# Introduction

The SuperMathDX coprocessor is a high-performance industry compatible floating-point mathematic coprocessor. It is 100 percent pin compatible and 100 percent software compatible with the industry standard 80387 while achieving performance improvements of up to 600 percent over the 80387. When installed in an 80386-based system, the SuperMathDX coprocessor executes add, subtract, multiply, divide, and transcendental floating-point operations magnitudes faster than a standalone 80386.

The SuperMathDX coprocessor is 100 percent code compatible with the 80387. Therefore, it flawlessly runs all software that has been assembled and/or compiled for the industry standard 80387.

In addition to maintaining industry standard compatibility, the SuperMathDX coprocessor is engineered to comply with the architecture mandated by the full extended double precision floating-point math IEEE-754-1985 specification. Full compliance with both the industry standard 80387 and the IEEE specification guarantees that the SuperMathDX coprocessor is compatible with all current and software and will be compatible with all future application software.

# System Architecture

The SuperMathDX coprocessor is fabricated in a 1.2μ double-metal CMOS process. It is available at operating speeds of up to 40MHz and is packaged in a 68 Pin Grid Array (PGA).

The internal structure of the SuperMathDX coprocessor can be divided into four major subsystems:

- System Interface Unit
- Mathematical Logic Unit
- Mathematical Control Unit
- Power Management Unit.

The System Interface Unit coordinates the communication and transfer of code and data between the system and the coprocessor.

The Mathematical Logic Unit performs all of the floating-point mathematic operations, including data conversions, data normalizations, and data roundings.

The Mathematical Control Unit supervises the operation of the Mathematical Logic Unit, coordinates the execution of complex calculations, and controls the data flow both to and from the System Interface Unit.

The Power Management Unit is transparent to application software. This unit monitors the activity levels within the functional block and "turns off" idle areas. This reduces power consumption and heat dissipation, positioning the SuperMathDX coprocessor well for battery-powered environments.

## Registers

The SuperMathDX coprocessor is register compatible with the 80387. It contains eight 80-bit data registers, which are accessed in a stack fashion: a data register tag word, a status register, and a control register.

In addition, the SuperMathDX coprocessor relies on registers in the 80386 compatible CPU, which contain pointers to:

- The memory address containing the SuperMathDX FPU's current instruction.
- The memory address containing any operand associated with the SuperMathDX FPU instruction.

## Instruction Set

The binary instructions and data format used by the SuperMathDX coprocessor are fully compatible with those defined for the industry standard 80387. The instructions available to the user include the standard mathematic instructions, transcendental function instructions, data load and store instructions, and SuperMathDX coprocessor initialization and control instructions. Again, because these instructions are identical to the 80387 instructions, the SuperMathDX coprocessor flawlessly runs all software that has been assembled and/or compiled for the 80387.

## Accuracy

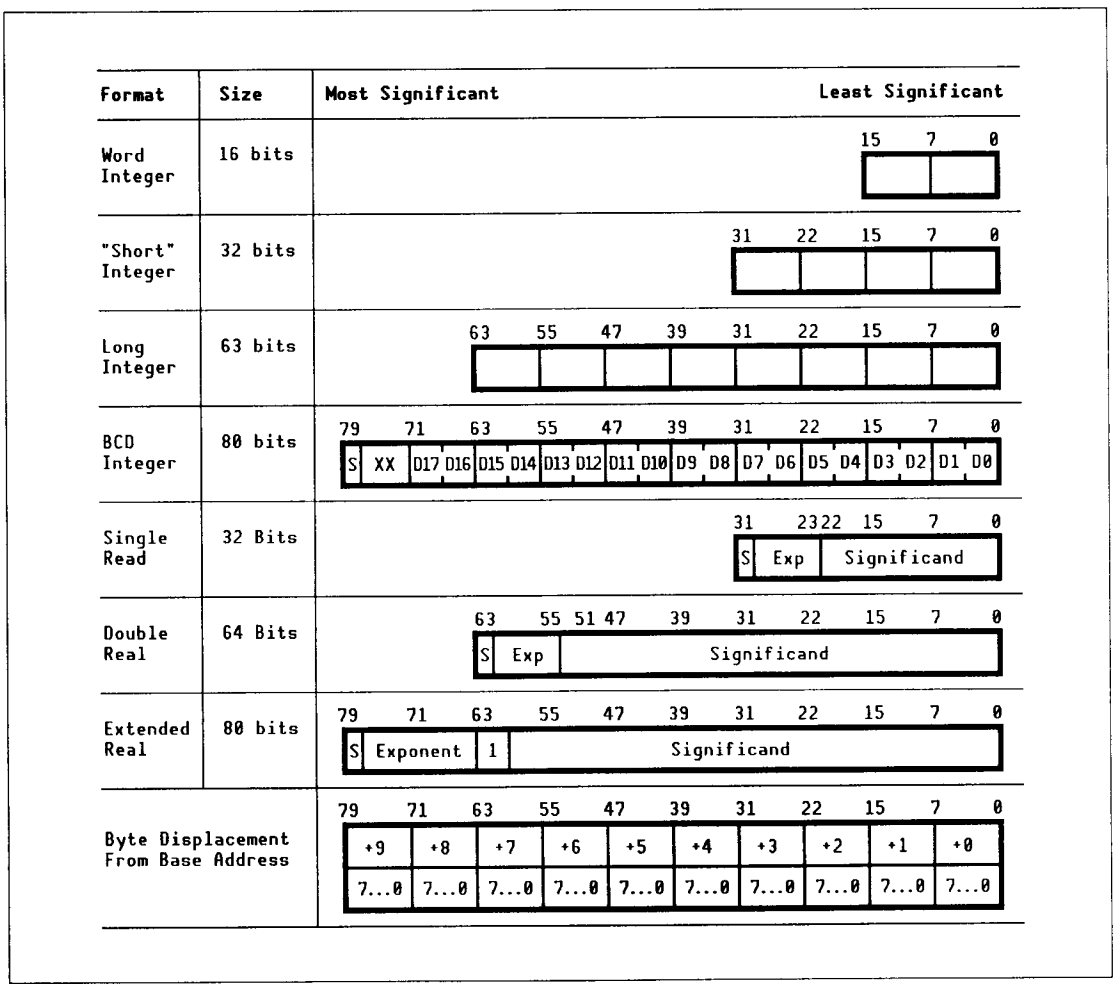
The SuperMathDX coprocessor supports all of the data formats mandated by the IEEE-754-1985 specification, along with other industry compatible formats. Figure 2 lists the data types and shows how they are represented in memory.

Because there is no "functional" one-to-one mapping between real numbers and the representation of real numbers in a discretely defined numbering system, the degree

of accuracy in representing real numbers in a chosen discrete numbering system is a significant issue.

The IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754-1985) specifies error bounds for floating-point calculations and values. This standard specifies both the allowable error in a given arithmetic computation and the rounding algorithms. The SuperMathDX coprocessor fully complies with the IEEE specification. Therefore, it performs calculations to the levels of accuracy specified in the IEEE document.

***Figure 2.*** *Memory Data Formats*



## Interface Signals

The SuperMathDX coprocessor maintains its high-speed data link with the 80386 DX compatible processor through a signal interface that is based on standard 80386 bus signals as well as signals dedicated specifically to the SuperMathDX microprocessor.

# Component Pin Assignment

The SuperMathDX coprocessor is packaged in a 68 Pin Grid Array (PGA) shown in Figures 3 and 4.

