MILITARY i387™ SX MATH COPROCESSOR

Military

- Interfaces with Military i386™ SX Microprocessor
- Expands Military i386 SX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands
- High Performance 80-Bit Internal Architecture
- Two to Three Times M8087/M80287 Performance at Equivalent Clock Speed
- Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic
- Fully compatible with the Military i387™ Math Coprocessor. Implements all Military i387 NPX architectural enhancements over M8087 and M80287.
- Upward Object-Code Compatible from M8087 and M80287
- Directly Extends Military i386 SX CPU Instruction Set to Trigonometric, Logarithmic, Exponential, and Arithmetic Instructions for All Data Types
- Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM
- Operates Independently of Real, Protected, and Virtual-8086 Modes of the Military i386 SX Microprocessor
- Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack
- Available in a 68-Lead PGA Package
  (see Packaging Specs: Order #231369)
- Available in Three Product Grades:
  — MIL-STD-883, −55°C to +125°C (T_C)
  — Military Temperature Only, −55°C to +125°C (T_C)
  — Extended Temperature, −40°C to +110°C (T_C)

The Intel Military i387 SX Math Coprocessor is an extension to the Intel Military i386 microprocessor architecture. The combination of the Military i387 SX with the i386 SX Microprocessor dramatically increases the processing speed of computer application software which utilizes mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The Military i387 SX Math Coprocessor adds over seventy mnemonics to the Military i386 SX Microprocessor instruction set. Specific Military i387 SX math operations include logarithmic, arithmetic, exponential, and trigonometric functions. The Military i387 SX supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The Military i387 SX Math Coprocessor is object code compatible with the Military i387 DX and upward object code compatible from the M80287 and M8087 Math Coprocessors. The Military i387 SX is manufactured with Intel's CHMOS III technology and packaged in a 68-lead PGA package. A low power consumption option allows use in portable applications.

NOTE:
References to devices within this document refer to the Military versions of those devices.

January 1994 Order Number: 271166-002

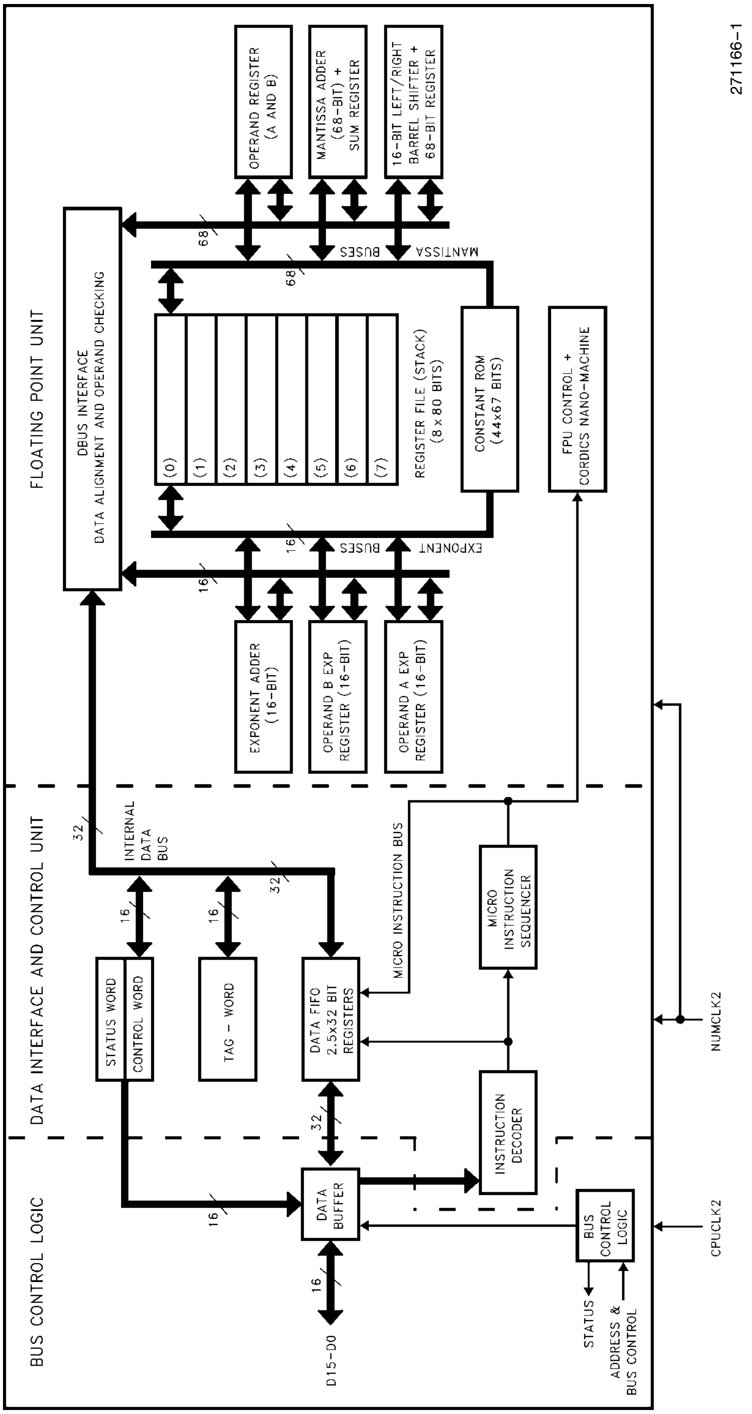

FLOATING POINT UNIT
DBUS INTERFACE
DATA ALIGNMENT AND OPERAND CHECKING
OPERAND REGISTER (A AND B)
MANTISSA ADDER (68-BIT) + SUM REGISTER
16-BIT LEFT/RIGHT BARREL SHIFTER + 68-BIT REGISTER
68
MANTISSA BUSES
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
REGISTER FILE (STACK) (8 x 80 BITS)
CONSTANT ROM (44x67 BITS)
FPU CONTROL + CORDICS NANO-MACHINE
16
EXPONENT BUSES
EXPONENT ADDER (16-BIT)
OPERAND B EXP REGISTER (16-BIT)
OPERAND A EXP REGISTER (16-BIT)
DATA INTERFACE AND CONTROL UNIT
32
INTERNAL DATA BUS
STATUS WORD
CONTROL WORD
TAG - WORD
DATA FIFO 2.5x32 BIT REGISTERS
MICRO INSTRUCTION BUS
MICRO INSTRUCTION SEQUENCER
INSTRUCTION DECODER
NUMCLK2
BUS CONTROL LOGIC
DATA BUFFER
D15-D0
STATUS
ADDRESS & BUS CONTROL
CPUCLK2
271166-1

Figure 0-1. Block Diagram

Military i387™ SX Math Coprocessor

CONTENTS

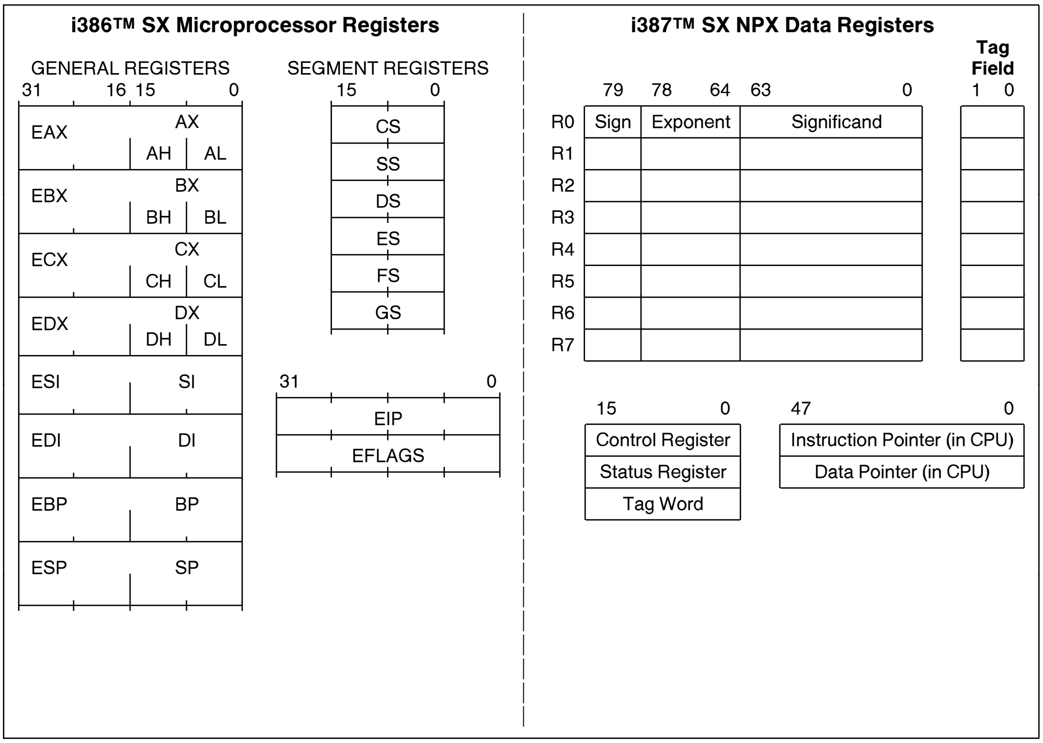


Figure 1-1. i386™ SX Microprocessor and i387™ SX Math Coprocessor Register Set

1.0 FUNCTIONAL DESCRIPTION

The i387 SX Math Coprocessor Extension (NPX) provides arithmetic instructions for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The i387 SX NPX effectively extends the register and instruction set of its CPU for existing data types and adds several new data types as well. Figure 1-1 shows the model of registers visible to i386 SX Microprocessor and i387 SX Math Coprocessor applications programs. Essentially, the i387 SX Math Coprocessor can be treated as an additional resource or an extension to the i386 SX Microprocessor. The i386 SX Microprocessor together with a i387 SX NPX can be used as a single unified system, the i386 SX Microprocessor and i387 SX Math Coprocessor.

The i387 SX Numerics Coprocessor Extension works the same whether the CPU is executing in real-address mode, protected mode, or virtual-8086 mode. All references to memory for numerics data or status information are performed by the CPU, and therefore obey the memory-management and protection rules of the CPU mode currently in effect. The i387 SX Numerics Coprocessor Extension merely operates on instructions and values passed to it by the CPU and therefore is not sensitive to the processing mode of the CPU.

In real-address mode and virtual-8086 mode, the i386 SX Microprocessor and i387 SX Math Coprocessor is completely upward compatible with software for the 8086/8087 and 80286/80287 real-address mode systems.

In protected mode, the i386 SX Microprocessor and i387 SX Math Coprocessor is completely upward compatible with software for the 80286/80287 protected mode system.

In all modes, the i386 SX Microprocessor and i387 SX Math Coprocessor is completely compatible with software for the i386 Microprocessor/i387 Math Coprocessor system.

The only differences of operation that may appear when 8086/8087 programs are ported to the protected-mode i386 SX Microprocessor and i387 SX Math Coprocessor system (*not* using virtual-8086 mode) is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR, and FSAVE. These instruction are normally used only by exception handlers and operating systems, not by applications programs.

