the instruction is complete. When possible, the Intel387 DX MCP may deactivate BUSY# prior to the completion of the current instruction allowing the CPU to transfer the next instruction's opcode and operands. PEREQ is activated in this interval. If ERROR# (not shown in the diagram) is ever asserted, it would occur at least six CPUCLK2 periods after the deactivation of PEREQ and at least six CPUCLK2 periods before the deactivation of BUSY#. Figure 3.8 shows also that STEN is activated at the beginning of a bus cycle.



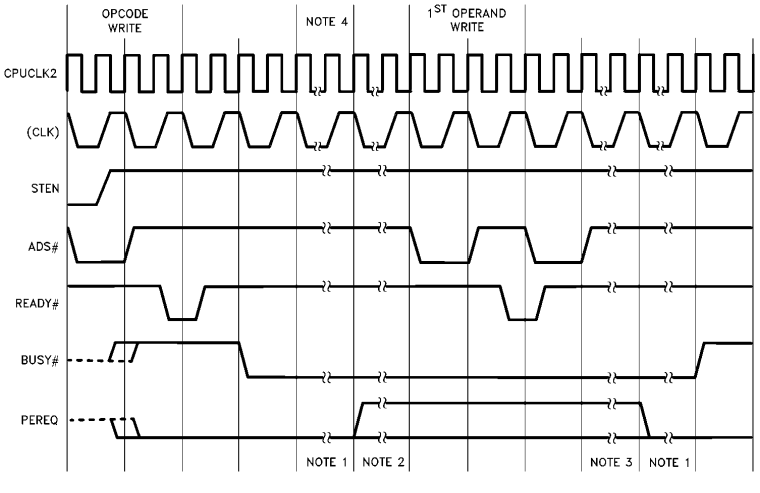

Cycle 1–Cycle 4 represent the operand transfer cycle for an instruction involving a transfer of two 32-bit loads in total. The opcode write cycles and other overhead are not shown.
Note that the next cycle will be a pipelined cycle if both READY# and ADS# are sampled active at the end of a $T_{RS}$ state of the current cycle.

**Figure 3.6. Fastest Transitions to and from Pipelined Cycles**




**NOTE:**
1. Cycles between operand write to the MCP and storing result.

**Figure 3.7. Pipelined Cycles with Wait States**




**NOTES:**
1. Instruction dependent.
2. PEREQ is an asynchronous input to the Intel386™ DX Microprocessor; it may not be asserted (instruction dependent).
3. More operand transfers.
4. Memory read (operand) cycle is not shown.

**Figure 3.8. STEN, BUSY # and PEREQ Timing Relationship**

## 4.0 ELECTRICAL DATA

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

### 4.1 Absolute Maximum Ratings*

Case Temperature $T_C$
Under Bias . . . . . −65°C to +110°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to Ground . . . . . −0.5 to $V_{CC}$ +0.5V
Power Dissipation . . . . . 1.5W

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

### 4.2 DC Characteristics

**Table 4.1. DC Specifications** $T_C$ = 0° to 85°C, $V_{CC}$ = 5V ±5%

| Symbol | Parameter | Min | Max | Units | Test Conditions |
|---|---|---|---|---|---|
| $V_{IL}$ | Input LO Voltage | −0.3 | +0.8 | V | (Note 1) |
| $V_{IH}$ | Input HI Voltage | 2.0 | $V_{CC}$ + 0.3 | V | (Note 1) |
| $V_{CL}$ | CPUCLK2 Input LO Voltage | −0.3 | +0.8 | V | |
| $V_{CH}$ | CPUCLK2 Input HI Voltage | 3.7 | $V_{CC}$ +0.3 | V | |
| $V_{OL}$ | Output LO Voltage | | 0.45 | V | (Note 2) |
| $V_{OH}$ | Output HI Voltage | 2.4 | | V | (Note 3) |
| $I_{CC}$ | Supply Current | | | | |
| | NUMCLK2 = 32 MHz(4) | | 160 | mA | $I_{CC}$ typ. = 95 mA |
| | NUMCLK2 = 40 MHz(4) | | 180 | mA | $I_{CC}$ typ. = 105 mA |
| | NUMCLK2 = 50 MHz(4) | | 210 | mA | $I_{CC}$ typ. = 125 mA |
| | NUMCLK2 = 66.6 MHz(4) | | 250 | mA | $I_{CC}$ typ. = 150 mA |
| $I_{LI}$ | Input Leakage Current | | ±15 | μA | $0V \le V_{IN} \le V_{CC}$ |
| $I_{LO}$ | I/O Leakage Current | | ±15 | μA | $0.45V \le V_O \le V_{CC}$ |
| $C_{IN}$ | Input Capacitance | | 10 | pF | fc = 1 MHz |
| $C_O$ | I/O or Output Capacitance | | 12 | pF | fc = 1 MHz |
| $C_{CLK}$ | Clock Capacitance | | 15 | pF | fc = 1 MHz |