***Figure 3.*** *SuperMathDX 68-Pin PGA Pin Diagram, Top Side*

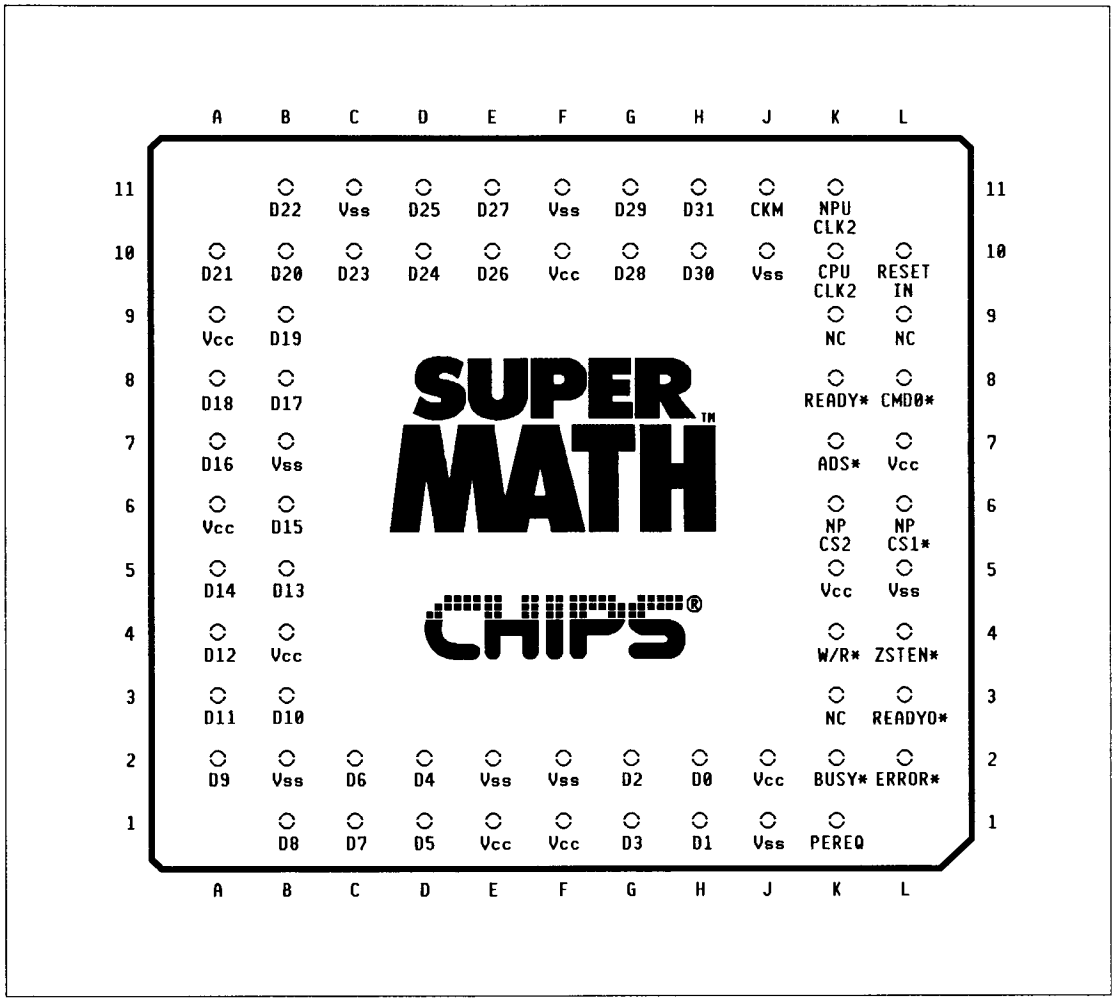
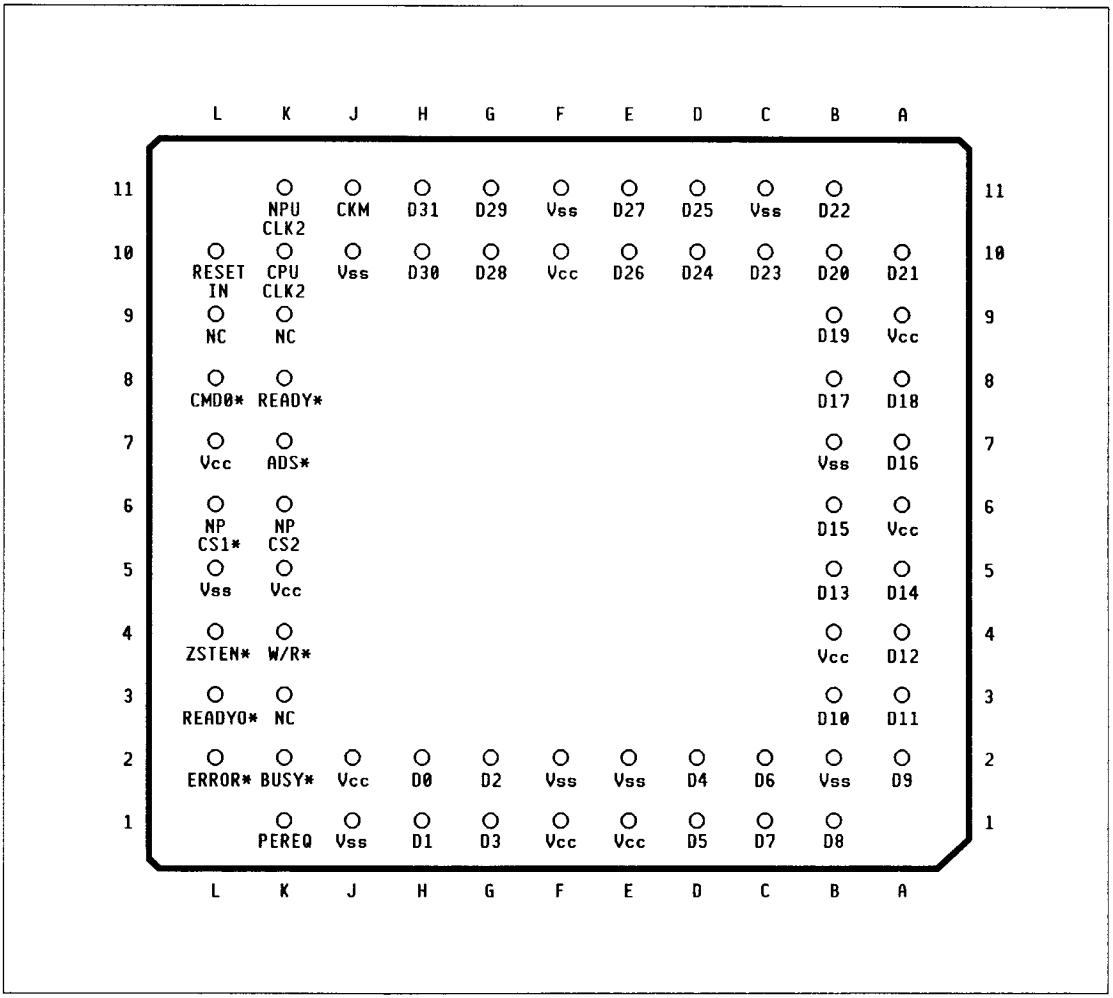


***Figure 4.*** *SuperMathDX 68-Pin PGA Pin Diagram, Pin Side*



The signal assignments to the component pins are shown in Table 1.

**Table 1.** *Assignment of Signals to Component Pins*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| D31 | H11 | I/O | Data Line 31 |
| D30 | H10 | I/O | Data Line 30 |
| D29 | G11 | I/O | Data Line 29 |
| D28 | G10 | I/O | Data Line 28 |
| D27 | E11 | I/O | Data Line 27 |
| D26 | E10 | I/O | Data Line 26 |
| D25 | D11 | I/O | Data Line 25 |
| D24 | D10 | I/O | Data Line 24 |
| D23 | C10 | I/O | Data Line 23 |
| D22 | B11 | I/O | Data Line 22 |
| D21 | A10 | I/O | Data Line 21 |
| D20 | B10 | I/O | Data Line 20 |
| D19 | B9 | I/O | Data Line 19 |
| D18 | A8 | I/O | Data Line 18 |
| D17 | B8 | I/O | Data Line 17 |
| D16 | A7 | I/O | Data Line 16 |
| D15 | B6 | I/O | Data Line 15 |
| D14 | A5 | I/O | Data Line 14 |
| D13 | B5 | I/O | Data Line 13 |
| D12 | A4 | I/O | Data Line 12 |
| D11 | A3 | I/O | Data Line 11 |
| D10 | B3 | I/O | Data Line 10 |
| D9 | A2 | I/O | Data Line 9 |
| D8 | B1 | I/O | Data Line 8 |
| D7 | C1 | I/O | Data Line 7 |
| D6 | C2 | I/O | Data Line 6 |
| D5 | D1 | I/O | Data Line 5 |
| D4 | D2 | I/O | Data Line 4 |
| D3 | G1 | I/O | Data Line 3 |
| D2 | G2 | I/O | Data Line 2 |
| D1 | H1 | I/O | Data Line 1 |
| D0 | H2 | I/O | Data Line 0 |

*Table 1. Assignment of Signals to Component Pins (continued)*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| ADS* | K7 | I | Address Strobe |
| BUSY* | K2 | O | Coprocessor Busy |
| CKM | J11 | N/C | Clock Mode Select (ignored by the SuperMathDX coprocessor) |
| CPUCLK2 | K10 | I | CPU Clock |
| CMD0* | L8 | I | Command |
| ERROR* | L2 | O | Error |
| NPCS1* | L6 | I | Chip Select 1 |
| NPCS2 | K6 | I | Chip Select 2 |
| NPUCLK2 | K11 | N/C | Alternate Clock (ignored by the SuperMathDX coprocessor) |
| PEREQ | K1 | O | Processor Extension Request |
| READY* | K8 | I | System Ready |
| READYO* | L3 | O | Ready Output |
| RESETIN | L10 | I | Reset |
| W/R* | K4 | I | Write/Read |
| ZSTEN* | L4 | I | Z-State Enable |
| Vcc | | | Pins A6, A9, B4, E1, F1, F10, J2, K5, and L7 |
| Vss | | | Pins B2, B7, C11, E2, F2, F11, J1, J10, and L5 |
| Not used | | | Pins J11 and K11 |
| No connect | | | Pins K3, K9, and L9 |

# Signal Description

Signal descriptions are summarized in Table 2.

*Table 2. Signal Descriptions*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| **Control and Data Bus Signals** | | | |
| ADS* | K7 | I | Address Strobe<br>An input that qualifies the control input signals (CMD0*, NPCS1*, NPCS2, and W/R*) from the system. After ADS* goes active, the SuperMathDX coprocessor samples the control signals. This signal is normally connected to the CPU's ADS* output. |
| CMD0* | L8 | I | Command<br>An input that distinguishes between a command port access (CMD0* active) and a data port access (CMD0* inactive). ADS* and CPUCLK2 qualify CMD0* as valid. This signal is normally connected to the CPU's A2 address line. |
| NPCS1* | L6 | I | Chip Select 1<br>This input, along with NPCS2, indicates that the system is selecting the SuperMathDX coprocessor for data transfer. ADS*, CPUCLK2, and NPCS2 qualify NPCS1* as valid. This signal is normally connected to the CPU's M/IO* output. |
| NPCS2 | K6 | I | Chip Select 2<br>This input, along with NPCS1*, indicates that the system is selecting the SuperMathDX coprocessor for data transfer. ADS*, CPUCLK2, and NPCS1* qualify NPCS2 as valid. This signal is normally connected to the CPU's A31 address line. |
| W/R* | K4 | I | Write/Read<br>An input that distinguishes between a read or write bus cycle. ADS* and CPUCLK2 qualify W/R* as valid. This signal is normally connected to the CPU's W/R* output. |

*Table 2. Signal Descriptions (continued)*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| **Control and Data Bus Signals, continued** | | | |
| D31 | H11 | I/O | 32-Bit Data Bus<br>This bus transfers commands, data, and status information between the system and the SuperMathDX coprocessor. CPUCLK2 qualifies the data on these lines as valid; the control input signals (CMD0*, NPCS1*, and W/R*) indicate the type of information.<br><br>This bus is normally connected to the CPU's data bus (D31:0). |
| D30 | H10 | I/O | |
| D29 | G11 | I/O | |
| D28 | G10 | I/O | |
| D27 | E11 | I/O | |
| D26 | E10 | I/O | |
| D25 | D11 | I/O | |
| D24 | D10 | I/O | |
| D23 | C10 | I/O | |
| D22 | B11 | I/O | |
| D21 | A10 | I/O | |
| D20 | B10 | I/O | |
| D19 | B9 | I/O | |
| D18 | A8 | I/O | |
| D17 | B8 | I/O | |
| D16 | A7 | I/O | |
| D15 | B6 | I/O | |
| D14 | A5 | I/O | |
| D13 | B5 | I/O | |
| D12 | A4 | I/O | |
| D11 | A3 | I/O | |
| D10 | B3 | I/O | |
| D9 | A2 | I/O | |
| D8 | B1 | I/O | |
| D7 | C1 | I/O | |
| D6 | C2 | I/O | |
| D5 | D1 | I/O | |
| D4 | D2 | I/O | |
| D3 | G1 | I/O | |
| D2 | G2 | I/O | |
| D1 | H1 | I/O | |
| D0 | H2 | I/O | |