2.0 PROGRAMMING INTERFACE

The i387 SX NPX adds to an i386 SX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the i387 SX NPX requires no special programming tools, because all new instructions and data types are directly supported by the assembler and compilers for high-level languages. All i386 Microprocessor development tools that support i387 NPX programs can also be used to develop software for the i386 SX Microprocessor and i387 SX Math Coprocessor. All 8086/8088 development tools that support the 8087 can also be used to develop software for the i386 SX Microprocessor and i387 SX 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 i386 SX Microprocessor and i387 SX Math Coprocessor.

The i387 SX NPX supports all i387 NPX instructions. The i386 SX Microprocessor and i387 SX Math Coprocessor supports all the same programs and gives the same results as an i386 Microprocessor and i387 Math Coprocessor.

All communication between the CPU and the NPX is transparent to applications software. The CPU automatically controls the NPX 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 NPX. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 7 at the end of this data sheet lists by class the instructions that the i387 SX NPX adds to the instruction set of an i386 SX Microprocessor system.

2.1 Data Types

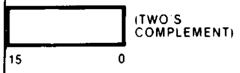
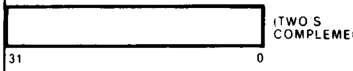
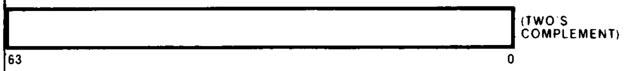
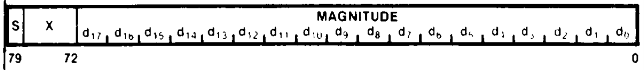
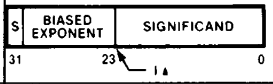
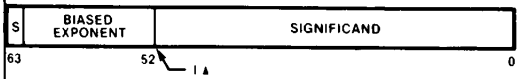
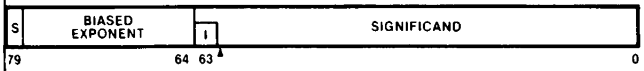
Table 2-1 lists the seven data types that the NPX supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses that correspond to the word size of the CPU; operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the NPX 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 NPX instruction accepts one or two operands and produces one (or sometimes two) results. In two-operand instructions, one operand is the contents of an NPX 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. i387™ SX NPX Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = HIGHEST ADDRESSED BYTE 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0 | 7 0
Word Integer	$\pm 10^4$	16 Bits	
Short Integer	$\pm 10^9$	32 Bits	
Long Integer	$\pm 10^{18}$	64 Bits	
Packed BCD	$\pm 10^{18}$	18 Digits	 S X MAGNITUDE d_{17} d_{16} d_{15} d_{14} d_{13} d_{12} d_{11} d_{10} d_9 d_8 d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0 79 72 0
Single Precision	$\pm 10^{\pm 38}$	24 Bits	
Double Precision	$\pm 10^{\pm 308}$	53 Bits	
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	 S BIASED EXPONENT I SIGNIFICAND 79 64 63 0

271166–2

NOTES:
(1) S = Sign bit (0 = positive, 1 = negative)
(2) d_n = Decimal digit (two per byte)
(3) X = Bits have no significance; NPX 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: $(-1)^S$ $(2^{E\text{-}BIAS})$ $(F_0\ F_1...)$

2.3 Register Set

Figure 1-1 shows the i387 SX NPX register set. When an NPX is present in a system, programmers may use these registers in addition to the registers normally available on the CPU.

2.3.1 DATA REGISTERS

i387 SX NPX computations use the NPX's data registers. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. Each of the eight data registers in the NPX is 80 bits wide and is divided into "fields" corresponding to the NPX's extended-precision real data type.

The NPX register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as individually addressable 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. The NPX 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 use. 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 data registers. The principal function of the tag word is to optimize the NPX's performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to identify special values (e.g. NaNs or denormals) in the contents of a stack location without the need to perform complex decoding of the actual data.

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 NPX. It may be read and inspected by programs.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It always has the same value as the ES bit (bit 7 of the status word); it does **not** indicate the status of the $\overline{\text{BUSY}}$ output of NPX.

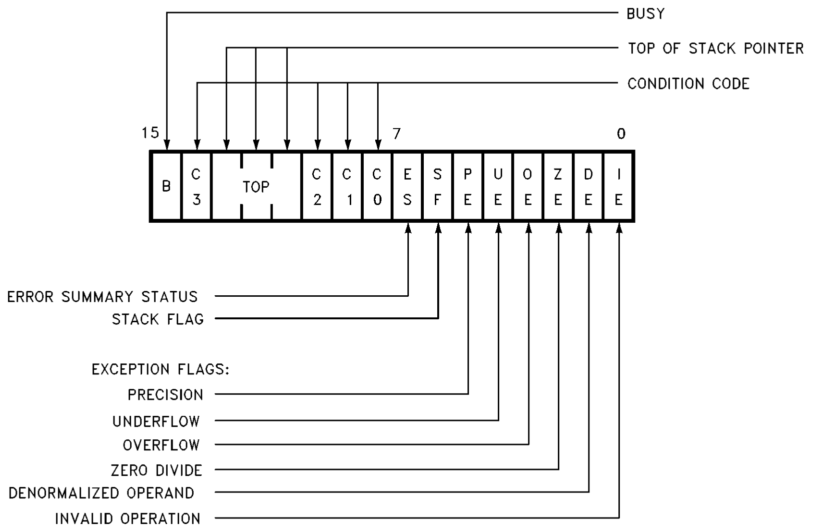
Bits 13–11 (TOP) point to the NPX register that is the current top-of-stack.

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.

15							0
TAG (7)	TAG (6)	TAG (5)	TAG (4)	TAG (3)	TAG (2)	TAG (1)	TAG (0)

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. Tag Word



271166–3

ES is set if any unmasked exception bit is set; cleared otherwise. See Table 2-2 for interpretation of condition code.
TOP values:

000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack
.
.
.
111 = Register 7 is Top of Stack

For definitions of exceptions, refer to the section entitled "Exception Handling"

Figure 2-2. Status Word

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 $\overline{\text{ERROR}}$ 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 NPX 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 $\overline{\text{ERROR}}$ output of the NPX 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 Q2	Q0	Q1 or O/$\overline{U}$	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/$\overline{U}$	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/$\overline{U}$	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/$\overline{U}$	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/$\overline{U}$	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/$\overline{U}$, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/$\overline{U}$	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 reudction bit is set if the oeprand 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

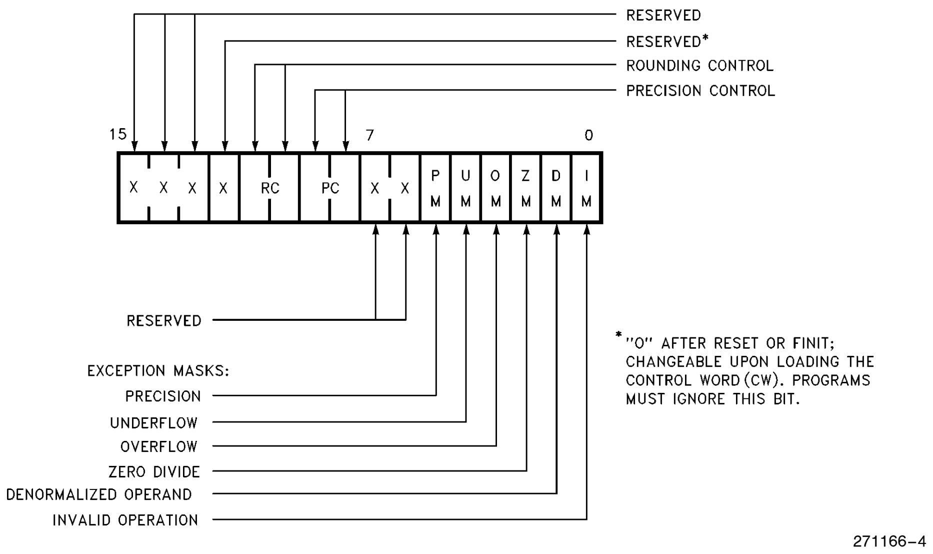
Condition Code					Interpretation after FPREM and FPREM1
C2	C3	C1	C0		
1	X	X	X		Incomplete Reduction: further interation required for complete reduction
0	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	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



Precision Control
00—24 bits (single precision)
01—(reserved)
10—53 bits (double precision)
11—64 bits (extended precision)

Rounding Control
00—Round to nearest or even
01—Round down (toward $-\infty$)
10—Round up (toward $+\infty$)
11—Chop (truncate toward zero)

Figure 2-3. Control Word

2.3.4 CONTROL WORD

The NPX provides several processing options that are selected by loading a control word from memory into the control register. Figure 2-3 shows the format and encoding of fields in the control word.

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

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

- The "infinity control bit" (bit 12) is not meaningful to the i387 SX NPX, and programs must ignore its value. To maintain compatibility with the 8087 and 80287, this bit can be programmed; however, regardless of its value, the i387 SX NPX always treats infinity in the affine sense ($-\infty < +\infty$). This bit is initialized to zero both after a hardware reset and after the FINIT instruction.
- 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 NPX 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.

2.3.5 INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any exceptions detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the i386 SX Microprocessor and i387 SX Math Coprocessor contains registers that aid in diagnosis. These registers supply the address of the failing instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written exception handlers. These registers are actually located in the CPU, but appear to be located in the NPX because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. Whenever the CPU executes 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 CPU (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). (See Figures 2-4, 2-5, 2-6, and 2-7.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

32-BIT PROTECTED MODE FORMAT

31	23	15	7 0	
RESERVED		CONTROL WORD		0
RESERVED		STATUS WORD		4
RESERVED		TAG WORD		8
IP OFFSET				C
00000	OPCODE $_{10..0}$	CS SELECTOR		10
DATA OPERAND OFFSET				14
RESERVED		OPERAND SELECTOR		18

Figure 2-4. Instruction and Data Pointer Image in Memory, 32-bit Protected-Mode Format

16-BIT PROTECTED MODE FORMAT

15 — 7 — 0	
CONTROL WORD	0
STATUS WORD	2
TAG WORD	4
IP OFFSET	6
CS SELECTOR	8
OPERAND OFFSET	A
OPERAND SELECTOR	C

Figure 2-5. Instruction and Data Pointer Image in Memory, 16-bit Protected-Mode Format

32-BIT REAL-ADDRESS MODE FORMAT

31	23	15	7	0	
RESERVED		CONTROL WORD			0
RESERVED		STATUS WORD			4
RESERVED		TAG WORD			8
RESERVED		INSTRUCTION POINTER 15..0			C
0 0 0 0	INSTRUCTION POINTER 31..16		0	OPCODE 10..0	10
RESERVED		OPERAND POINTER 15..0			14
0 0 0 0	OPERAND POINTER 31..16		0 0 0 0	0 0 0 0 0 0 0 0	18

Figure 2-6. Instruction and Data Pointer Image in Memory, 32-bit Real-Mode Format

16-BIT REAL-ADDRESS MODE AND VIRTUAL 8086 MODE FORMAT

15		7 — 0	
CONTROL WORD			0
STATUS WORD			2
TAG WORD			4
INSTRUCTION POINTER 15..0			6
IP19.16	0	OPCODE 10..0	8
OPERAND POINTER 15..0			A
DP 19.16	0	0 0 0 0 0 0 0 0 0 0 0	C

Figure 2-7. Instruction and Data Pointer Image in Memory, 16-bit Real-Mode Format

Table 2-6. CPU Interrupt Vectors Reserved for NPX

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of 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 NPX context may not belong to the current task.
9	In a protected-mode system, an operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for expand-up segments, zero for expand-down segments) and spanned inaccessible addresses[a]. 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. The segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The exception can be avoided by never allowing numerics operands to cross the end of a segment.
13	In a protected-mode system, the first word 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 NPX 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 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 NPX. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

NOTE:
a. 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 FFFCH will span addresses FFFC–FFFFH and 0000-0003H; however addresses FFFEH and FFFFH are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible segments but intermediate bytes of the operand fall in a not-present page or in a segment or page to which the procedure does not have access rights.