**NOTES:**
1. This parameter is for all inputs, including NUMCLK2 but excluding CPUCLK2.
2. This parameter is measured at $I_{OL}$ as follows:
   data = 4.0 mA
   READYO# = 2.5 mA
   ERROR#, BUSY#, PEREQ = 2.5 mA
3. This parameter is measured at $I_{OH}$ as follows:
   data = 1.0 mA
   READYO# = 0.6 mA
   ERROR#, BUSY#, PEREQ = 0.6 mA
4. $I_{CC}$ is measured at steady state, maximum capacitive loading on the outputs, CPUCLK2 at the same frequency as NUMCLK2.

## 4.3 AC Characteristics

**Table 4.2a. i387 DX/i386 DX Interface and Execution Frequencies**

| i386 DX System Frequency (MHz) | i387 DX 16-33 Execution Frequency (MHz) | |
|---|---|---|
| | Min | Max |
| **16 MHz** | 10.0 MHz | 22.4 MHz |
| **20 MHz** | 12.5 MHz | 28.0 MHz |
| **25 MHz** | 15.6 MHz | 33.0 MHz |
| **33 MHz** | 20.6 MHz | 33.0 MHz |

**NOTE:**
The external clock frequencies for the i387 DX and i386 DX are equal to twice the interface and execution frequencies show above.

**Table 4.2b. Timing Requirements of the Execution Unit**
**$T_C$ = 0°C to +85°C, $V_{CC}$ = 5V ±5%**

| Pin | Symbol | Parameter | 16 MHz–33 MHz | | Test Conditions | Figure Reference |
|---|---|---|---|---|---|---|
| | | | Min (ns) | Max (ns) | | |
| NUMCLK2 | t1 | Period | 15 | 125 | 2.0V | 4.1 |
| NUMCLK2 | t2a | High Time | 6.25 | | 2.0V | |
| NUMCLK2 | t2b | High Time | 4.5 | | 3.7V | |
| NUMCLK2 | t3a | Low Time | 6.25 | | 2.0V | |
| NUMCLK2 | t3b | Low Time | 4.5 | | 0.8V | |
| NUMCLK2 | t4 | Fall Time | | 6 | 3.7V to 0.8V | |
| NUMCLK2 | t5 | Rise Time | | 6 | 0.8V to 2.7V | |

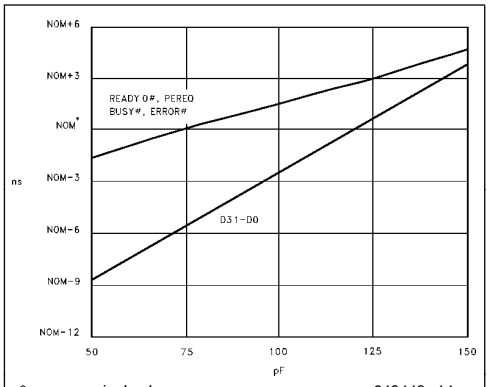
**Table 4.2c. Timing Requirements of the Bus Interface Unit**
**$T_C$ = 0°C to +85°C, $V_{CC}$ = 5V ±5%**
**(All measurements made at 1.5V and 50 pF unless otherwise specified)**

| Pin | Symbol | Parameter | 16 MHz–33 MHz | | Test Conditions | Figure Reference |
|---|---|---|---|---|---|---|
| | | | Min (ns) | Max (ns) | | |
| CPUCLK2 | t1 | Period | 15 | 125 | 2.0V | 4.1 |
| CPUCLK2 | t2a | High Time | 6.25 | | 2.0V | |
| CPUCLK2 | t2b | High Time | 4.5 | | 3.7V | |
| CPUCLK2 | t3a | Low Time | 6.25 | | 2.0V | |
| CPUCLK2 | t3b | Low Time | 4.5 | | 0.8V | |
| CPUCLK2 | t4 | Fall Time | | 6 | 3.7V to 0.8V | |
| CPUCLK2 | t5 | Rise Time | | 6 | 0.8V to 3.7V | |
| NUMCLK2/ CPUCLK2 | | Ratio | 10/16 | 14/10 | | |
| READYO# | t7 | Out Delay | 4 | 17 | | 4.2 |
| READYO#(1) | t7 | Out Delay | 4 | 15 | $C_L$ = 25 pF | |
| PEREQ | t7 | Out Delay | 4 | 25 | | |
| BUSY# | t7 | Out Delay | 4 | 21 | | |
| BUSY#(1) | t7 | Out Delay | 4 | 19 | $C_L$ = 25 pF | |
| ERROR# | t7 | Out Delay | 4 | 25 | | |
| D31–D0 | t8 | Out Delay | 0 | 37 | | 4.3 |
| D31–D0 | t10 | Setup Time | 8 | | | |
| D31–D0 | t11 | Hold Time | 8 | | | |
| D31–D0(2) | t12 | Float Time | 3 | 19 | | |