**Table 2.** *Signal Descriptions (continued)*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| **Clock Signals** | | | |
| CKM | J11 | N/C | Clock Mode Select<br>If an input is connected to this signal, it will be ignored by the SuperMathDX coprocessor. This signal is normally connected to Vcc. |
| CPUCLK2 | K10 | I | CPU Clock<br>This input provides the clock synchronization between the 80386-compatible CPU and the SuperMathDX coprocessor. This clock also provides the timing for internal operations. CPUCLK2 must be connected to the same clock source used by the CPU. |
| NPUCLK2 | K11 | N/C | Alternate Clock<br>If an input is connected to this signal, it will be ignored by the SuperMathDX coprocessor. This signal is normally left unconnected for typical applications. |
| **SuperMathDX Status Signals** | | | |
| BUSY* | K2 | O | Coprocessor Busy<br>This signal's output indicates that the SuperMathDX coprocessor is busy executing an instruction. The status of BUSY* is valid only when qualified by CPUCLK2. CPUCLK2 is typically connected to either the CPU's BUSY* input or the chipset's BUSY* input. |
| ERROR* | L2 | O | Error<br>This output indicates the status of the ES bit in the Status Register. ERROR* is valid before the SuperMathDX coprocessor sets BUSY* inactive. ERROR* is also set active immediately after a system reset (to indicate the presence of the SuperMathDX coprocessor to the CPU). This signal is typically connected to either the CPU's ERROR* input or the chipset's ERROR* input. |
| PEREQ | K1 | O | Processor Extension Request<br>This output indicates that the SuperMathDX coprocessor requires the CPU to perform a data transfer. PEREQ is active for as long as the SuperMathDX coprocessor requires further data transfers. PEREQ can only be active when BUSY* is active. This signal is normally connected to the CPU's PEREQ input. |

*Table 2. Signal Descriptions (continued)*

| Signal Name | Pin No. | Type | Description |
|---|---|---|---|
| **Miscellaneous SuperMathDX Signals** | | | |
| RESETIN | L10 | I | SuperMathDX Reset<br>This input instructs the SuperMathDX coprocessor to reset itself. RESETIN must remain active for at least 20 CPUCLK2 cycle periods and must be synchronized with CPUCLK2 when it goes inactive. After RESETIN goes inactive, the first SuperMathDX coprocessor access must be delayed by at least 8 CPUCLK2 cycle periods. During reset, ERROR* is set to active (to indicate the presence of the SuperMathDX coprocessor to the CPU). This signal is normally connected to the same source that drives the CPU's reset input. |
| READY* | K8 | I | System Ready<br>When active, this signal, along with ADS*, indicates to the SuperMathDX coprocessor that the CPU's bus cycle is about to terminate. This signal is normally connected to the same source which drives the CPU's READY* input. |
| READYO* | L3 | O | Ready Output<br>This output indicates that the SuperMathDX coprocessor is about to complete a bus cycle. This signal is typically connected to either the CPU's READY* input or the chipset's READY* input. |
| ZSTEN* | L4 | I | Z-State Enable<br>When active, this pin tri-states the SuperMathDX coprocessor's BUSY*, ERROR*, PEREQ, READYO*, and D31:0 outputs. (Also, during a ZSTEN* active input, all other inputs are ignored.) Pulling this input low isolates the SuperMathDX coprocessor from the rest of the board circuitry, facilitating board-level testing. This signal is normally pulled-up to Vcc through a resistor. |

Table 3 shows the values of CMD0* and W/R* during various bus cycles.

*Table 3. SuperMathDX Bus Cycle Indication*

| Bus Cycle[1] | CMD0* | W/R* |
|---|---|---|
| Control/Status Read from the SuperMathDX coprocessor | 0 | 0 |
| Write Opcode to the SuperMathDX coprocessor | 0 | 1 |
| Read Data from the SuperMathDX coprocessor | 1 | 0 |
| Write Data to the SuperMathDX coprocessor | 1 | 1 |

[1] If NPS1* = 0 and NPS2 = 1 is not true, the bus cycle is not for the SuperMathDX coprocessor and the CMD0* and W/R* signal states are don't care. Also, ZSTEN* must be inactive (=1).

# Bus Operation

The SuperMathDX coprocessor supports the numeric coprocessor interface provided by 80386-compatible CPUs running in both pipelined and nonpipelined formats. The SuperMathDX coprocessor performs data exchanges with the CPU with wait state accesses.

The system interface activity depends on the instruction being executed. The protocol varies considerably for different types of instructions. Generally, the SuperMathDX coprocessor performs handshaking with the CPU through the BUSY*, ERROR* and PEREQ signals. The CPU provides dedicated pins to accomodate these signals. In this way, the SuperMathDX coprocessor can generate the READYO* signal immediately after receiving the ADS* signal, thus freeing the CPU buses for DMA cycles while the SuperMathDX coprocessor is executing an instruction.

## ERROR* Checking Protocol

Before initiating any SuperMathDX FPU operation, the CPU must check the ERROR* signal according to a specific protocol. The CPU first waits for BUSY* to go inactive, then checks ERROR* and proceeds as follows:

- If the CPU sees ERROR* active, it generates a coprocessor exception and executes the appropriate routine.
- If the CPU sees ERROR* inactive, it writes the instruction opcode or data to the SuperMathDX coprocessor and continues program execution.

The descriptions which follow refer to the ERROR* checking protocol.

## Instructions with No Operand Transfer

For the most basic SuperMathDX FPU instructions, no external information transfer is required. Table 5 lists these instructions.

The CPU follows the ERROR* checking protocol described above to write the instruction opcode. Once the opcode is written, the SuperMathDX coprocessor sets BUSY* active and begins executing the instruction. If an exception occurs and the exception is enabled, the SuperMathDX coprocessor sets ERROR* active. This type of operation occurs in only one bus cycle and does not involve the PEREQ signal.

One instruction, FINIT, does not require the CPU to check BUSY*. The CPU can send its opcode regardless of the state of BUSY* (although the SuperMathDX coprocessor will set BUSY* active upon receipt of this instruction).

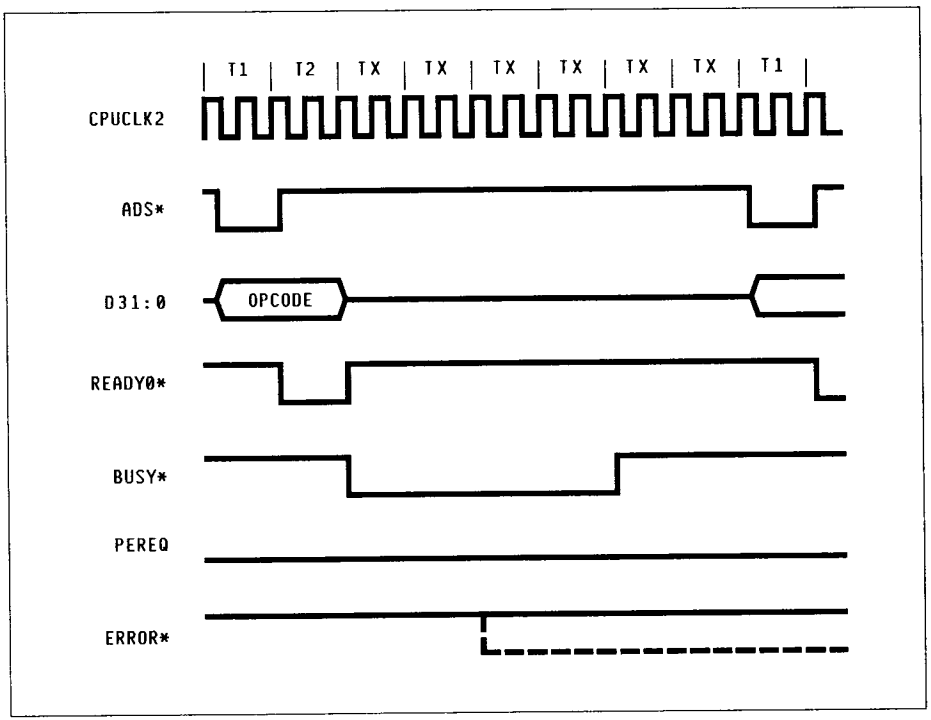
**Table 4.** *Instructions With No Operand Transfer*

| Instruction | Instruction | Instruction |
|---|---|---|
| F2XM1 | FLD1 | FSCALE |
| FABS | FLDL2E | FSIN |
| FADD[1] | FLDL2T | FSINCOS |
| FCHS | FLDLG2 | FSQRT |
| FCOM[1] | FLDLN2 | FSUB[1] |
| FCOMP | FLDPI | FSUBR[1] |
| FCOMPP | FLDZ | FTST |
| FCOS | FMUL[1] | FUCOM |
| FDECSTP | FNOP | FUCOMP |
| FDIV[1] | FPATAN | FUCOMPP |
| FDIVR[1] | FPREM | FXAM |
| FFREE | FPREM1 | FXCH[1] |
| FINCSTP | FPTAN | FXTRACT |
| FLD[1] | FRNDINT | FYL2X |
| FYL2XP1 | | |

[1] Register-to-register operations only.

Figure 5 shows an example of the timing for this type of operation. FINIT does not execute in the manner depicted in Figure 5.