2.4 Interrupt Description

CPU interrupts are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2-6 shows these interrupts and their functions.

2.5 Exception Handling

The NPX 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 NPX 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 $\overline{\text{ERROR}}$ signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 16 occurs. The exception condition must be resolved via an interrupt service routine. The return address pushed onto the CPU stack upon entry to the service routine does not necessarily point to the failing instruction nor to the following instruction. The CPU 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

After FNINIT or RESET, the control word contains the value 037FH (all exceptions masked, precision control 64 bits, rounding to nearest) the same values as in an 80287 after RESET. For compatibility with the 8087 and 80287, the bit that used to indicate infinity control (bit 12) is set to zero; however, regardless of its setting, infinity is treated in the affine sense. After FNINIT or RESET, the status word is initialized as follows:

- All exceptions are set to zero.
- Stack TOP is zero, so that after the first push the stack top will be register seven (111B).
- The condition code C_3–C_0 is undefined.
- The B-bit is zero.

The tag word contains FFFFH (all stack locations are empty).

The i386 SX Microprocessor and i387 SX Math Coprocessor initialization software must execute an FNINIT instruction (i.e an FINIT without a preceding WAIT) after RESET. The FNINIT is not strictly required for the 80287 software, but Intel recommends its use to help ensure upward compatibility with other processors. After a hardware RESET, the $\overline{\text{ERROR}}$ output is asserted to indicate that a i387 SX NPX 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 cleared. 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 i387 SX NPX and the 80287. Any migration from the 8087 directly to the i387 SX NPX 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 i387 SX NPX 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 i387 SX NPX supports only affine closure for infinity arithmetic, not projective closure.

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

Rounding control is in effect for FLD *constant*.

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

After reset, FINIT, and incomplete FPREM, the i387 SX NPX resets to zero the condition code bits C_3–C_0 of the status word.

Table 2-7. Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signalling NaN, unsupported format, indeterminate for ($0\cdot\infty$, 0/0, $(+\infty) + (-\infty)$, etc.), or stack overflow/underflow (SF is also set)	Result is a quiet NaN, integer indefinite, or BCD indefinte
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 ∞
Overflow	The result is too large in magnitude to fit in the specified format	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes the 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

In conformance with the IEEE standard, the i387 SX NPX does not support the special data formats pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

The denormal exception has a different purpose on the i387 SX NPX. A system that uses the denormal-exception handler solely to normalize the denormal operands, would better mask the denormal exception on the i387 SX NPX. The i387 SX NPX automatically normalizes denormal operands when the denormal exception is masked.

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 i387 SX NPX:

1. When the overflow or underflow exception is masked, the i387 SX NPX differs from the 80287 in rounding when overflow or underflow occurs. The i387 SX NPX produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the i387 SX NPX 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 i387 SX NPX 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 signalling NaN, FLD *single/double precision* signals an invalid-operand exception.
12. The i387 SX NPX only generates quiet NaNs (as on the 80287); however, the i387 SX NPX 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, ∞) generates the invalid operation exception.
    - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE(finite, $+\infty$) generates ∞ 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 i387 SX NPX 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, an overscore appearing over a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no overscore is present over the signal, the signal is asserted when at the high voltage level.

3.1 Signal Description

In the following signal descriptions, the i387 SX NPX pins are grouped by function as shown by Table 3-1. Table 3-1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. Refer to Figure 3-2 and Table 3-2 for pin configuration.

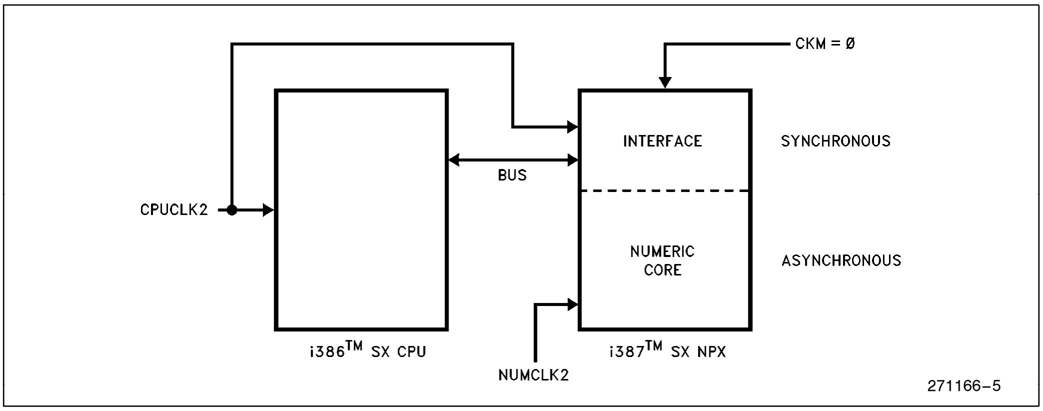


Figure 3.1. Asynchronous Operation

Table 3-1. Pin Summary

Pin Name	Function	Active State	Input/ Output	Referenced To...
Execution Control				
CPUCLK2	i386™ SX Microprocessor CLocK 2		I	
NUMCLK2	NPX CLocK 2		I	
CKM	NPX ClocKing Mode		I	
RESETIN	System reset	High	I	CPUCLK2
NPX Handshake				
PEREQ	Processor Extension REQuest	High	O	STEN/CPUCLK2
$\overline{\text{BUSY}}$	Busy status	Low	O	STEN/CPUCLK2
$\overline{\text{ERROR}}$	Error status	Low	O	STEN/NUMCLK2
Bus Interface				
D15–D0	Data pins	High	I/O	CPUCLK2
W/$\overline{\text{R}}$	Write/Read bus cycle	Hi/Lo	I	CPUCLK2
$\overline{\text{ADS}}$	ADdress Strobe	Low	I	CPUCLK2
$\overline{\text{READY}}$	Bus ready input	Low	I	CPUCLK2
$\overline{\text{READYO}}$	Ready output	Low	O	STEN/CPUCLK2
Chip/Port Select				
STEN	STatus ENable	High	I	CPUCLK2
$\overline{\text{NPS1}}$	NPX select #1	Low	I	CPUCLK2
NPS2	NPX select #2	High	I	CPUCLK2
$\overline{\text{CMD0}}$	CoMmanD	Low	I	CPUCLK2
Power and Ground				
V_{CC}	System power			
V_{SS}	System ground			

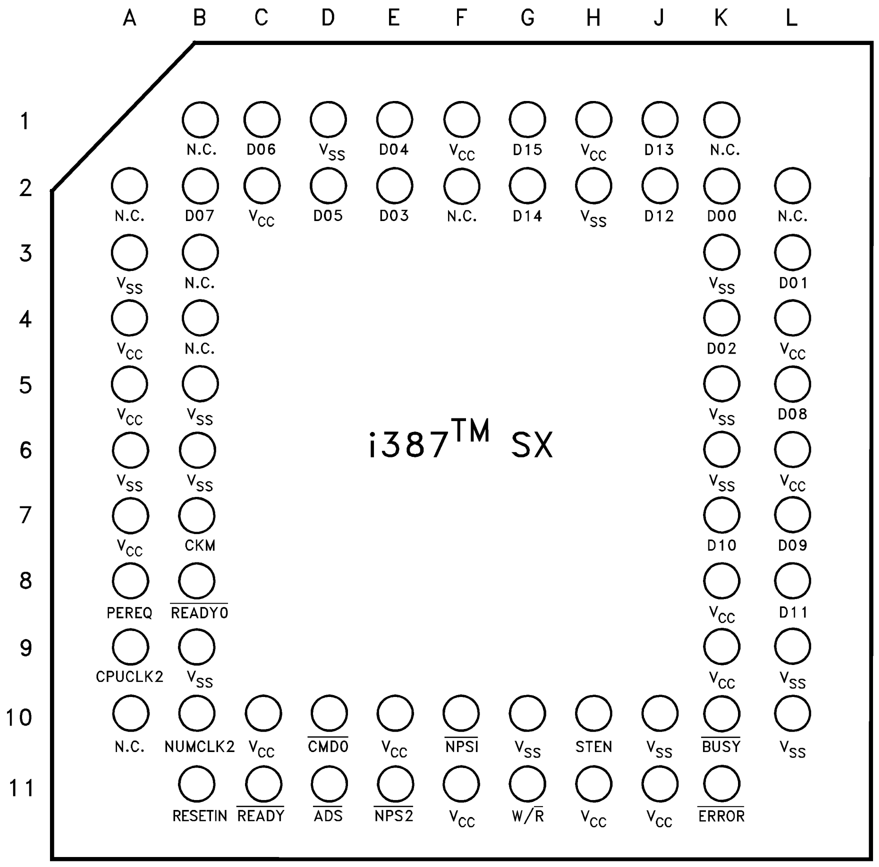
Table 3-2

Mi387SX PGA Pin Cross Reference
(Sorted by Signal Name)

Signal	Pin
$W/\overline{R}$	G11
V_{SS}	L9
V_{SS}	L10
V_{SS}	K6
V_{SS}	K5
V_{SS}	K3
V_{SS}	J10
V_{SS}	H2
V_{SS}	G10
V_{SS}	D1
V_{SS}	B9
V_{SS}	B6
V_{SS}	B5
V_{SS}	A6
V_{SS}	A3
V_{CC}	L6
V_{CC}	L4
V_{CC}	K9
V_{CC}	K8
V_{CC}	J11
V_{CC}	H11
V_{CC}	H1
V_{CC}	F11
V_{CC}	F1
V_{CC}	E10
V_{CC}	C2
V_{CC}	C10
V_{CC}	A7
V_{CC}	A5
V_{CC}	A4
STEN	H10
RESETIN	B11
$\overline{\text{READY}}$	C11
$\overline{\text{READYO}}$	B8
PEREQ	A8
NUMCLK2	B10
NPS2	E11
$\overline{\text{NPS1}}$	F10
NC	L2
NC	K1
NC	F2
NC	B4
NC	B3
NC	B1
NC	A2
NC	A10
$\overline{\text{ERROR}}$	K11
D15	G1
D14	G2
D13	J1
D12	J2
D11	L8
D10	K7
D09	L7
D08	L5
D07	B2
D06	C1
D05	D2
D04	E1
D03	E2
D02	K4
D01	L3
D00	K2
CPUCLK2	A9
$\overline{\text{CMD0}}$	D10
CKM	B7
$\overline{\text{BUSY}}$	K10
$\overline{\text{ADS}}$	D11

Mi387SX PGA Pin Cross Reference
(Sorted by Pin)

Pin	Signal
L10	V_{SS}
L9	V_{SS}
L8	D11
L7	D09
L6	V_{CC}
L5	D08
L4	V_{CC}
L3	D01
L2	NC
K11	$\overline{\text{ERROR}}$
K10	$\overline{\text{BUSY}}$
K9	V_{CC}
K8	V_{CC}
K7	D10
K6	V_{SS}
K5	V_{SS}
K4	D02
K3	V_{SS}
K2	D00
K1	NC
J11	V_{CC}
J10	V_{SS}
J2	D12
J1	D13
H11	V_{CC}
H10	STEN
H2	V_{SS}
H1	V_{CC}
G11	$W/\overline{R}$
G10	V_{SS}
G2	D14
G1	D15
F11	V_{CC}
F10	$\overline{\text{NPS1}}$
F2	NC
F1	V_{CC}
E11	NPS2
E10	V_{CC}
E2	D03
E1	D04
D11	$\overline{\text{ADS}}$
D10	$\overline{\text{CMD0}}$
D2	D05
D1	V_{SS}
C11	$\overline{\text{READY}}$
C10	V_{CC}
C2	V_{CC}
C1	D06
B11	RESETIN
B10	NUMCLK2
B9	V_{SS}
B8	$\overline{\text{READYO}}$
B7	CKM
B6	V_{SS}
B5	V_{SS}
B4	NC
B3	NC
B2	D07
B1	NC
A10	NC
A9	CPUCLK2
A8	PEREQ
A7	V_{CC}
A6	V_{SS}
A5	V_{CC}
A4	V_{CC}
A3	V_{SS}
A2	NC



271166–18

The term "top view" means "as viewed when mounted in a printed-circuit board".