**Table 4.2c. Timing Requirements of the Bus Interface Unit** (Continued)
**$T_C$ = 0°C to +85°C, $V_{CC}$ = 5V ±5%**
**(All measurements made at 1.5V and 50 pF unless otherwise specified)**

| Pin | Symbol | Parameter | 16 MHz–33 MHz | | Test Conditions | Figure Reference |
|---|---|---|---|---|---|---|
| | | | Min (ns) | Max (ns) | | |
| PEREQ(2) | t13 | Float Time | 1 | 30 | | 4.5 |
| BUSY#(2) | t13 | Float Time | 1 | 30 | | |
| ERROR#(2) | t13 | Float Time | 1 | 30 | | |
| READYO#(2) | t13 | Float Time | 1 | 30 | | |
| ADS# | t14 | Setup Time | 13 | | | 4.3 |
| ADS# | t15 | Hold Time | 4 | | | |
| W/R# | t14 | Setup Time | 13 | | | |
| W/R# | t15 | Hold Time | 4 | | | |
| READY# | t16 | Setup Time | 7 | | | |
| READY# | t17 | Hold Time | 4 | | | |
| CMDO# | t16 | Setup Time | 13 | | | |
| CMDO# | t17 | Hold Time | 2 | | | |
| NPS1# | t16 | Setup Time | 13 | | | |
| NPS2 | | | | | | |
| NPS1# | t17 | Hold Time | 2 | | | |
| NPS2 | | | | | | |
| STEN | t16 | Setup Time | 13 | | | |
| STEN | t17 | Hold Time | 2 | | | |
| RESETIN | t18 | Setup Time | 5 | | | 4.4 |
| RESETIN | t19 | Hold Time | 3 | | | |

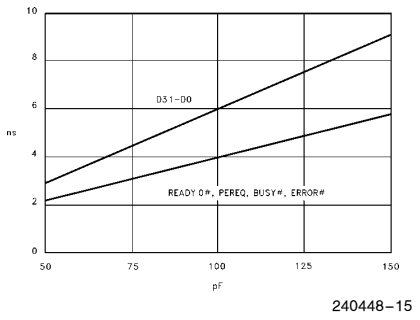
**NOTES:**
1. Not tested at 25 pF.
2. Float delay is not tested. Float condition occurs when maximum output current becomes less than $I_{LO}$ in magnitude.