***Figure 5.*** *Timing for Instructions With No Operand Transfer*




Example Instructions:

| | | | | | |
|---|---|---|---|---|---|
| FADD | reg,reg | FFREE | FCOS | FSCALE | FABS |
| FCOM | reg,reg | FLD1 | FDECSTP | FSIN | FCHS |
| FDIV | reg,reg | FLDL2E | FINCSTP | FSINCOS | FNOP |
| FMUL | reg,reg | FLDLTE | FPATAN | FSQRT | FTST |
| FSUB | reg,reg | FLDLG2 | FPREM | FXTRACT | FXAM |
| FUCOMP | reg,reg | FLDLN2 | FPREM1 | FYL2X | |
| FLD | reg,reg | FLDPI | FPTAN | FYL2XP1 | |
| FST | reg,reg | FLDZ | | | |
| FXCH | reg,reg | | | | |

## Instructions That Transfer Data to the SuperMathDX Coprocessor

Certain instructions require a single operand transfer from the CPU to the SuperMathDX coprocessor. Table 5 lists these instructions. The CPU need not wait for BUSY* to go inactive in this case; it simply writes the instruction opcode to the SuperMathDX microprocessor. It then must wait for BUSY* to go inactive and follow the ERROR* checking protocol described above before writing the operand. This operation requires 2 bus cycles for 32-bit operands, and 3 bus cycles for 64-bit operands.

Once the operand is written, the SuperMathDX coprocessor sets BUSY* active and begins executing the instruction. If an exception occurs and is enabled, the SuperMathDX coprocessor sets ERROR* active. The PEREQ line is not used.

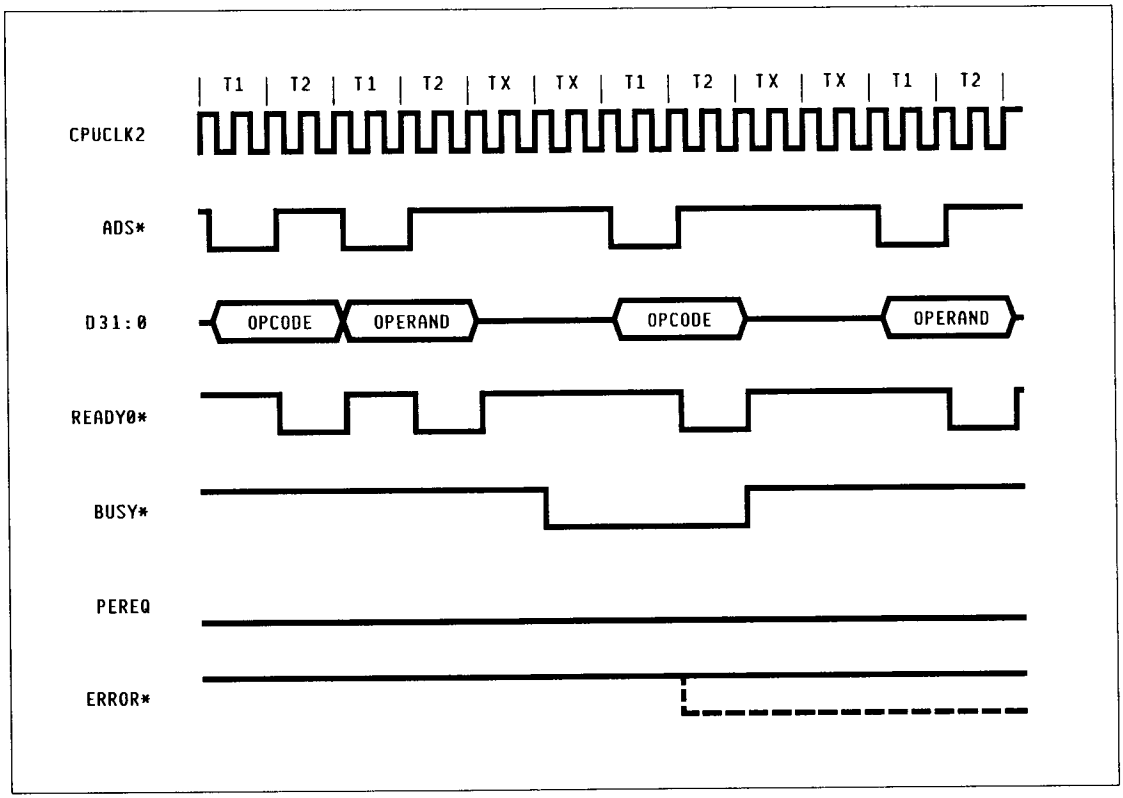
**Table 5.** *Instructions That Transfer Data to the SuperMathDX Coprocessor*

| Instruction | Instruction | Instruction |
|---|---|---|
| FLD[1] | FCOMP | FMUL |
| FADD | FDIV | FSUB |
| FCOM | FDIVR | FSUBR |

[1] 32-bit and 64-bit transfers only.

Figures 6 and 7 show an example of the timing for this type of operation.

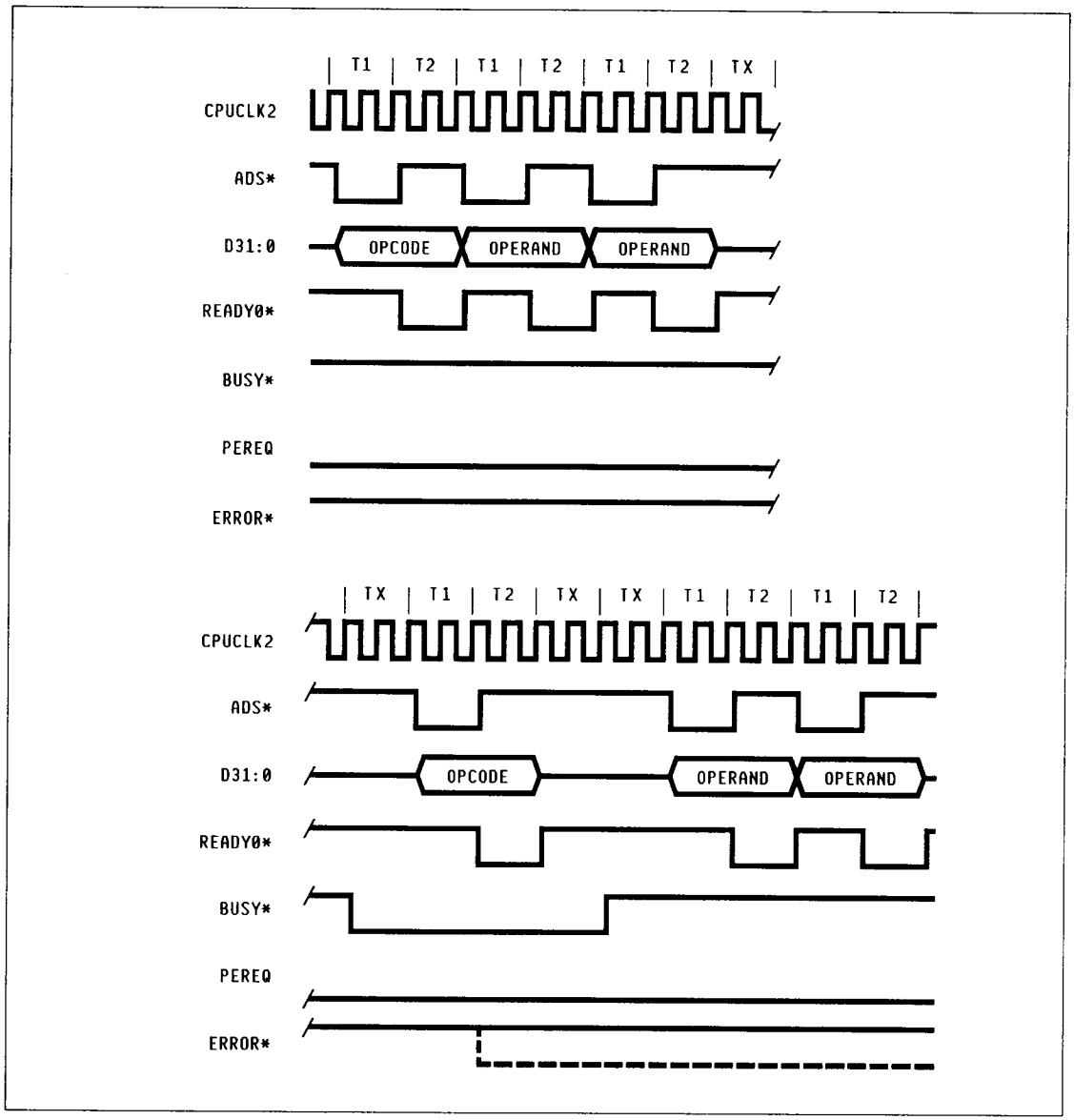
**Figure 6.** *Timing for Transfer of Instruction With 32-Bit Operand to the SuperMathDX Coprocessor*




Example Instructions:

| | | | |
|---|---|---|---|
| FLD | reg,mem (32-bit real) | FDIV | reg,mem (32-bit real) |
| FLD | reg,mem (32-bit integer) | | reg,mem (32-bit integer) |
| FADD | reg,mem (32-bit real) | FDIV | reg,mem (32-bit real) |
| | reg,mem (32-bit integer) | | reg,mem (32-bit integer) |
| | | FMUL | reg,mem (32-bit real) |
| | | | reg,mem (32-bit integer) |
| | | FSUB | reg,mem (32-bit real) |
| | | | reg,mem (32-bit integer) |

**Figure 7.** *Timing for the Transfer of Instruction With 64-Bit Operand to the SuperMathDX Coprocessor*



Example Instructions:

| | | | |
|---|---|---|---|
| FLD | reg,mem (64-bit real) | FCOM | reg,mem (64-bit real) |
| FLD | reg,mem (64-bit integer) | | reg,mem (64-bit integer) |
| FADD | reg,mem (64-bit real) | FDIV | reg,mem (64-bit real) |
| | reg,mem (64-bit integer) | | reg,mem (64-bit integer) |
| | | FMUL | reg,mem (64-bit real |
| | | | reg,mem (64-bit integer) |
| | | FSUB | reg,mem (64-bit real) |
| | | | reg,mem (64-bit integer) |

## Instructions That Use PEREQ to Synchronize Transfer

The instructions that return an operand from the SuperMathDX coprocessor, as well as instructions that write 16-bit and 80-bit operands to the SuperMathDX coprocessor, must synchronize the operand transfer with the PEREQ signal. Table 6 lists these instructions.