Figure 3-2. i387™ SX Math Coprocessor PGA Pinout —View from Pin Side

All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D15–D0 are also tristate; they leave floating state only during cycles when the NPX is selected (i.e. when STEN, $\overline{\text{NPS1}}$, and NPS2 are all active).

3.1.1 i386™ SX CPU CLOCK 2 (CPUCLK2)

This input uses the CLK2 signal of the CPU to time the bus control logic. Several other NPX 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 NPX. 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 i387™ SX NPX 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 NPX. 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) signals on this pin are ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires MOS-level input.

3.1.3 CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to V_{CC} (HIGH), the NPX operates in synchronous mode; when strapped to V_{SS} (LOW), the NPX 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 CPU.

3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the NPX to terminate its present activity and to enter a dormant state. RESETIN must remain active (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 two) is the same as the phase of the internal clock of the CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first NPX instruction is written into the NPX. This pin should be connected to the CPU RESET pin. Table 3-3 shows the status of the output pins during the reset sequence. After a reset, all output pins return to their inactive states.

Table 3-3. Output Pin Status during Reset

Pin Value	Pin Name
HIGH	$\overline{\text{READYO}}$, $\overline{\text{BUSY}}$
LOW	PEREQ, $\overline{\text{ERROR}}$
Tri-State OFF	D15–D0

3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the CPU that the NPX 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 $\overline{\text{BUSY}}$ goes inactive. This signal is referenced to CPUCLK2. It should be connected to the CPU PEREQ input.

3.1.6 BUSY STATUS ($\overline{\text{BUSY}}$)

When active, this pin signals to the CPU that the NPX is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the CPU $\overline{\text{BUSY}}$ pin.

3.1.7 ERROR STATUS ($\overline{\text{ERROR}}$)

This pin reflects the ES bit of the status register. When active, it indicates that an unmasked exception has occurred. This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, FNSAVE, FLDCW, FLDENV, and FRSTOR. This pin is referenced to CPUCLK2. It should be connected to the $\overline{\text{ERROR}}$ pin of the CPU.

3.1.8 DATA PINS (D15–D0)

These bidirectional pins are used to transfer data and opcodes between the CPU and NPX. They are normally connected directly to the corresponding 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/$\overline{\text{R}}$)

This signal indicates to the NPX whether the CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the CPU's W/$\overline{\text{R}}$ pin. HIGH indicates a write cycle; LOW a read cycle. This input is ignored if any of the signals STEN, $\overline{\text{NPS1}}$, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

3.1.10 ADDRESS STROBE ($\overline{\text{ADS}}$)

This input, in conjunction with the $\overline{\text{READY}}$ input, indicates when the NPX bus-control logic may sample W/$\overline{\text{R}}$ and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the $\overline{\text{ADS}}$ pin of the CPU.

3.1.11 BUS READY INPUT ($\overline{\text{READY}}$)

This input indicates to the NPX when a 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 $\overline{\text{READY}}$. This input should be connected to the same signal that drives the CPU's $\overline{\text{READY}}$ input. Setup and hold times are referenced to CPUCLK2.

3.1.12 READY OUTPUT ($\overline{\text{READYO}}$)

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 $\overline{\text{READY}}$ input of the CPU. Refer to the section entitled "Bus Operation" for details. This pin is activated only during bus cycles that select the NPX. This signal is referenced to CPUCLK2.

3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the NPX. When inactive, this pin forces, $\overline{\text{BUSY}}$, $\overline{\text{PEREQ}}$, $\overline{\text{ERROR}}$, and $\overline{\text{READYO}}$ outputs into floating state. D15–D0 are normally floating; they 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 NPX. 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 $\overline{\text{NPS1}}$, NPS2, and $\overline{\text{CMD0}}$ (i.e. if STEN changes state during an NPX bus cycle, it must change state during the same CLK period as the $\overline{\text{NPS1}}$, NPS2, and $\overline{\text{CMD0}}$ signals).

3.1.14 NPX SELECT 1 ($\overline{\text{NPS1}}$)

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

3.1.15 NPX SELECT 2 (NPS2)

When active (along with STEN and $\overline{\text{NPS1}}$) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the A23 pin of the CPU, so that the NPX is selected only when the CPU issues one of the I/O addresses reserved for the NPX (8000F8H, 8000FCH or 8000FEH which is treated as 8000FCH by the NPX). Setup and hold times are referenced to CPUCLK2.

3.1.16 COMMAND ($\overline{\text{CMD0}}$)

During a write cycle, this signal indicates whether an opcode ($\overline{\text{CMD0}}$ active) or data ($\overline{\text{CMD0}}$ inactive) is being sent to the NPX. During a read cycle, it indicates whether the control or status register ($\overline{\text{CMD0}}$ active) or a data register ($\overline{\text{CMD0}}$ inactive) is being read. $\overline{\text{CMD0}}$ should be connected directly to the A2 output of the CPU. Setup and hold times are referenced to CPUCLK2.

3.1.17 SYSTEM POWER (V_{CC})

System power provides the +5V DC supply input. All V_{CC} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.1.18 SYSTEM GROUND (V_{SS})

All V_{SS} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.2 System Configuration

The i387 SX Math Coprocessor is designed to interface with the i386 SX Microprocessor as shown by Figure 3-3. A dedicated communication protocol makes possible high-speed transfer of opcodes and operands between the CPU and NPX. The i387 SX NPX is designed so that no additional components are required for interface with the CPU. Most control pins of the NPX are connected directly to pins of the CPU.

The interface between the NPX and the CPU has these characteristics:

- The NPX shares the local bus of the i386 SX Microprocessor.
- The CPU and NPX share the same reset signals. They may also share the same clock input; however, for greatest performance, an external oscillator may be needed.
- The corresponding $\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$, and PEREQ pins are connected together.
- The NPX $\overline{\text{NPS1}}$ and NPS2 inputs are connected to the latched CPU M/$\overline{\text{IO}}$ and A23 outputs respectively. For coprocessor cycles, M/$\overline{\text{IO}}$ is always LOW and A23 always HIGH.
- The NPX input CMD0 is connected to the latched A_2 output. The i386 SX Microprocessor generates address 8000F8H when writing a command and address 8000FCH or 8000FEH (treated as 8000FCH by the i387 SX NPX) when writing or reading data. It does not generate any other addresses during NPX bus cycles.

3.3 Processor Architecture

As shown by the block diagram on the front page, the i387 SX NPX 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 CPU bus tracking and interface. The BCL is the only unit in the NPX that must run synchronously with the CPU; the rest of the NPX can run asynchronously with respect to the CPU.

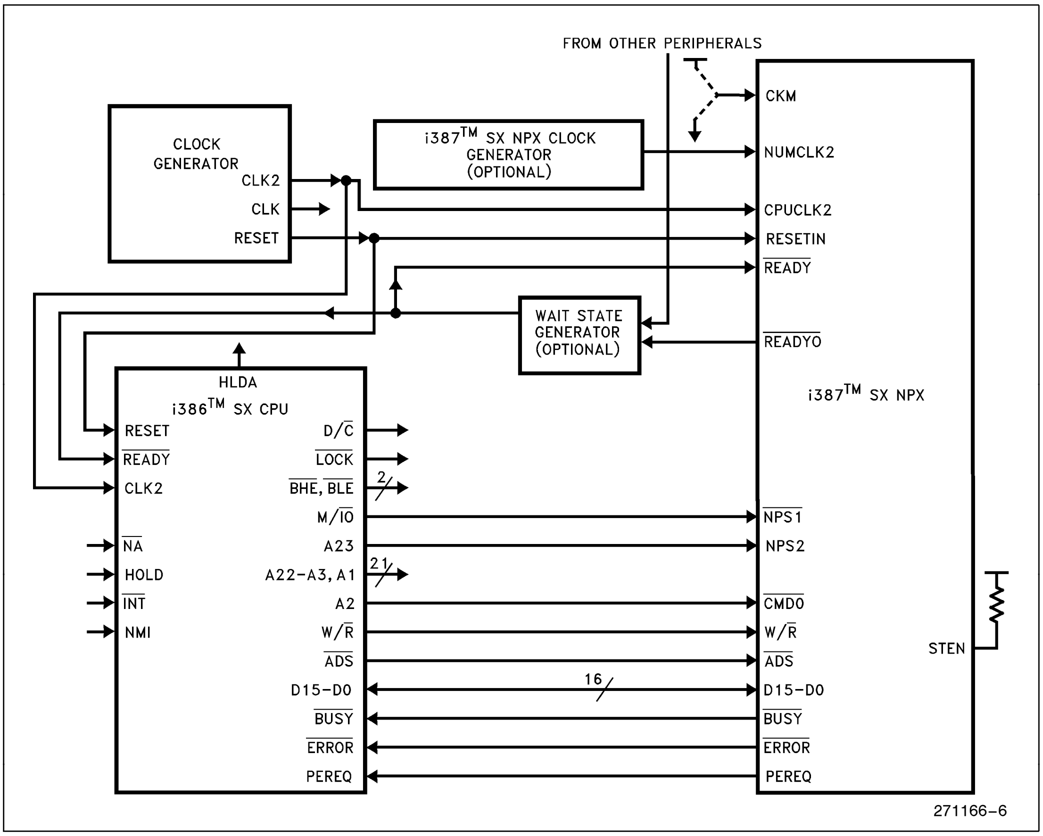


Figure 3-3. i386™ SX CPU and i387™ SX NPX System Configuration

3.3.1 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 NPX and transferring outputs from the NPX to memory.

3.3.2 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, FSTCW, FSETPM, or FRSTPM, 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 NPX activities with the CPU.

3.3.3 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.