*nom - nominal value 240448–14

**NOTE:**
This graph will not be linear outside of the $C_L$ range shown.

**Figure 4.0a. Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature**




240448–15

**NOTE:**
This graph will not be linear outside of the $C_L$ range shown.

**Figure 4.0b. Typical Output Rise Time vs Load Capacitance at Max Operating Temperature**

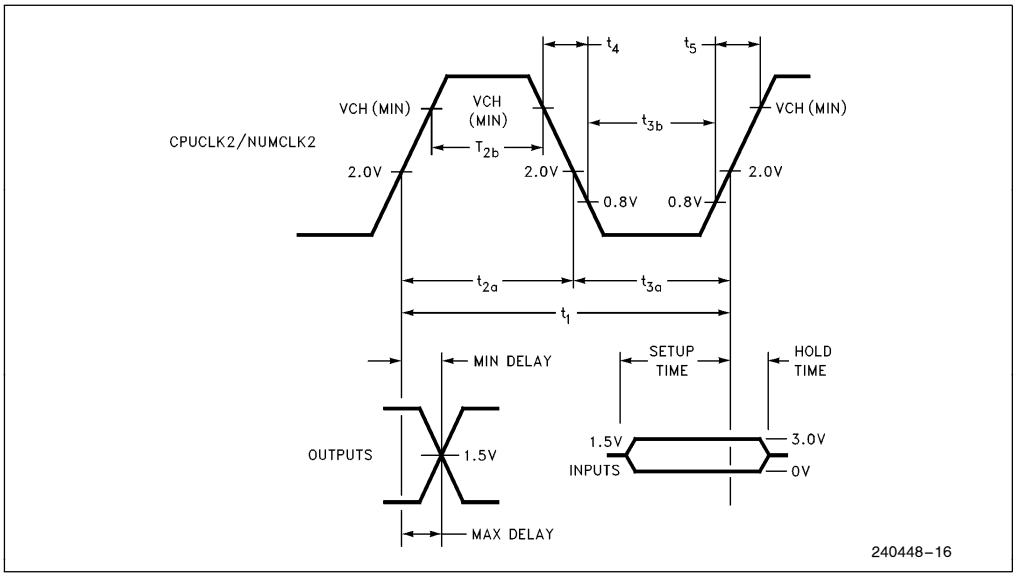



**Figure 4.1. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output A.C. Specifications**

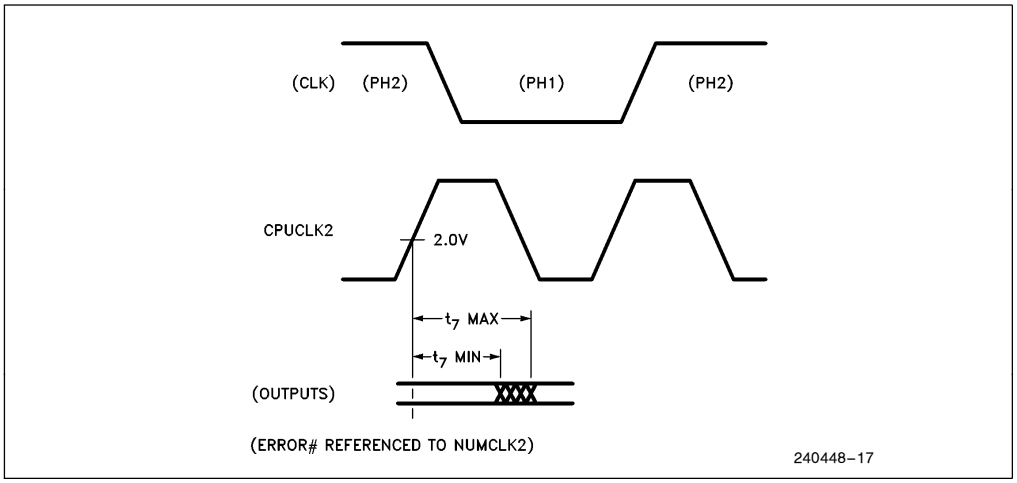



**Figure 4.2. Output Signals**

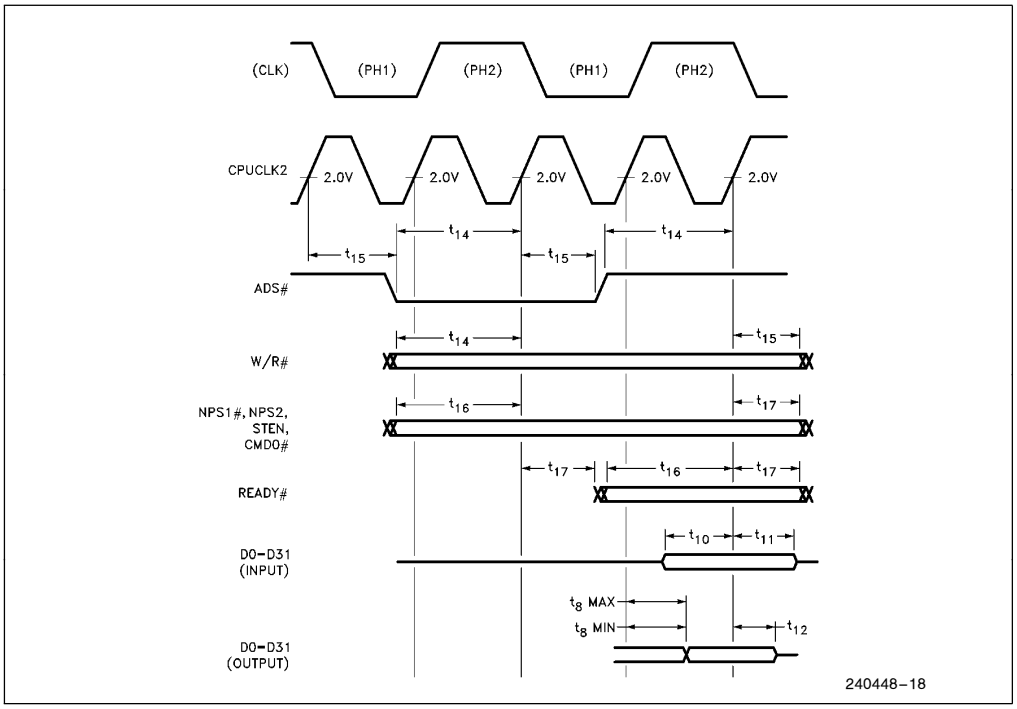
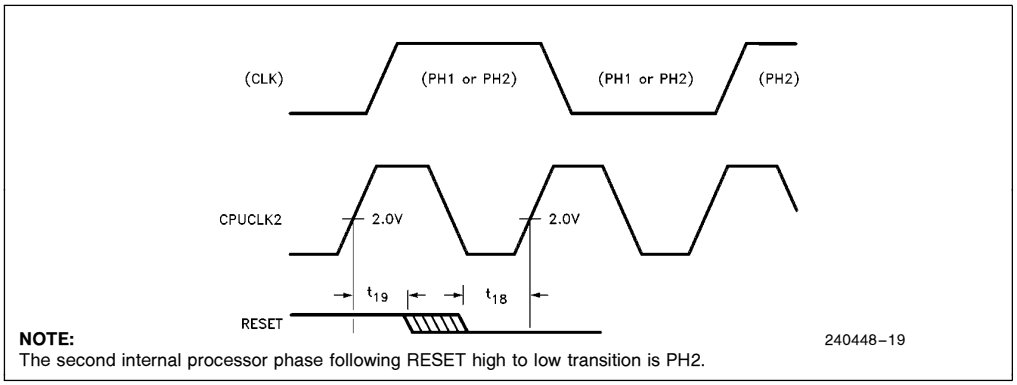