The CPU follows the ERROR* checking protocol described above to write the instruction opcode. It then waits for both BUSY* and PEREQ to go active before transferring the operand. BUSY* stays active, and PEREQ inactive, while the operand is converted to the proper format. PEREQ then goes active when the value is ready for transfer.

This operation requires 1 bus cycle to transfer a 16-bit word integer, 32-bit short integer, and 32-bit single-precision real operands; 2 bus cycles to transfer a 64-bit long integer and 64-bit double precision real operands; 3 bus cycles to transfer 80-bit operands; and various values for special operations as noted in Table 6.

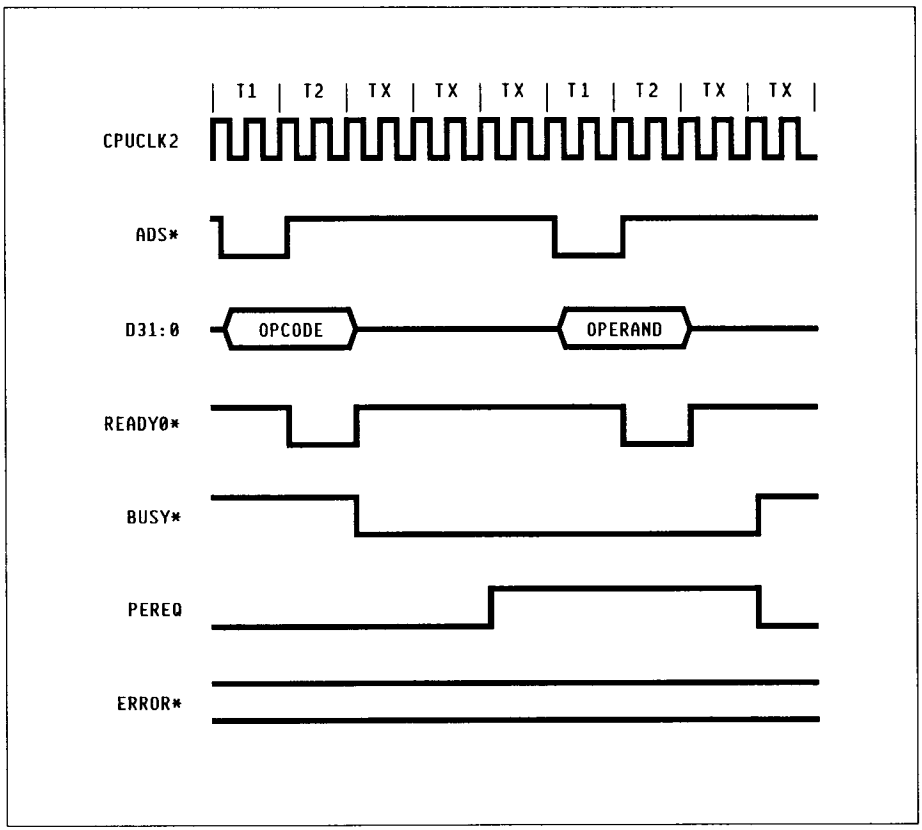
**Table 6.** *Instructions That Use PEREQ to Synchronize Transfer*

| Instruction | Bus Cycles | Instruction | Bus Cycles |
|---|---|---|---|
| FLD[1] | 2/4 | FISUBR[2] | 2 |
| FICOMP[2] | 2 | FLDCW | 2 |
| FIMUL[2] | 2 | FLDENV | 8 |
| FIADD[2] | 2 | FRSTOR | 28 |
| FIDIV[2] | 2 | FSAVE | 28 |
| FISUB[2] | 2 | FST | 2/3/4 |
| FICOM[2] | 2 | FSTENV | 8 |
| FIDIVR[2] | 2 | | |

1 80-bit operands only.
2 16-bit integers only.

Figures 8, 9, and 10 show examples of the timing for this type of operation.

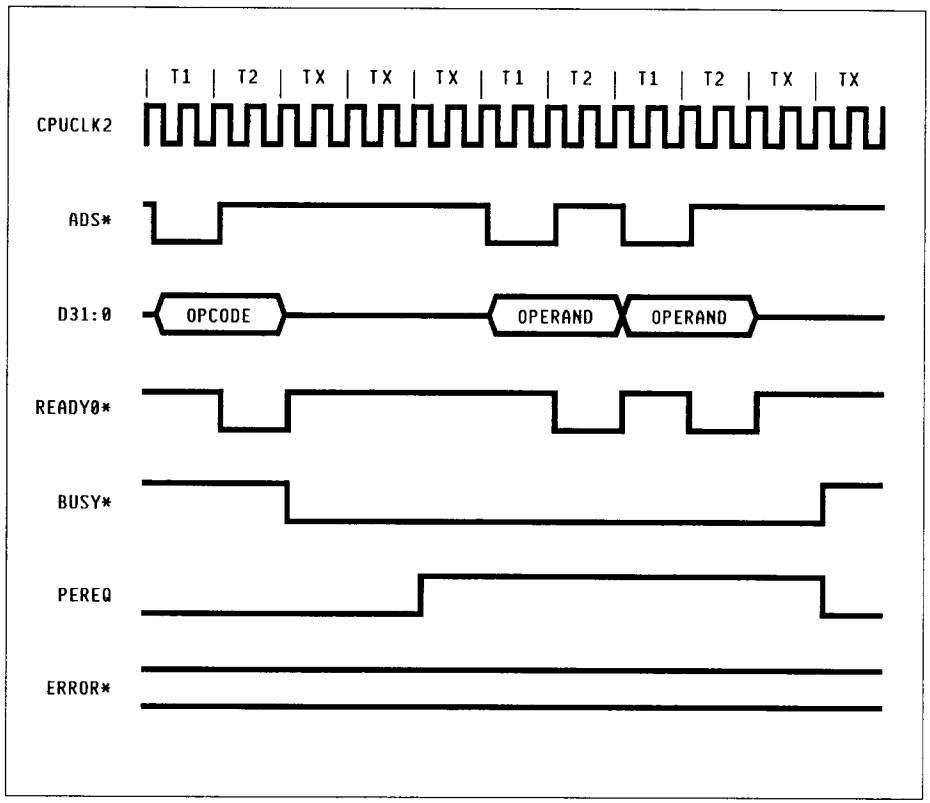
***Figure 8.*** *Timing for Transfer of Instructions That Use PEREQ to Synchronize (16 Bits)*




Example Instructions:

| | | | |
|---|---|---|---|
| FLD | reg,mem (16-bit integer) | FADD | reg,mem (16-bit integer) |
| FLDCW | | FCOM | reg,mem (16-bit integer) |
| FST | reg,mem (32-bit real) | FDIV | reg,mem (16-bit integer) |
| FST | reg,mem (32-bit integer) | FMUL | reg,mem (16-bit integer) |
| FST | reg,mem (16-bit integer) | FSUB | reg,mem (16-bit integer) |

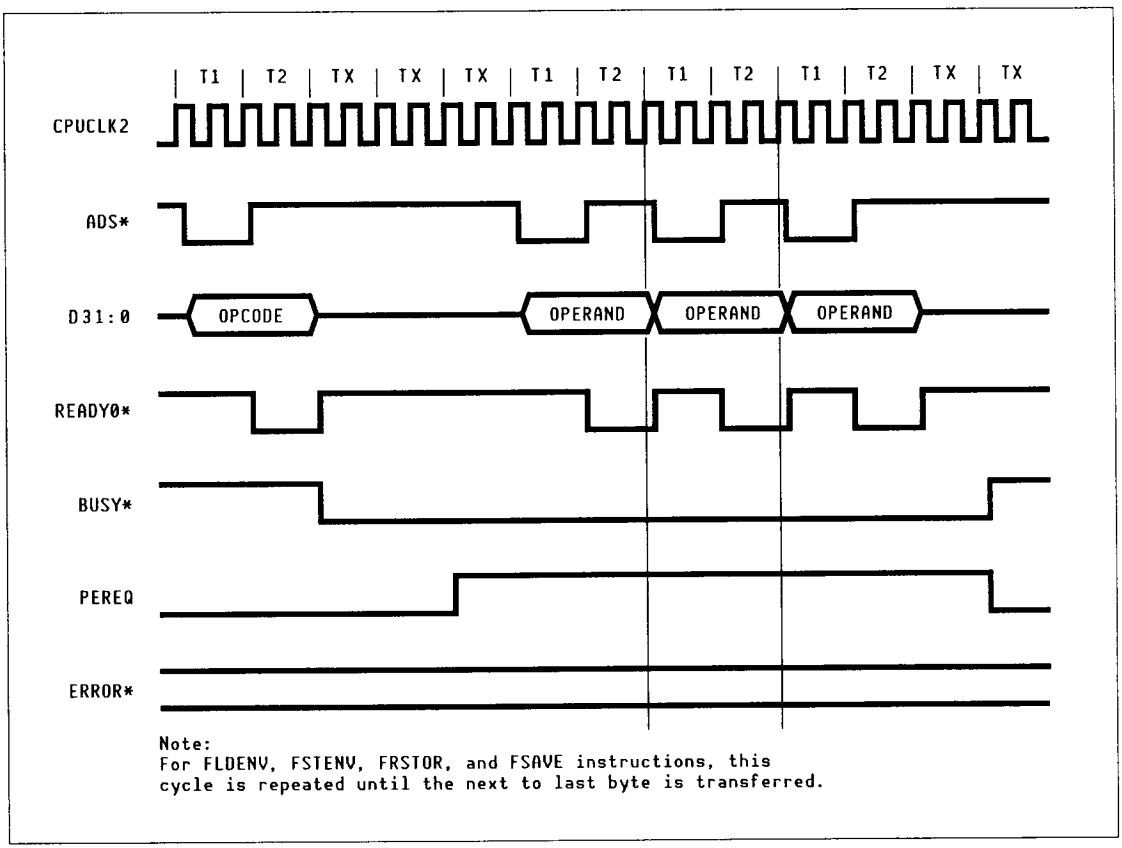
**Figure 9.** *Timing for Transfer of Instructions That Use PEREQ to Synchronize (64 Bits)*




Example Instructions:

FST reg,mem (64-bit real)
FST reg,mem (64-bit integer)