3.4 Bus Cycles

The pins STEN, $\overline{\text{NPS1}}$, NPS2, CMD0, and W/$\overline{\text{R}}$ identify bus cycles for the NPX. Table 3-4 defines the types of NPX bus cycles.

3.4.1 i387™ SX NPX ADDRESSING

The $\overline{\text{NPS1}}$, NPS2, and CMD0 signals allow the NPX to identify which bus cycles are intended for the NPX. The NPX responds to I/O cycles when the I/O address is 8000F8H, 8000FCH or 8000FEH (treated as 8000FCH by the i387 SX NPX). The NPX responds to I/O cycles when bit 23 of the I/O address is set. In other words, the NPX acts as an I/O device in a reserved I/O address space.

Because A23 is used to select the i387 SX Numerics Coprocessor Extension for data transfers, it is not possible for a program running on the CPU to address the NPX with an I/O instruction. Only ESC instructions cause the CPU to communicate with the NPX.

3.4.2 CPU/NPX SYNCHRONIZATION

The pins $\overline{\text{BUSY}}$, PEREQ, and $\overline{\text{ERROR}}$ are used for various aspects of synchronization between the CPU and the NPX.

$\overline{\text{BUSY}}$ is used to synchronize instruction transfer from the CPU to the NPX. When the NPX recognizes an ESC instruction, it asserts $\overline{\text{BUSY}}$. For most ESC instructions, the CPU waits for the NPX to deassert $\overline{\text{BUSY}}$ before sending the new opcode.

The NPX uses the PEREQ pin of the CPU to signal that the NPX is ready for data transfer to or from its data FIFO. The NPX does not directly access memory; rather, the CPU provides memory access services for the NPX. (For this reason, memory access on behalf of the NPX always obeys the protection rules applicable to the current CPU mode.) Once the CPU initiates an NPX instruction that has operands, the CPU waits for PEREQ signals that indicate when the NPX is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the CPU continues program execution while the NPX executes the ESC instruction.

In 8086/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In the i386 SX Microprocessor and i387 SX Math Coprocessor systems, however, WAIT instructions are required only for operand synchronization; namely, after NPX stores to memory (except FSTSW and FSTCW) or load from memory. (In 80286/80287 systems, WAIT is required before FLDENV and FRSTOR; with the i386 SX Microprocessor and i387 SX Math Coprocessor, WAIT is not required in these cases.) Used this way, WAIT ensures that the value has already been written or read by the NPX before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the CPU, the NPX can process the instruction in parallel with and independent of the host CPU. When the NPX detects an exception, it asserts the $\overline{\text{ERROR}}$ signal, which causes a CPU interrupt.

3.4.3 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the NPX (the FPU) can operate either 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 NPX is synchronized with the CPU

Table 3-4. Bus Cycle Definition

STEN	$\overline{\text{NPS1}}$	NPS2	$\overline{\text{CMD0}}$	W/$\overline{\text{R}}$	Bus Cycle Type
0	x	x	x	x	NPX not selected and all outputs in floating state
1	1	x	x	x	NPX not selected
1	x	0	x	x	NPX not selected
1	0	1	0	0	CW or SW read from NPX
1	0	1	0	1	Opcode write to NPX
1	0	1	1	0	Data read from NPX
1	0	1	1	1	Data write to NPX

clock. Use of asynchronous mode allows the CPU and the FPU section of the NPX 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.

3.4.4 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, $\overline{\text{READYO}}$ can drive the CPU's $\overline{\text{READY}}$ input. If this pin is used, it should be connected to the logic that ORs all READY outputs from peripherals on the CPU bus. $\overline{\text{READYO}}$ is asserted by the NPX only during I/O cycles that select the NPX. Refer to Section 4.0 "Bus Operation" for details.

4.0 BUS OPERATION

With respect to bus interface, the i387 SX NPX is fully synchronous with the CPU. Both operate at the same rate, because each generates its internal CLK signal by dividing CPUCLK2 by two. Furthermore, both internal CLK signals are in phase, because they are synchronized by the same RESETIN signal.

A bus cycle for the NPX starts when the CPU activates $\overline{\text{ADS}}$ and drives new values on the address and cycle-definition lines. The NPX examines the address and cycle-definition lines in the same CLK period during which $\overline{\text{ADS}}$ is activated. This CLK period is considered the first CLK of the bus cycle. During this first CLK period, the NPX also examines the R/$\overline{\text{W}}$ 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 i387 SX NPX supports both pipelined (i.e. overlapped) and nonpipelined bus cycles. A nonpipelined cycle is one for which the CPU asserts $\overline{\text{ADS}}$ when no other NPX bus cycle is in progress. A pipelined bus cycle is one for which the CPU asserts $\overline{\text{ADS}}$ and provides valid next-address and control signals before the prior NPX cycle terminates. The CPU may do this as early as the second CLK period after asserting $\overline{\text{ADS}}$ for the prior cycle. Pipelining increases the availability of the bus by at least one CLK period. The i387 SX NPX supports pipelined bus cycles in order to optimize address pipelining by the CPU for memory cycles.

Bus operation is described in terms of an abstract state machine. Figure 4-1 illustrates the states and state transitions for NPX 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 every nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the $\overline{\text{READY}}$-sensitive state. Different types of bus cycles may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until $\overline{\text{READY}}$ is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying $\overline{\text{READY}}$, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle. This state is not used by nonpipelined cycles.

Note that the bus logic tracks bus state regardless of the values on the chip/port select pins.

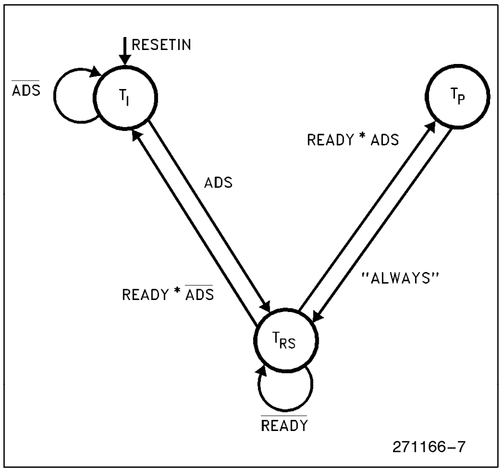


Figure 4-1. Bus State Diagram

The $\overline{\text{READYO}}$ output of the NPX indicates when an NPX bus cycle may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), $\overline{\text{READYO}}$ is always asserted during the first T_{RS} state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and FRSTOR, $\overline{\text{READYO}}$ 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 may use $\overline{\text{READYO}}$ in one of the following ways:

1. Connect it (directly or through logic that ORs $\overline{\text{READY}}$ signals from other devices) to the $\overline{\text{READY}}$ inputs of the CPU and NPX.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of i387 SX NPX 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 NPX operand-transfer cycles. Note however that, during FRSTOR, some consecutive accesses to the NPX do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see the figure "Other Parameters" in section 5.

4.1 Nonpipelined Bus Cycles

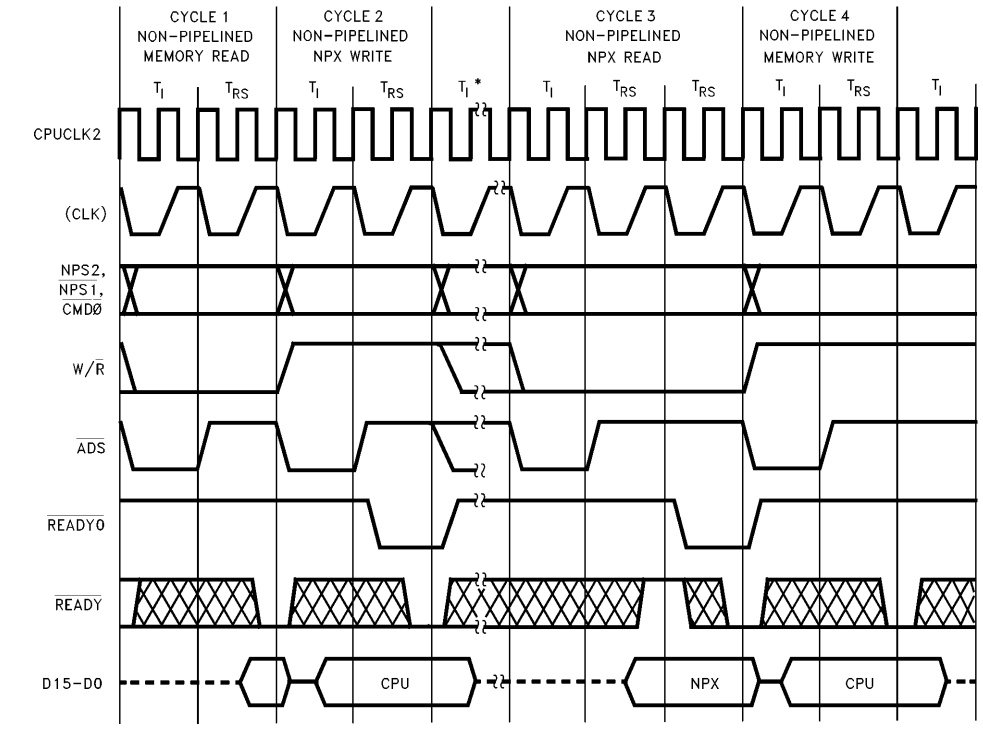
Figure 4-2 illustrates bus activity for consecutive nonpipelined bus cycles.

At the second clock of the bus cycle, the NPX enters the T_{RS} state. During this state, it samples the $\overline{\text{READY}}$ input and stays in this state as long as $\overline{\text{READY}}$ is inactive.

4.1.1 WRITE CYCLE

In write cycles, the NPX drives the $\overline{\text{READYO}}$ signal for one CLK period during the second CLK period of the cycle (i.e. the first T_{RS} state); therefore, the fastest write cycle takes two CLK periods (see cycle 2 of Figure 4-2). For the instructions FLDENV and FRSTOR, however, the NPX forces a wait state by delaying the activation of $\overline{\text{READYO}}$ to the second T_{RS} state (not shown in Figure 4-2).

The NPX samples the D15–D0 inputs into data latches at the falling edge of CLK as long as it stays in T_{RS} state.



CYCLE 1 NON-PIPELINED MEMORY READ
CYCLE 2 NON-PIPELINED NPX WRITE
CYCLE 3 NON-PIPELINED NPX READ
CYCLE 4 NON-PIPELINED MEMORY WRITE
T_I
T_RS
T_I *
CPUCLK2
(CLK)
NPS2, NPS1, CMD0
W/R
ADS
READYO
READY
D15-D0
CPU
NPX

271166–8

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 4-2. Nonpipelined Read and Write Cycles

When $\overline{\text{READY}}$ is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert $\overline{\text{ADS}}$ again, signaling the beginning of yet another cycle.