**Figure 4.3. Input and I/O Signals**




**NOTE:**
The second internal processor phase following RESET high to low transition is PH2.

**Figure 4.4. RESET Signal**

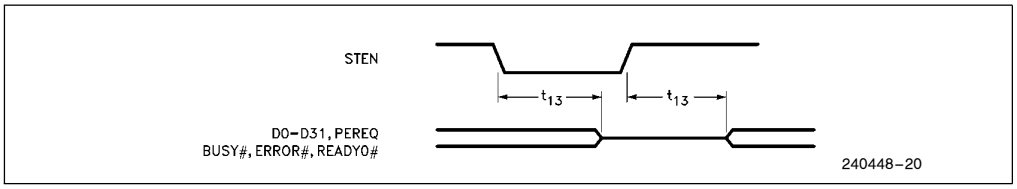
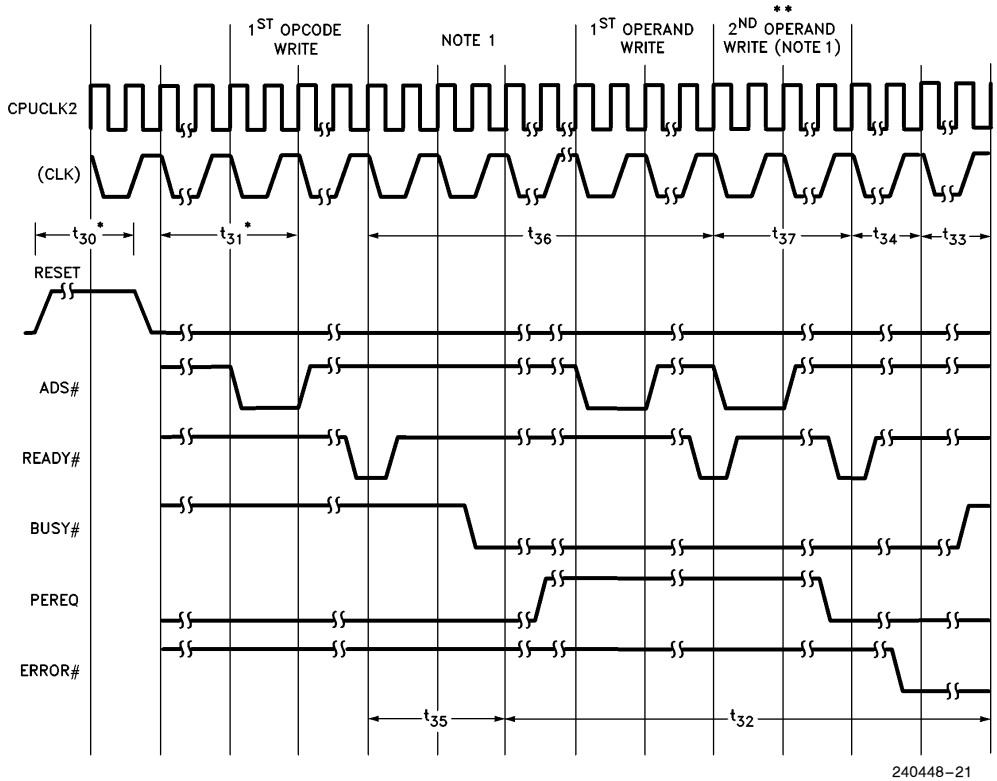