***Figure 10.*** *Timing for Transfer of Instructions That Use PEREQ to Synchronize (80 Bits or More)*



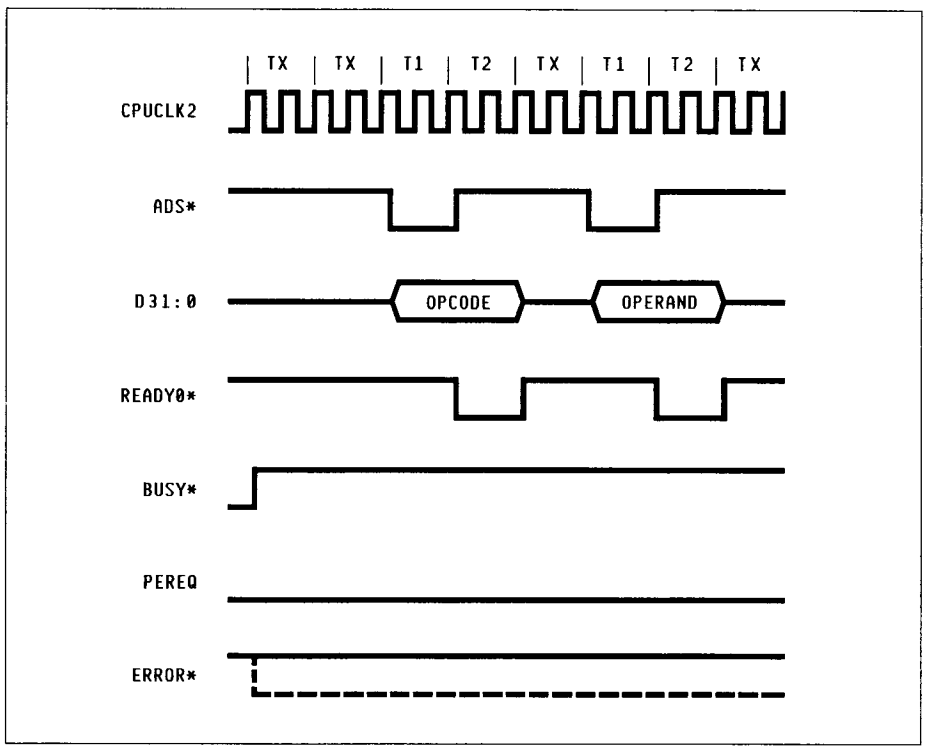

Example Instructions:

| | | | |
|---|---|---|---|
| FLD | reg,mem (80-bit real) | FST | reg,mem (80-bit real) |
| FLD | reg,mem (80-bit BCD) | FST | reg,mem (80-bit BCD) |
| FLDENV | | FRSTOR | |
| FSTENV | | FSAVE | |

Two instructions, FSTCW and FSTSW, perform a return of instantly available data to the CPU from the SuperMathDX coprocessor. Since there is no calculation required on the part of the SuperMathDX coprocessor, the CPU only has to follow the ERROR* checking protocol in waiting for BUSY* to go inactive (see "Error Checking Protocol"). The CPU can then write the instruction opcode and immediately read the requested data without waiting for PEREQ. These instructions take two bus cycles to complete.

Figure 11 shows an example of the timing for this type of operation.

***Figure 11.*** *Timing for FSTCW, FSTSW Instructions*




Example Instructions:

FSTCW

FSTSW
FSTSWAX

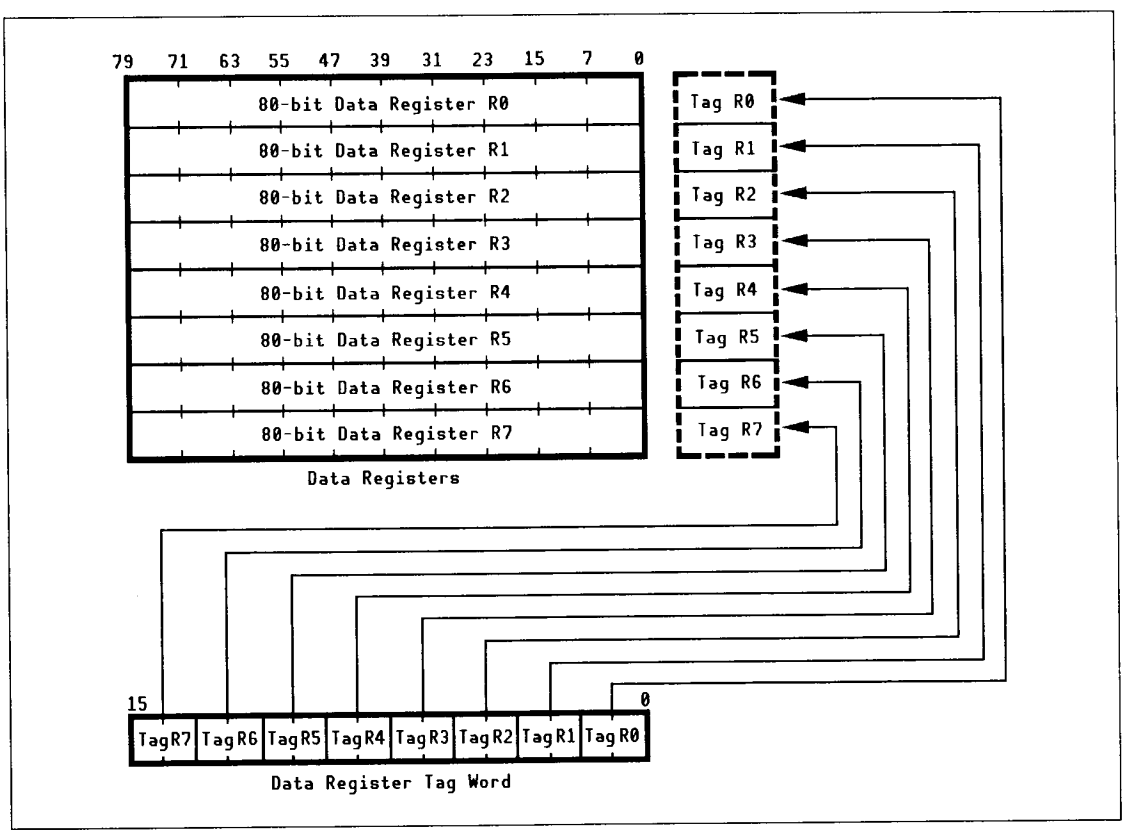
# Register Set

The SuperMathDX coprocessor provides eight 80-bit data registers, which are accessed in a stack-like fashion. In addition, there is a Data Register Tag Word, a Status Register, and a SuperMathDX Control Register. The SuperMathDX coprocessor is register compatible with the Intel® 80387 DX coprocessor.

## Data Registers

The eight 80-bit data registers serve for all SuperMath computations (Figure 12). These registers are either accessed in stack-like fashion or are individually addressed.

***Figure 12.*** *Data Registers*



As a stack, the SuperMathDX FPU instruction set can perform operations on the value in the register designated as the Top of Stack and either a value located in system memory or a value residing in one of SuperMathDX coprocessor's other seven data registers. The register designated as the Top of Stack is selected by the TS2:0 bit field in the Status Register. The remaining data registers are then addressed in unit incremental steps (with a rollover occurring from Data Register R7 to Data Register R0) relative to the chosen Top of Stack register (see Figures 13 and 14 ).

After a RESET sequence, the values in the data registers are undefined. Execution of a FINIT (initialize) instruction does not change the values in the data registers. However, each data register tag field defaults to a binary value of 11 (Data Register Empty).

## Data Register Tag Word

Each data register has a corresponding tag field (Figures 13 and 14). The eight tag fields are contained in the Data Register Tag Word. The tag field for each data register indicates the validity of the contents in the data register, in accordance with Table 7.

***Table 7.*** *Tag Field Validity*

| B1 | B0 | Data Register Contents |
|---|---|---|
| 0 | 0 | Register contents valid. |
| 0 | 1 | Register contents value equals zero. |
| 1 | 0 | Register contents are either QNaN, SNaN, Infinity, Denormal, or in an unsupported format. |
| 1 | 1 | Register empty (default after RESET/FINIT). |

Note that the tag fields are not Top of Stack relative. Each tag field corresponds to a specific data register, regardless of the Top of Stack designated by the TS2:0 bit field in the Status Register. (See Figure 14.) Firmware should read the value in the TS2:0 bit field to determine which tag field in the Data Register Tag Word corresponds to the Top of Stack.

The Data Register Tag Word defaults to a value of FFh at the completion of either a RESET sequence or FINIT instruction.