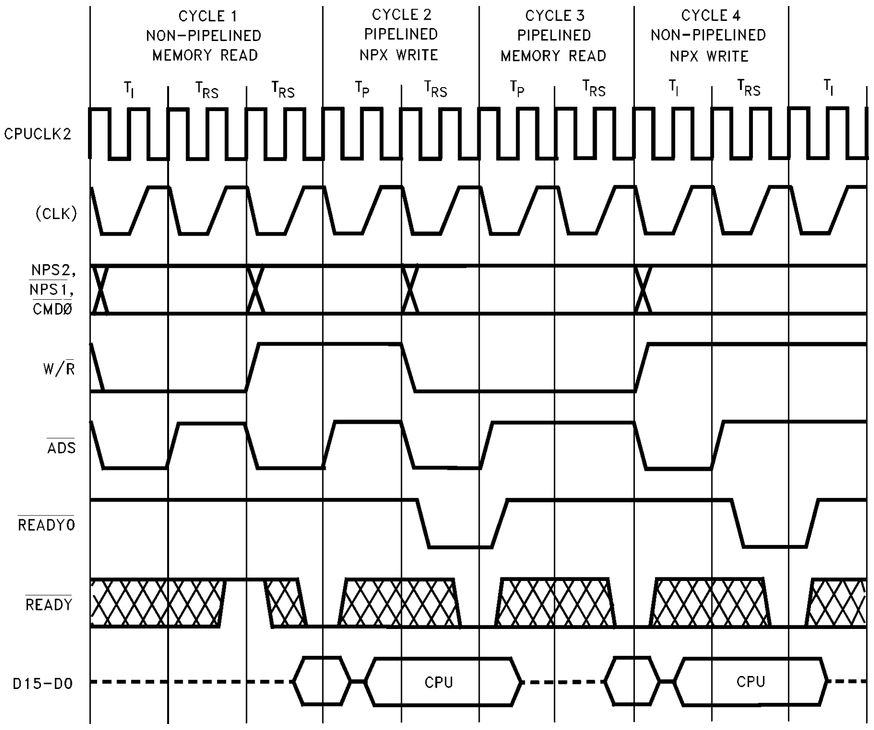
4.1.2 READ CYCLE

At the rising edge of CLK in the second CLK period of the cycle (i.e. the first T_{RS} state), the NPX starts to drive the D15–D0 outputs and continues to drive them as long as it stays in T_{RS} state.

At least one wait state must be inserted to ensure that the CPU latches the correct data. Because the NPX starts driving the 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 CPU before the next falling edge of CLK. Therefore, the NPX does not drive the $\overline{\text{READYO}}$ signal until the third CLK period of the cycle. Thus, if the $\overline{\text{READYO}}$ output drives the CPU's $\overline{\text{READY}}$ input, one wait state is automatically inserted.

Because one wait state is required for NPX reads, the minimum length of an NPX read cycle is three CLK periods, as cycle 3 of Figure 4-2 shows.

When $\overline{\text{READY}}$ is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert $\overline{\text{ADS}}$ again, signaling the beginning of yet another cycle. The transition from T_{RS} state to idle state causes the NPX to put the tristate D15–D0 outputs into the floating state, allowing another device to drive the data bus.



271166–9

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 $\overline{\text{READY}}$ and $\overline{\text{ADS}}$ are sampled active at the end of a T_{RS} state of the current cycle.

Figure 4-3. Fastest Transitions to and from Pipelined Cycles

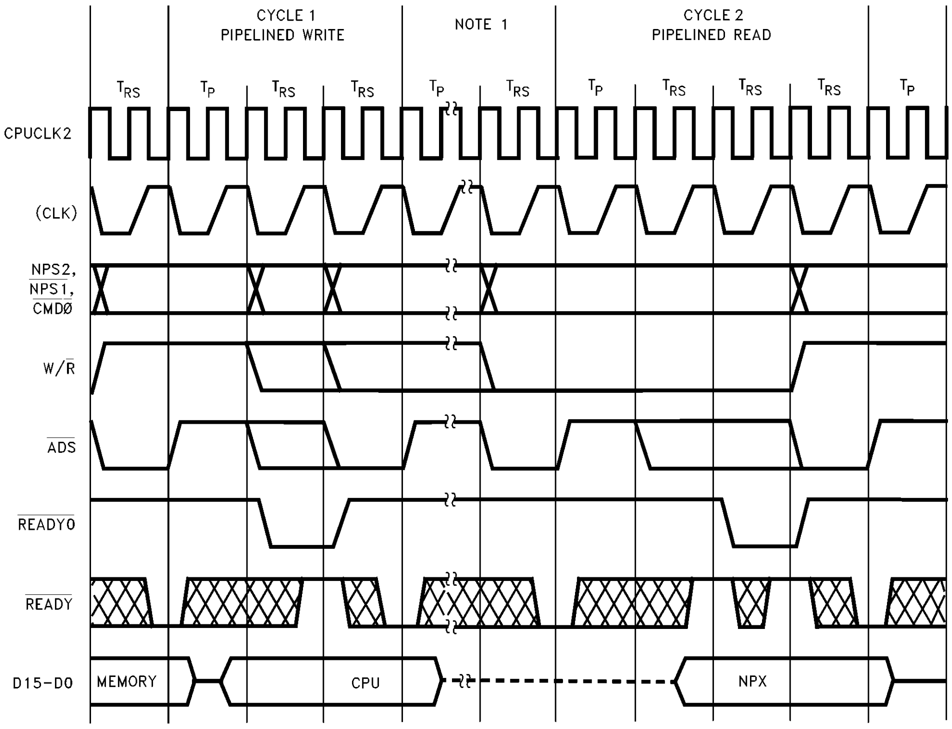
4.2 Pipelined Bus Cycles

Because all the activities of the NPX 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 during each state are detailed in the previous section "Nonpipelined Bus Cycles".

When the CPU asserts $\overline{ADS}$ before the end of a bus cycle, both $\overline{ADS}$ and $\overline{READY}$ are active during a T_{RS} state. This condition causes the NPX to change to a different state named T_P. One clock period after a T_P state, the NPX always returns to T_{RS} state. In consecutive pipelined cycles, the NPX bus logic uses only the T_{RS} and T_P states.

Figure 4-3 shows the fastest transitions into and out of the pipelined bus cycles. Cycle 1 in the 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 $\overline{READY}$ is asserted before the earliest possible assertion of $\overline{ADS}$ for the next cycle.)

Figure 4-4 shows pipelined write and read cycles with one additional T_{RS} state beyond the minimum required. To delay the assertion of $\overline{READY}$ requires external logic.



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

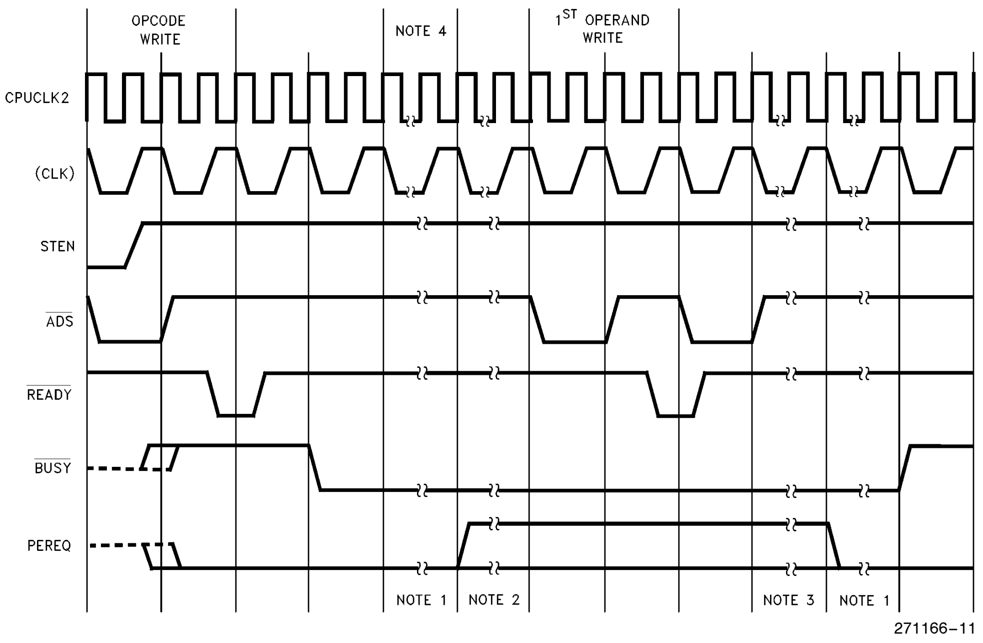
Figure 4-4. Pipelined Cycles with Wait States

4.3 Bus Cycles of Mixed Type

When the NPX bus logic is in the T_{RS} state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of $\overline{ADS}$ and $\overline{READY}$. In a nonpipelined cycle, only $\overline{READY}$ is activated, and the transition is from T_{RS} state to idle state. In a pipelined cycle, both $\overline{READY}$ and $\overline{ADS}$ 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.

4.4 $\overline{BUSY}$ and PEREQ Timing Relationship

Figure 4-5 shows the activation of $\overline{BUSY}$ at the beginning of instruction execution and its deactivation upon completion of the instruction. PEREQ is activated within this interval. If $\overline{ERROR}$ (not shown in the figure) is ever asserted, it would be asserted at least six CPUCLK2 periods after the deactivation of PEREQ and would be deasserted at least six CPUCLK2 periods before the deactivation of $\overline{BUSY}$. Figure 4-5 also shows that STEN is activated at the beginning of an NPX bus cycle.



NOTES:
1. Instruction dependent.
2. PEREQ is an asynchronous input to the i386™ Microprocessor; it may not be asserted (instruction dependent).
3. More operand transfers.
4. Memory read (operand) cycle is not shown.

Figure 4-5. STEN, $\overline{BUSY}$, and PEREQ Timing Relationships

5.0 ELECTRICAL DATA

5.1 Absolute Maximum Ratings

NOTE:
Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Case Temperature T_C under Bias	$-55°C$ to $+125°C$
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

5.2 Operating Conditions

Table 5-1

MIL-STD-883

Symbol	Description	Min	Max	Units
T_C	Case Temperature (Instant On)	-55	$+125$	°C
V_{CC}	Digital Supply Voltage	4.75	5.25	V

Extended Temperature

Symbol	Description	Min	Max	Units
T_C	Case Temperature (Instant On)	-40	$+110$	°C
V_{CC}	Digital Supply Voltage	4.75	5.25	V

Military Temperature Only (MTO)

Symbol	Description	Min	Max	Units
T_C	Case Temperature (Instant On)	-55	$+125$	°C
V_{CC}	Digital Supply Voltage	4.75	5.25	V

5.3 DC Characteristics (Over Specified Operating Conditions)

Table 5-2. DC Specifications

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 and NUMCLK2 Input LO Voltage	-0.3*	$+0.8$	V	
V_{CH}	CPUCLK2 and NUMCLK2 Input HI Voltage	$V_{CC}-0.8$	$V_{CC}+0.3$*	V	

Table 5-2. DC Specifications (Continued)

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{OL}	Output LO Voltage		0.45	V	(Note 2)
V_{OH}	Output HI Voltage	2.4		V	(Note 3)
V_{OH}	Output HI Voltage	$V_{CC} - 0.8$		V	(Note 4)
I_{CC}	Power Supply Current				
	NUMCLK2 = 40 MHz		300	mA	(Note 5)
	NUMCLK2 = 32 MHz		250	mA	(Note 5)
	NUMCLK2 = 2 MHz		100	mA	(Note 5)
I_{LI}	Input Leakage Current		±15	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	I/O Leakage Current		±15	μA	$0.45V \leq V_O \leq V_{CC}$
C_{IN}	Input Capacitance		10*	pF	fc = 1MHz
C_O	I/O or Output Capacitance		12*	pF	fc = 1MHz
C_{CLK}	Clock Capacitance		20*	pF	fc = 1MHz

NOTES:
*Guaranteed, not tested.
1. This parameter is for all inputs, excluding the clock inputs.
2. This parameter is measured at I_{OL} as follows: data = 4.0 mA
$\overline{\text{READY0}}$, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, PEREQ = 2.5 mA
3. This parameter is measured at I_{OH} as follows: data = 1.0 mA
$\overline{\text{READY0}}$, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, PEREQ = 0.6 mA
4. This parameter is measured at I_{OH} as follows: data = 1 mA
$\overline{\text{READY0}}$, $\overline{\text{ERROR}}$, $\overline{\text{BUSY}}$, PEREQ = 0.12 mA
5. I_{CC} is measured at steady state, 50 pF capacitive loading on the outputs, and worst-case DC level at the inputs; CPUCLK2 at the same frequency as NUMCLK2.