**Figure 4.5. Float from STEN**

**Table 4.3. Other Parameters**

| Pin | Symbol | Parameter | Min | Max | Units |
|---|---|---|---|---|---|
| RESETIN | t30 | Duration | 40 | | NUMCLK2 |
| RESETIN | t31 | RESETIN Inactive to 1st Opcode Write | 50 | | NUMCLK2 |
| BUSY# | t32 | Duration | 6 | | CPUCLK2 |
| BUSY#, ERROR# | t33 | ERROR# (In) Active to BUSY# Inactive | 6 | | CPUCLK2 |
| PEREQ, ERROR# | t34 | PEREQ Inactive to ERROR# Active | 6 | | CPUCLK2 |
| READY#, BUSY# | t35 | READY# Active to BUSY# Active | 4 | 4 | CPUCLK2 |
| READY# | t36 | Minimum Time from Opcode Write to Opcode/Operand Write | 6 | | CPUCLK2 |
| READY# | t37 | Minimum Time from Operand Write to Operand Write | 8 | | CPUCLK2 |




* In NUMCLK2's
** or last operand

**NOTE:**
1. Memory read (operand) cycle is not shown.

**Figure 4.6. Other Parameters**

| | Instruction | | | | | | | | Optional Fields | |
|---|---|---|---|---|---|---|---|---|---|---|
| | First Byte | | | | Second Byte | | | | | |
| 1 | 11011 | OPA | | 1 | MOD | 1 | OPB | R/M | SIB | DISP |
| 2 | 11011 | MF | | OPA | MOD | OPB | | R/M | SIB | DISP |
| 3 | 11011 | d | P | OPA | 1 | 1 | OPB | ST(i) | | |
| 4 | 11011 | 0 | 0 | 1 | 1 | 1 | 1 | OP | | |
| 5 | 11011 | 0 | 1 | 1 | 1 | 1 | 1 | OP | | |
| | 15–11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 3 2 1 0 | | |

## 5.0 Intel387™ DX MCP EXTENSIONS TO THE Intel386™ DX CPU INSTRUCTION SET

Instructions for the Intel387 DX MCP assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the Intel386 DX CPU addressing modes.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

ESC = 11011

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of the Intel386 DX Microprocessor instructions (refer to *Intel386™ DX Microprocessor Programmer's Reference Manual*).

SIB (Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for Intel386 DX Microprocessor instructions.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD request delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

## Intel387™ DX MCP Extensions to the Intel386™ DX CPU Instruction Set

| Instruction | Encoding | | | Clock Count Range | | | |
|---|---|---|---|---|---|---|---|
| | Byte 0 | Byte 1 | Optional Bytes 2–6 | 32-Bit Real | 32-Bit Integer | 64-Bit Real | 16-Bit Integer |
| **DATA TRANSFER** | | | | | | | |
| **FLD** = Load[a] | | | | | | | |
| Integer/real memory to ST(0) | ESC MF 1 | MOD 000 R/M | SIB/DISP | 9–18 | 26–42 | 16–23 | 42–53 |
| Long integer memory to ST(0) | ESC 111 | MOD 101 R/M | SIB/DISP | | 26–54 | | |
| Extended real memory to ST(0) | ESC 011 | MOD 101 R/M | SIB/DISP | | 12–43 | | |
| BCD memory to ST(0) | ESC 111 | MOD 100 R/M | SIB/DISP | | 45–97 | | |
| ST(i) to ST(0) | ESC 001 | 11000 ST(i) | | | 7–12 | | |
| **FST** = Store | | | | | | | |
| ST(0) to integer/real memory | ESC MF 1 | MOD 010 R/M | SIB/DISP | 25–43 | 57–76 | 32–44 | 58–76 |
| ST(0) to ST(i) | ESC 101 | 11010 ST(i) | | | 7–11 | | |
| **FSTP** = Store and Pop | | | | | | | |
| ST(0) to integer/real memory | ESC MF 1 | MOD 011 R/M | SIB/DISP | 25–43 | 57–76 | 32–44 | 58–76 |
| ST(0) to long integer memory | ESC 111 | MOD 111 R/M | SIB/DISP | | 60–82 | | |
| ST(0) to extended real | ESC 011 | MOD 111 R/M | SIB/DISP | | 46–52 | | |
| ST(0) to BCD memory | ESC 111 | MOD 110 R/M | SIB/DISP | | 112–190 | | |
| ST(0) to ST(i) | ESC 101 | 11011 ST (i) | | | 7–11 | | |
| **FXCH** = Exchange | | | | | | | |
| ST(i) and ST(0) | ESC 001 | 11001 ST(i) | | | 10–17 | | |
| **COMPARISON** | | | | | | | |
| **FCOM** = Compare | | | | | | | |
| Integer/real memory to ST(0) | ESC MF 0 | MOD 010 R/M | SIB/DISP | 13–25 | 34–52 | 14–27 | 39–62 |
| ST(i) to ST(0) | ESC 000 | 11010 ST(i) | | | 13–21 | | |
| **FCOMP** = Compare and pop | | | | | | | |
| Integer/real memory to ST | ESC MF 0 | MOD 011 R/M | SIB/DISP | 13–25 | 34–52 | 14–27 | 39–62 |
| ST(i) to ST(0) | ESC 000 | 11011 ST(i) | | | 13–21 | | |
| **FCOMPP** = Compare and pop twice | | | | | | | |
| ST(1) to ST(0) | ESC 110 | 1101 1001 | | | 13–21 | | |
| **FTST** = Test ST(0) | ESC 001 | 1110 0100 | | | 17–25 | | |
| **FUCOM** = Unordered compare | ESC 101 | 11100 ST(i) | | | 13–21 | | |
| **FUCOMP** = Unordered compare and pop | ESC 101 | 11101 ST(i) | | | 13–21 | | |
| **FUCOMPP** = Unordered compare and pop twice | ESC 010 | 1110 1001 | | | 13–21 | | |
| **FXAM** = Examine ST(0) | ESC 001 | 11100101 | | | 24–37 | | |
| **CONSTANTS** | | | | | | | |
| **FLDZ** = Load +0.0 into ST(0) | ESC 001 | 1110 1110 | | | 10–17 | | |
| **FLD1** = Load +1.0 into ST(0) | ESC 001 | 1110 1000 | | | 15–22 | | |
| **FLDPI** = Load pi into ST(0) | ESC 001 | 1110 1011 | | | 26–36 | | |
| **FLDL2T** = Load $\log_2(10)$ into ST(0) | ESC 001 | 1110 1001 | | | 26–36 | | |