***Figure 13.*** *Data Register Tag Words, Data Register R0*

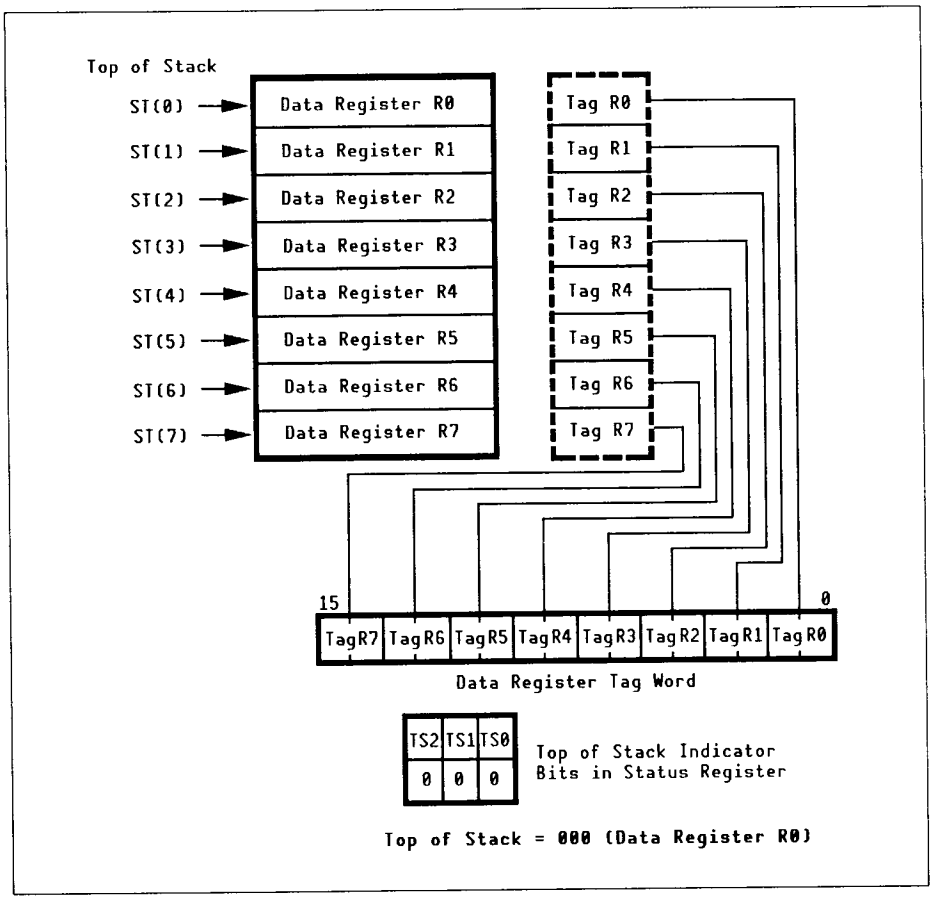
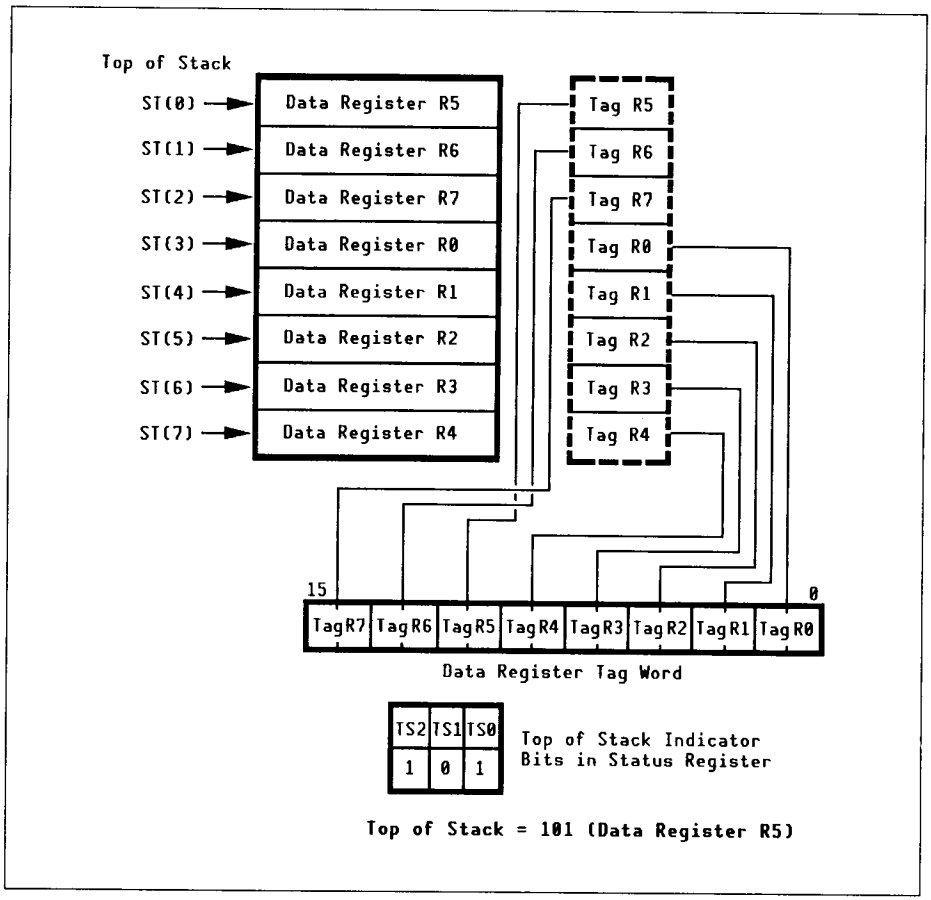


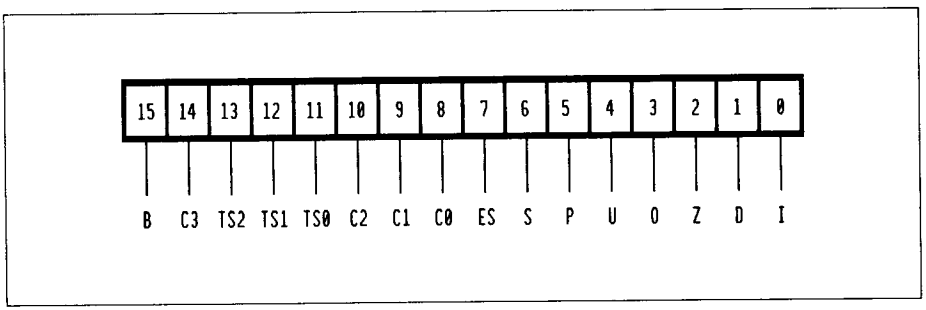
*Figure 14.* *Data Register Tag Words, Data Register R5*



## Status Register

The 16-bit Status Register contains information about the results of coprocessor operations (Figure 15). It contains dedicated bit fields which represent the exception status, condition codes, stack fault, and Top of Stack pointer. The state of the condition codes, exception status bits, and stack fault bit after execution of an instruction is provided in the description of each instruction.

***Figure 15.*** *16-Bit Status Register*




The information in the Status Register can be transferred to the host system for its own processing.

*bits:* 15 B — Busy Bit. This bit exists to maintain backward compatability with the 8087. It has the same value as the contents of the ES Bit (bit 7). Bit B defaults to a logic 0 after a FINIT instruction. After a RESET sequence, this bit defaults to a logic 1 to indicate to the host system that the SuperMath coprocessor is present.

14 C3 — Condition Code Bit 3. This bit, in conjunction with condition code bits C2-C0 (bits 10-8), reflect the condition of the result of an instruction execution. Bits C3-C0 are updated by an instruction at the completion of its execution. Bit C3 defaults to a logic 0 after a RESET sequence or FINIT instruction.

13:11 TS2 Top of Stack Bit Field. The value in these bits designates the Top of Stack Data Register. These bits default to logic 0 values after a RESET sequence or FINIT instruction.

| TS2 | TS1 | TS0 | Description |
|---|---|---|---|
| 0 | 0 | 0 | Top of Stack = Data Register 0 (default after RESET/FINIT) |
| 0 | 0 | 1 | Top of Stack = Data Register 1 |
| 0 | 1 | 0 | Top of Stack = Data Register 2 |
| 0 | 1 | 1 | Top of Stack = Data Register 3 |
| 1 | 0 | 0 | Top of Stack = Data Register 4 |
| 1 | 0 | 1 | Top of Stack = Data Register 5 |
| 1 | 1 | 0 | Top of Stack = Data Register 6 |
| 1 | 1 | 1 | Top of Stack = Data Register 7 |

10:8 C2 Condition Code Bits 2-0. These bits, in conjunction with Condition Code Bit 3 (bit 14), reflect the condition of the result of an instruction's execution. These bits are updated by an instruction at the completion of its execution. They default to logic 0 values after a RESET sequence or a FINIT instruction.

7 ES Exception Status Bit. This bit is set if any unmasked Exception Flags (bits 5-0) are set. When this bit is set, the ERROR* output goes to an active (logic low) level. Bit ES defaults to a logic 0 after a FINIT instruction. After a RESET sequence, this bit defaults to a logic 1 to indicate to the host system that the SuperMathDX coprocessor is present.

6 S Stack Fault Bit. This bit is set if an invalid operation occurs due to a stack underflow or overflow. When this bit is set, condition code bit 1 (C1) shows whether the fault was an underflow (C1 = 0) or an overflow (C1 = 1). Bit S defaults to a logic 0 after a RESET sequence or FINIT instruction.

5 P Precision Exception Flag. This bit is set if a precision exception is realized as a result of an instruction's execution. A detailed explanation is provided in the "Exceptions" section of this document. Bit P defaults to a logic 0 after a RESET sequence or FINIT instruction.

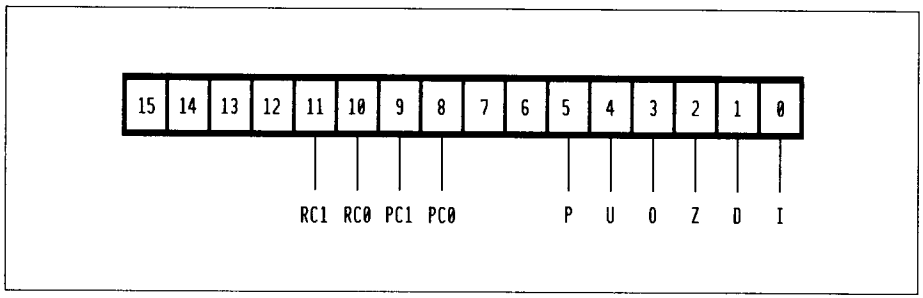
4 U Data Underflow Exception Flag. Bit U is set if a data underflow exception is realized as a result of instruction's execution. A detailed explanation is provided in the "Exceptions" section of this document. This bit defaults to a logic 0 after a RESET sequence or FINIT instruction.

| | | |
|---|---|---|
| 3 | O | Data Overflow Exception Flag. Bit O is set if a data overflow exception is realized as a result of an instruction's execution. A detailed explanation is provided in the "Exceptions" section of this document. This bit defaults to a logic 0 after a RESET sequence or FINIT instruction. |
| 2 | Z | Divide By Zero Exception Flag. Bit Z is set if an attempt to divide by zero is realized as a result of an instruction's execution. A detailed explanation is provided in the "Exceptions" section of this document. Note that this bit may be set by instructions other than FDIV-type instructions. Bit Z defaults to a logic 0 after a RESET sequence or FINIT instruction. |
| 1 | D | Denormalized Operand Exception Flag. Bit D is set if at least one of the operands on which an instruction is to be performed is a denormalized number. A detailed explanation is provided in the "Exceptions" section of this document. This bit defaults to a logic 0 after or FINIT instruction. |
| 0 | I | Invalid Operation Exception Flag. Bit I is set if an invalid operation is attempted. A detailed explanation is provided in the "Exceptions" section of this document. This bit defaults to a logic 0 after a FINIT instruction; after a RESET sequence, it defaults to a logic 1. |

## Control Register

The 16-bit Control Register (Figure 16) specifies rounding control, precision control, and the exception masks. The Rounding Control bits determine the method of rounding the results of an operation. The Precision Control bits specify the realized precision of an operation's result. The Exception Mask bits are programmable masks for each of the six exception conditions.