5.4 AC Characteristics (Over Specified Operating Conditions)

Table 5-3a. Combinations of Bus Interface and Execution Speeds

Functional Block	80387SX-16	80387SX-20
Bus Interface Unit (MHz)	16	20
Execution Unit (MHz)	16	20

Table 5-3b. Timing Requirements of Execution Unit

Pin	Symbol	Parameter	16 MHz		20 MHz		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)	Min (ns)	Max (ns)		
NUMCLK2	t1	Period	31.25	500	25	500	2.0V	5.1
NUMCLK2	t2a*	High Time	7		6		2.0V	
NUMCLK2	t2b*	High Time	3		3		$V_{CC} - 0.8V$	
NUMCLK2	t3a*	Low Time	7		6		2.0V	
NUMCLK2	t3b*	Low Time	5		4		0.8V	
NUMCLK2	t4*	Fall Time		8		8	From $V_{CC} - 0.8V$ to 0.8V	(Note 1)
NUMCLK2	t5*	Rise Time		8		8	From 0.8V to $V_{CC} - 0.8V$	

NOTE:
*Guaranteed, not tested.
1. If not used (CKM = 1), tie LOW.

Table 5-3c. Timing Requirements of Bus Interface Unit

Pin	Symbol	Parameter	16 MHz (1.5V)		20 MHz (1.5V)		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)	Min (ns)	Max (ns)		
CPUCLK2	t1	Period	31.25	500	25	500	2.0V	5.1
CPUCLK2	t2a*	High Time	7		6		2.0V	
CPUCLK2	t2b*	High Time	3		3		$V_{CC} - 0.8V$	
CPUCLK2	t3a*	Low Time	7		6		2.0V	
CPUCLK2	t3b*	Low Time	5		4		0.8V	
CPUCLK2	t4*	Fall Time		8		8	From $V_{CC} - 0.8V$ to 0.8V	
CPUCLK2	t5*	Rise Time		8		8	From 0.8V to $V_{CC} - 0.8V$	
CPUCLK2/ NUMCLK2		Ratio	10/16	14/10				
$\overline{\text{READYO}}$	t7	Out Delay	4	34	3	31	$C_L = 75$ pF	5.2
$\overline{\text{READYO}}$	t7	Out Delay	4	31	3	27	$C_L = 25$ pF**	
PEREQ	t7	Out Delay	5	34	5	34	$C_L = 75$ pF	
$\overline{\text{BUSY}}$	t7	Out Delay	5	34	5	29	$C_L = 75$ pF	
$\overline{\text{ERROR}}$	t7	Out Delay	5	34	5	34	$C_L = 75$ pF	
D15–D0	t8	Out Delay	1	54	1	54	$C_L = 120$ pF	5.3
D15–D0	t10	Setup Time	11		11			
D15–D0	t11	Hold Time	11		11			
D15–D0	t12***	Float Time	6	33	6	27	$C_L = 120$ pF	
PEREQ	t13***	Float Time	1	60	1	50	$C_L = 75$ pF	5.5
$\overline{\text{BUSY}}$	t13***	Float Time	1	60	1	50	$C_L = 75$ pF	
$\overline{\text{ERROR}}$	t13***	Float Time	1	60	1	50	$C_L = 75$ pF	
$\overline{\text{READYO}}$	t13***	Float Time	1	60	1	50	$C_L = 75$ pF	
$\overline{\text{ADS}}$	t14	Setup Time	26		21			5.3
$\overline{\text{ADS}}$	t15	Hold Time	4		4			
W/$\overline{\text{R}}$	t14	Setup Time	26		21			
W/$\overline{\text{R}}$	t15	Hold Time	4		4			
$\overline{\text{READY}}$	t16	Setup Time	19		12			5.3
$\overline{\text{READY}}$	t17	Hold Time	4		3			
$\overline{\text{CMD0}}$	t16	Setup Time	21		19			
$\overline{\text{CMD0}}$	t17	Hold Time	2		2			
$\overline{\text{NPS1}}$, NPS2	t16	Setup Time	21		19			
$\overline{\text{NPS1}}$, NPS2	t17	Hold Time	2		2			
STEN	t16	Setup Time	21		21			
STEN	t17	Hold Time	2		2			
RESETIN	t18	Setup Time	13		11			5.4
RESETIN	t19	Hold Time	3		3			

NOTES:
*Guaranteed, not tested.
**Not tested at 25 pf.
***Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.

These AC Characteristics are preliminary and subject to change.

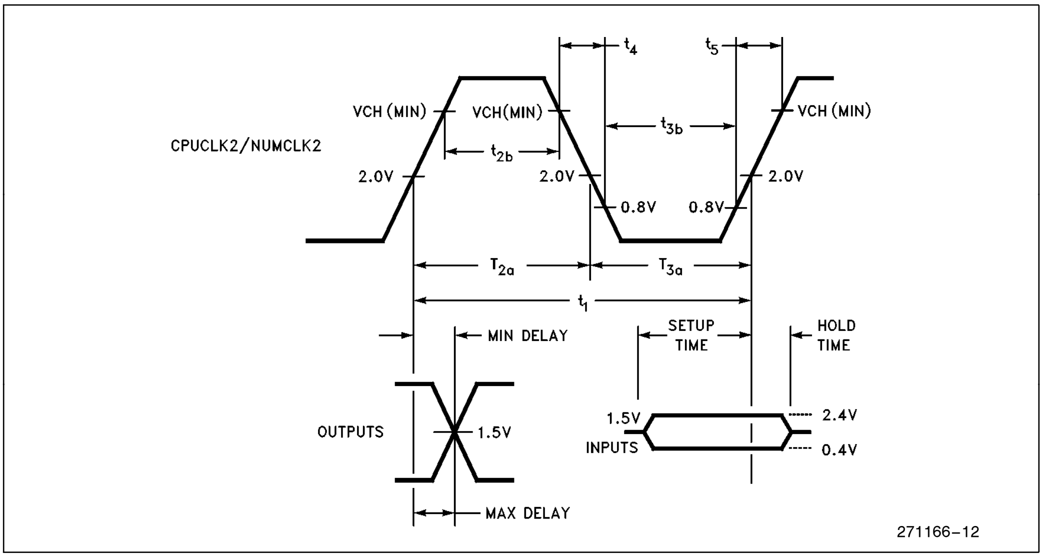


Figure 5-1. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output

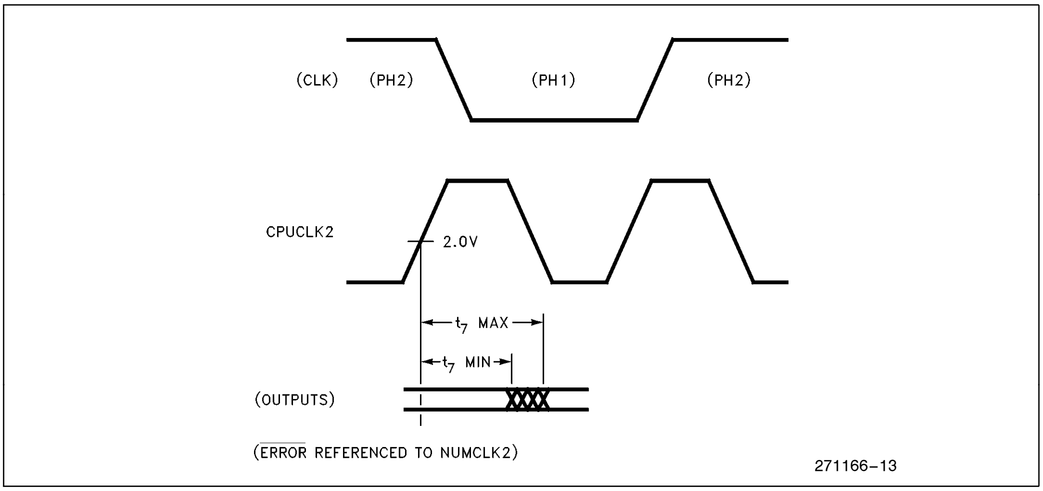


Figure 5-2. Output Signals

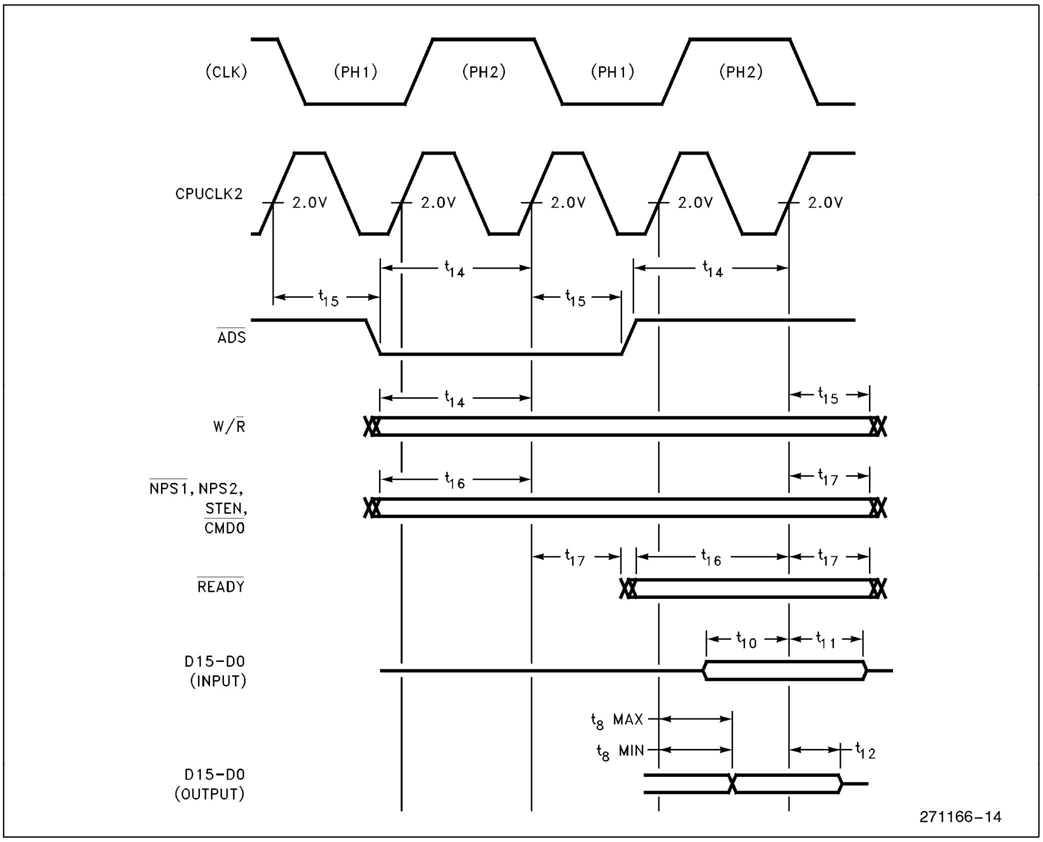
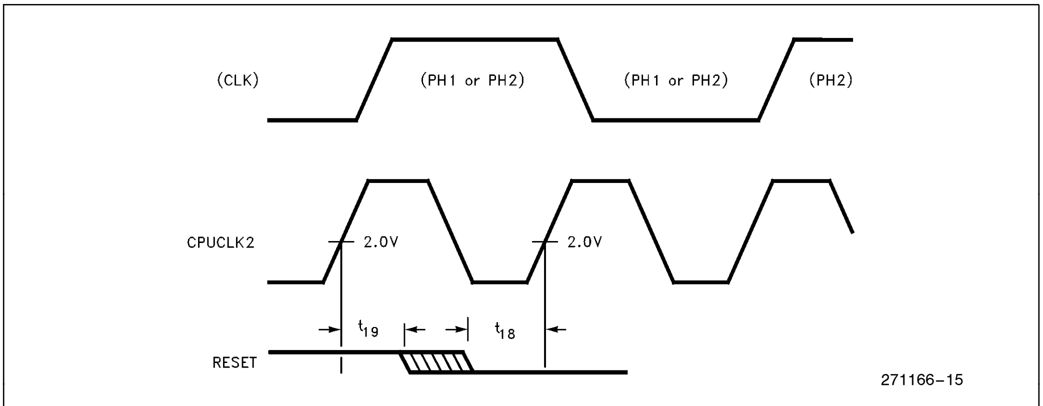