Shaded areas indicate instructions not available in 8087/80287.

**NOTE:**
a. When loading single- or double-precision zero from memory, add 5 clocks.

**Intel387™ DX MCP Extensions to the Intel386™ DX CPU Instruction Set** (Continued)

| Instruction | Encoding | | | Clock Count Range | | | |
|---|---|---|---|---|---|---|---|
| | Byte 0 | Byte 1 | Optional Bytes 2–6 | 32-Bit Real | 32-Bit Integer | 64-Bit Real | 16-Bit Integer |
| **CONSTANTS** (Continued) | | | | | | | |
| **FLDL2E** = Load $\log_2(e)$ into ST(0) | ESC 001 | 1110 1010 | | | 26–36 | | |
| **FLDLG2** = Load $\log_{10}(2)$ into ST(0) | ESC 001 | 1110 1100 | | | 25–35 | | |
| **FLDLN2** = Load $\log_e(2)$ into ST(0) | ESC 001 | 1110 1101 | | | 26–38 | | |
| **ARITHMETIC** | | | | | | | |
| **FADD** = Add | | | | | | | |
| Integer/real memory with ST(0) | ESC MF 0 | MOD 000 R/M | SIB/DISP | 12–29 | 34–56 | 15–34 | 38–64 |
| ST(i) and ST(0) | ESC d P 0 | 11000 ST(i) | | | 12–26[b] | | |
| **FSUB** = Subtract | | | | | | | |
| Integer/real memory with ST(0) | ESC MF 0 | MOD 10 R R/M | SIB/DISP | 12–29 | 34–56 | 15–34 | 38–64[c] |
| ST(i) and ST(0) | ESC d P 0 | 1110 R R/M | | | 12–26[d] | | |
| **FMUL** = Multiply | | | | | | | |
| Integer/real memory with ST(0) | ESC MF 0 | MOD 001 R/M | SIB/DISP | 19–32 | 43–71 | 23–53 | 46–74 |
| ST(i) and ST(0) | ESC d P 0 | 1100 1 R/M | | | 17–50[e] | | |
| **FDIV** = Divide | | | | | | | |
| Integer/real memory with ST(0) | ESC MF 0 | MOD 11 R R/M | SIB/DISP | 77-85 | 101–114[f] | 81–91 | 105–124[g] |
| ST(i) and ST(0) | ESC d P 0 | 1111 R R/M | | | 77-80[h] | | |
| **FSQRT**[i] = Square root | ESC 001 | 1111 1010 | | | 97–111 | | |
| **FSCALE** = Scale ST(0) by ST(1) | ESC 001 | 1111 1101 | | | 44–82 | | |
| **FPREM** = Partial remainder | ESC 001 | 1111 1000 | | | 56–140 | | |
| **FPREM1** = Partial remainder (IEEE) | ESC 001 | 1111 0101 | | | 81–168 | | |
| **FRNDINT** = Round ST(0) to integer | ESC 001 | 1111 1100 | | | 41–62 | | |
| **FXTRACT** = Extract components of ST(0) | ESC 001 | 1111 0100 | | | 42–63 | | |
| **FABS** = Absolute value of ST(0) | ESC 001 | 1110 0001 | | | 14–21 | | |
| **FCHS** = Change sign of ST(0) | ESC 001 | 1110 0000 | | | 17–24 | | |

Shaded areas indicate instructions not available in 8087/80287.