Figure 5-3. Input and I/O Signals



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

Figure 5-4. RESET Signal

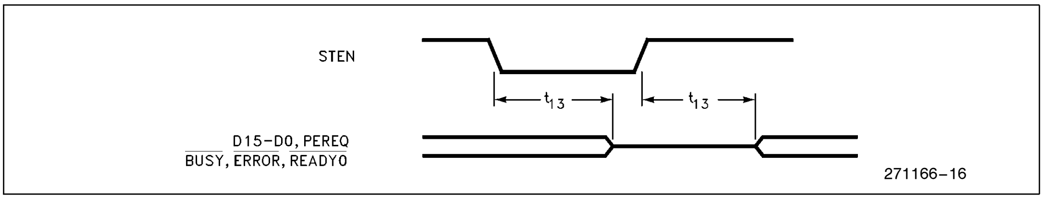
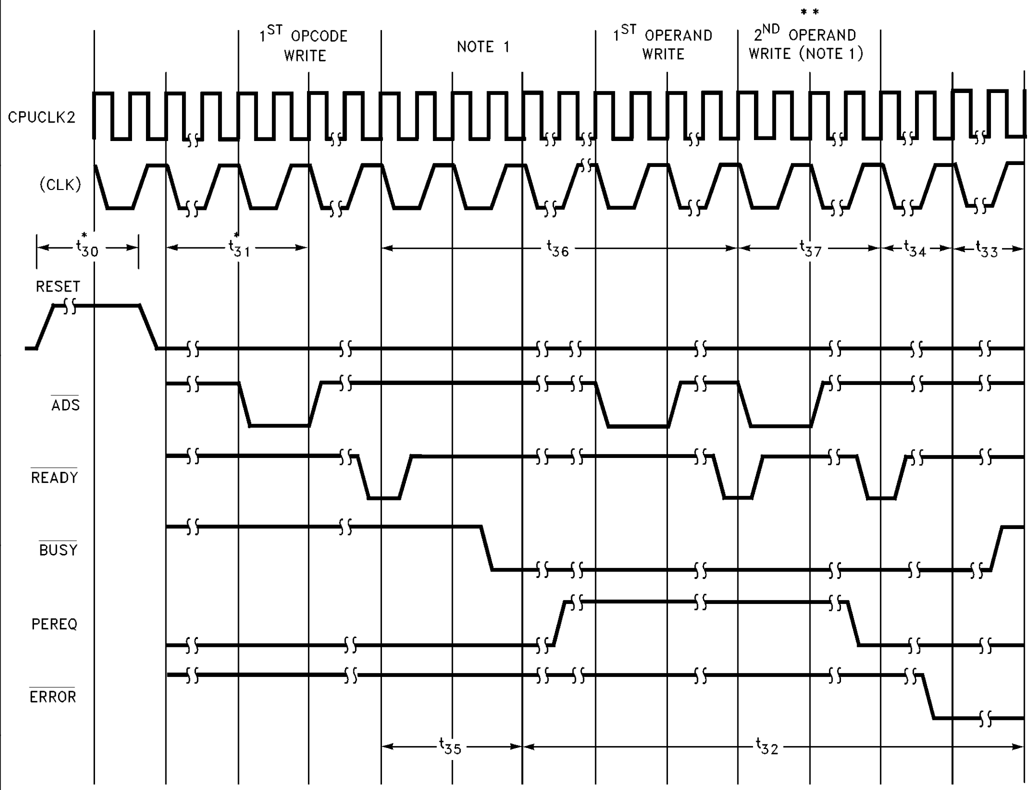

STEN
t_{13}
t_{13}
D15–D0, PEREQ
$\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$, $\overline{\text{READYO}}$
271166–16

Figure 5-5. Float from STEN

Table 5-4. Other Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN inactive to 1st opcode write	50		NUMCLK2
$\overline{\text{BUSY}}$	t32	Duration	6		CPUCLK2
$\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$	t33	$\overline{\text{ERROR}}$ (in)active to $\overline{\text{BUSY}}$ inactive	6		CPUCLK2
PEREQ, $\overline{\text{ERROR}}$	t34	PEREQ inactive to $\overline{\text{ERROR}}$ active	6		CPUCLK2
$\overline{\text{READY}}$, $\overline{\text{BUSY}}$	t35	$\overline{\text{READY}}$ active to $\overline{\text{BUSY}}$ active	4	4	CPUCLK2
$\overline{\text{READY}}$	t36	Minimum time from opcode write to opcode/operand write	4		CPUCLK2
$\overline{\text{READY}}$	t37	Minimum time from operand write to operand write	4		CPUCLK2



* In NUMCLK2's
** or last operand

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

Figure 5-6. Other Parameters

6.0 i387™ SX NPX EXTENSIONS TO THE CPU'S INSTRUCTION SET

Instructions for the i387 SX NPX assume one of the five forms shown in Table 6-1. 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 CPU's addressing modes.

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

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 requests 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.

Table 6-1. Instruction Formats

	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		

OP = Instruction opcode, possibly 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
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
*In FSUB and FDIV, the low-order bit of OPB is the R (reversed) bit
P = POP
0—Do not pop stack
1—Pop stack after operation
ESC = 11011
ST(i) = Register stack element i
000 = Stack top
001 = Second stack element
.
.
.
111 = Eighth stack element

i387™ SX NPX Extension to the i386™ SX Microprocessor 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	24	49–56	33	61–65
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP		64–75		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP		52		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP		274–283		
ST(i) to ST(0)	ESC 001	11000 ST(i)			14		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	49	84–98	55	82–95
ST(0) to ST(i)	ESC 101	11010 ST(i)			11		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	49	84–98	55	82–95
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP		90–107		
ST(0) to extended real	ESC 011	MOD 111 R/M	SIB/DISP		63		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP		522–544		
ST(0) to ST(i)	ESC 101	11011 ST (i)			12		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			18		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	30	60–67	39	71–75
ST(i) to ST(0)	ESC 000	11010 ST(i)			24		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	SIB/DISP	30	60–67	39	71–75
ST(i) to ST(0)	ESC 000	11011 ST(i)			26		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			26		
FTST = Test ST(0)	ESC 001	1110 0100			28		
FUCOM = Unordered compare	ESC 101	11100 ST(i)			24		
FUCOMP = Unordered compare and pop	ESC 101	11101 ST(i)			26		
FUCOMPP = Unordered compare and pop twice	ESC 010	1110 1001			26		
FXAM = Examine ST(0)	ESC 001	11100101			30-38		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)	ESC 001	1110 1110			20		
FLD1 = Load +1.0 into ST(0)	ESC 001	1110 1000			24		
FLDPI = Load pi into ST(0)	ESC 001	1110 1011			40		
FLDL2T = Load $\log_2(10)$ into ST(0)	ESC 001	1110 1001			40		

Shaded areas indicate instructions not available in 8087/80287.

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

i387™ SX NPX Extension to the i386™ SX Microprocessor 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		40			
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100		41			
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101		41			
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	28–36	61–76	37–45	71–85
ST(i) and ST(0)	ESC d P 0	11000 ST(i)		23–31[b]			
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	28–36	61–76	36–44	71–83[c]
ST(i) and ST(0)	ESC d P 0	1110 R R/M		26–34[d]			
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	SIB/DISP	31–39	65–86	40–65	76–87
ST(i) and ST(0)	ESC d P 0	1100 1 R/M		29–57[e]			
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	93	124–131[f]	102	136–140[g]
ST(i) and ST(0)	ESC d P 0	1111 R R/M		88[h]			
FSQRT[i] = Square root	ESC 001	1111 1010		122–129			
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101		67–86			
FPREM = Partial remainder	ESC 001	1111 1000		74–155			
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101		95–185			
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100		66–80			
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100		70–76			
FABS = Absolute value of ST(0)	ESC 001	1110 0001		22			
FCHS = Change sign of ST(0)	ESC 001	1110 0000		24–25			

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. $-0 \leq \text{ST}(0) \leq +\infty$.

i387™ SX NPX Extension to the i386™ SX Microprocessor 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		123–772[j]
FPTAN[k] = Partial tangent of ST(0)	ESC 001	1111 0010		191–497[j]
FPATAN = Partial arctangent	ESC 001	1111 0011		314–487
FSIN[k] = Sine of ST(0)	ESC 001	1111 1110		122–771[j]
FSINCOS[k] = Sine and cosine of ST(0)	ESC 001	1111 1011		194–809[j]
F2XM1[l] = $2^{ST(0)} - 1$	ESC 001	1111 0000		211–476
FYL2X[m] = ST(1) * $\log_2$(ST(0))	ESC 001	1111 0001		120–538
FYL2XP1[n] = ST(1) * $\log_2$(ST(0) + 1.0)	ESC 001	1111 1001		257–547
PROCESSOR CONTROL				
FINIT = Initialize NPX	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	475–476
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	388
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. $0 \leq |ST(0)| < (2 - SQRT(2))/2, -\infty < ST(1) < +\infty$.

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 NPX and the 8087 NPX, exception-handling routines *may* need to be changed.

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

1. The NPX signals exceptions through a dedicated $\overline{\text{ERROR}}$ line to the 80286. The NPX 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 $\overline{\text{BUSY}}$ 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.