**NOTES:**
b. Add 3 clocks to the range when d = 1.
c. Add 1 clock to **each** range when R = 1.
d. Add 3 clocks to the range when d = 0.
e. typical = 52 (When d = 0, 46–54, typical = 49).
f. Add 1 clock to the range when R = 1.
g. 135–141 when R = 1.
h. Add 3 clocks to the range when d = 1.
i. $-0 \leq ST(0) \leq +\infty$.

**Intel387™ DX MCP Extensions to the Intel386™ DX CPU Instruction Set** (Continued)

| Instruction | Encoding | | | Clock Count Range |
|---|---|---|---|---|
| | Byte 0 | Byte 1 | Optional Bytes 2–6 | |
| **TRANSCENDENTAL** | | | | |
| **FCOS**[k] = Cosine of ST(0) | ESC 001 | 1111 1111 | | 122–680 |
| **FPTAN**[k] = Partial tangent of ST(0) | ESC 001 | 1111 0010 | | 162–430[j] |
| **FPATAN** = Partial arctangent | ESC 001 | 1111 0011 | | 250–420 |
| **FSIN**[k] = Sine of ST(0) | ESC 001 | 1111 1110 | | 121–680 |
| **FSINCOS**[k] = Sine and cosine of ST(0) | ESC 001 | 1111 1011 | | 150–650 |
| **F2XM1**[l] = $2^{ST(0)} - 1$ | ESC 001 | 1111 0000 | | 167–410 |
| **FYL2X**[m] = ST(1) * $\log_2(ST(0))$ | ESC 001 | 1111 0001 | | 99–436 |
| **FYL2XP1**[n] = ST(1) * $\log_2(ST(0) + 1.0)$ | ESC 001 | 1111 1001 | | 210–447 |
| **PROCESSOR CONTROL** | | | | |
| **FINIT** = Initialize MCP | ESC 011 | 1110 0011 | | 33 |
| **FSTSW AX** = Store status word | ESC 111 | 1110 0000 | | 13 |
| **FLDCW** = Load control word | ESC 001 | MOD 101 R/M | SIB/DISP | 19 |
| **FSTCW** = Store control word | ESC 101 | MOD 111 R/M | SIB/DISP | 15 |
| **FSTSW** = Store status word | ESC 101 | MOD 111 R/M | SIB/DISP | 15 |
| **FCLEX** = Clear exceptions | ESC 011 | 1110 0010 | | 11 |
| **FSTENV** = Store environment | ESC 001 | MOD 110 R/M | SIB/DISP | 103–104 |
| **FLDENV** = Load environment | ESC 001 | MOD 100 R/M | SIB/DISP | 71 |
| **FSAVE** = Save state | ESC 101 | MOD 110 R/M | SIB/DISP | 375–376 |
| **FRSTOR** = Restore state | ESC 101 | MOD 100 R/M | SIB/DISP | 308 |
| **FINCSTP** = Increment stack pointer | ESC 001 | 1111 0111 | | 21 |
| **FDECSTP** = Decrement stack pointer | ESC 001 | 1111 0110 | | 22 |
| **FFREE** = Free ST(i) | ESC 101 | 1100 0 ST(i) | | 18 |
| **FNOP** = No operations | ESC 001 | 1101 0000 | | 12 |

Shaded areas indicate instructions not available in 8087/80287.

**NOTES:**
j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.
k. $0 \leq |ST(0)| < 2^{63}$.
l. $-1.0 \leq ST(0) \leq 1.0$.
m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.
n. $0 \leq |ST(0)| < (2 - SQRT(2))/2$, $-\infty < ST(1) < +\infty$.



39

# APPENDIX A
# COMPATIBILITY BETWEEN THE 80287 AND THE 8087

The 80286/80287 operating in Real-Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 MCP and the 8087 MCP, exception-handling routines *may* need to be changed.

This appendix summarizes the differences between the 80287 MCP and the 8087 MCP, and provides details showing how 8086/8087 programs can be ported to the 80286/80287.

1. The MCP signals exceptions through a dedicated ERROR# line to the 80286. The MCP error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored—none of the 80287 internal states will be updated. While 8086/8087 containing these instructions may be executed on the 80286/80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected-Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode—exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating-point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU—the 80286 automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instructions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

   The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instruction with a CPU WAIT instruction, in the identical manner as does ASM86.

## DATA SHEET REVISION REVIEW

The following list represents the key differences between this and the -003 versions of the Intel387™ Math Coprocessor Data Sheet. Please review this summary carefully.

1. Corrected typographical errors.
2. Corrected clock ratio "PIN" name on Table 4.2c to NUMCLK/CPUCLK.