***Figure 16.*** *16-Bit Control Register*




*bits:* 15:13 Reserved. Firmware should ignore the state of these bits during reads of the contents of this register (reading the values transferred by a FSTSW instruction). All values written to these bits (with an FLDCW instruction) are ignored by the SuperMathDX coprocessor. These bits are undefined after a RESET sequence or FINIT instruction.

12 Reserved. At the completion of a RESET sequence and after execution of a FINIT instruction, the state of this bit is 0. Firmware should ignore the state of these bits during reads of the contents of this register (reading the values transferred by a FSTSW instruction). All values written to these bits (with an FLDCW instruction) are ignored by the SuperMathDX coprocessor.

11:10 RC1:RC0 Rounding Control Bits 1-0. Per the following table, these bits specify the rounding method to be applied to any result that requires rounding after the instruction's completion:

| RC1 | RC0 | Description |
|---|---|---|
| 0 | 0 | Round to nearest even (default) |
| 0 | 1 | Round toward -∞ |
| 1 | 0 | Round toward +∞ |
| 1 | 1 | Truncate towards zero |

These bits default to logic 0 values after a RESET sequence or FINIT instruction.

9:8 PC1:PC0 Precision Control Bits 1-0. Per the table, these bits specify the significand's precision after the completion of FADD, FSUB, FMUL, FDIV, and FSQRT instructions.

| PC1 | PC0 | Description |
|---|---|---|
| 0 | 0 | Single Precision/24-bit |
| 0 | 1 | Reserved |
| 1 | 0 | Double Precision/53-bit |
| 1 | 1 | Double Extended Precision/64-bit (default) |

These bits default to logic 1 values after a RESET sequence or FINIT instruction.

7:6 Reserved. Firmware should ignore the state of these bits during reads of the contents of this register (reading the values transferred by a FSTSW instruction). All values written to these bits (with an FLDCW instruction) are ignored by the SuperMathDX coprocessor. These bits are undefined after a RESET sequence or FINIT instruction.

5 P Precision Exception Mask. Bit P masks a Precision exception should one be realized. This bit defaults to a logic 1 after a RESET sequence or FINIT instruction.

4 U Data Underflow Exception Mask. Bit U masks a Data Underflow exception should one be realized. This bit defaults to a logic 1 after a RESET sequence or FINIT instruction.

3 O Data Overflow Exception Mask. Bit O masks a Data Overflow exception should one be realized. This bit defaults to a logic 1 after a RESET sequence or FINIT instruction.

2 Z Divide By Zero Exception Mask. Bit Z masks a Divide by Zero exception should one be realized. This bit defaults to a logic 1 after a RESET sequence or FINIT instruction.

| | | |
|---|---|---|
| 1 | D | Denormalized Operand Exception Mask. Bit D masks a Denormalized Operand exception should one be realized. This bit defaults to a logic 1 after a RESET sequence or FINIT instruction. |
| 0 | I | Invalid Operation Exception Mask. Bit I masks an Invalid Operation exception should one be realized. This bit defaults to a logic 1 after a FINIT instruction. After a RESET sequence, this bit defaults to a logic 0. |

## Data Register Errors

The SuperMathDX coprocessor detects the following Data Register errors:

- Source Register Empty
- Destination Register Full.

Source Register Empty (stack underflow) errors occur when the SuperMathDX coprocessor attempts to perform an operation with a source operand from a data register that is empty (indicated by a binary value of 11 in the data register's tag field). Similary, Destination Register Full (stack overflow) errors occur when the SuperMathDX coprocessor attempts to perform an operation to a destination data register that is not empty (indicated by a binary value of 00, 01, or 10 in the data register's tag field).

In both instances, the SuperMathDX coprocessor reports the stack errors in the S bit field of the Status Register. When the S bit is set, the value of condition code C1 indicates whether the stack has overflowed (C1 = 1) or underflowed (C1 = 0). Both stack overflows and underflows cause Invalid Operation exceptions.

# Exceptions

While performing operations, the SuperMathDX coprocessor constantly monitors data characteristics and reports all inconsistencies and errors. Errors or inconsistencies are classified as one of the following exception conditions:

- Precision
- Data Underflow
- Data Overflow
- Denormal Operand
- Divide by Zero
- Invalid Operation.

The Status Register contains a flag bit for each of the exception conditions.

When an unmasked exception is realized (corresponding exception mask bit is 0), the SuperMathDX coprocessor begins an error trap sequence, activating the ERROR* output to the system. If the unmasked exception is a Precision exception, Data Underflow exception, or Data Overflow exception, the exception is processed in accordance with the IEEE-754-1985 specification, and the result is stored in the destination before the error trap sequence is begun. System hardware generally invokes a firmware routine to process the exception.

When a masked exception is realized (corresponding exception mask bit is set), the SuperMathDX coprocessor proceeds in a prioritized predetermined manner, which is presented in Table 8. The exception handling sequence shown in Table 8 conforms to specification IEEE-754-1985. (Note that a priority of 1 has the highest ranking.)

*Table 8. SuperMathDX Coprocessor Masked Exception Handling*

| Priority | Exception | Cause | SuperMathDX FPU Action With Exception Masked |
|---|---|---|---|
| 1 | Invalid Operation | Operation attempted on a SNaN, unsupported format, indeterminate form, stack underflow, or stack overflow. (Stack underflows and overflows also set the S bit field.) | Result is either a QNaN, indefinite integer, or indefinite BCD. |
| 2 | Denormalized | At least one of the instruction's operands is denormalized. | Normal processing occurs. |
| 3 | Divide by Zero | At some time during the instruction's execution with a non-infinite, nonzero dividend, the corresponding divisor was 0. | Result is ∞. |
| 4 | Data Overflow | The calculated result exceeds the largest value permitted by the specified format. | Result is either the largest permitted finite value or ∞. |
| 5 | Data Underflow | The calculated result is nonzero, but is smaller than the smallest value permitted by the specified format. | Result is either zero or denormalized. |
| 6 | Precision | The calculated result is not discretely representable in the specified format; hence, it has been rounded according to the rounding mode specified by the RC1-0 bits. | Normal processing occurs. |

## Precision Exceptions

A precision exception occurs when the result of an operation cannot be exactly (discretely) represented in the characteristic format of the destination. As an example, transcendental operations, such as FCOS, generally produce irrational results. When real repeating results, irrational results, or results more accurate than the specified destination format occur, the result is rounded according to the method specified by Rounding Control bits 1-0 of the Status Register. The rounded results are then presented to their destination.

## Data Underflow Exception

A Data Underflow exception occurs when any nonzero result of an operation is too small to be represented by the destination's format.

With the Data Underflow exception masked, the Data Underflow exception flag is set only if the resulting 0 or denormalized number loses its precision. When this occurs, the Precision exception flag is also set.

When the Data Underflow exception is unmasked, a Data Underflow exception occurs for each resultant data underflow. The SuperMathDX coprocessor then processes the underflow in the exact fashion mandated by the IEEE-754-1985 specification.

If the instruction's destination is located in system memory, the processing halts and no result is provided. If the destination is a data register and the instruction was not an FSCALE calculation resulting in an extreme data underflow, the result is multiplied by 24576, rounded according to the manner specified by Rounding Control bits 1-0 of the Status Register, and stored in the destination register. If the destination is a data register and the instruction was an FSCALE calculation resulting in an extreme data underflow, a +0/-0 is produced, just as if the Data Underflow exception were masked.

## Data Overflow Exception

A Data Overflow exception occurs when the result of an operation is too large to be represented in the specified destination. When the Data Overflow exception is masked, the exception flag is set and the result is rounded in the manner specified by the Rounding Control bits.

When the Data Overflow exception is unmasked, a Data Overflow exception occurs for each resultant data overflow. The SuperMathDX coprocessor then processes the overflow in the exact fashion mandated by the IEEE-754-1985 specification.

If the instruction's destination is located in system memory, the processing halts and no result is provided. If the destination is a data register and the instruction was not an FSCALE calculation resulting in an extreme data overflow, the result is divided by 24576, rounded according to the manner specified by Rounding Control bits 1-0 of the Status Register, and stored in the destination register. If the destination is a Data Register and the instruction was an FSCALE calculation resulting in an extreme data overflow, a $+\infty/-\infty$ is produced, just as if the Data Underflow exception were masked.

## Denormal Operand Exception

A Denormal Operand exception occurs if at least one of an instruction's operands is a denormalized number. If the Denormal Operation exception is masked, the operation completes in the fashion specified by the IEEE-754-1985 specification using the denormal number and sets the Denormal Operand exception flag. If the Denormal Operation exception is unmasked, the Denormal Operation exception flag is set and an error trap sequence begins with no result presented to the destination.

## Divide by Zero Exception

A Divide by Zero exception occurs when an instruction attempts to divide a finite nonzero operand by zero. The FDIV, FYL2X and FXTRACT instructions can all cause this exception. If the Divide by Zero exception is masked, the FDIV and FYL2X instructions return a signed infinity (with the sign resulting in the Exclusive OR of the operands) to the destination. The FXTRACT instruction places a signed zero (with the same sign as the original operand) in the Top of Stack ST(0) and a negative infinity in ST(1). If the Divide by Zero exception is unmasked, the Divide by Zero exception flag is set and an error trap sequence begins with no result presented to the destination.

## Invalid Operation

Invalid Operation exceptions occur when operations that produce meaningless results are attempted. The IEEE-754-1985 document specifies the manner of detecting and reporting instances when instructions are performed on invalid operands. Invalid Operation exceptions result from both data register errors (stack underflows and overflows) and invalid operands specified in arithmetic operations.

When the Invalid Operation exception is masked, the SuperMathDX coprocessor operates as shown in Table 9, in accordance with specification IEEE-754-1985. If the Invalid Operation exception is unmasked, the Invalid Operation exception flag is set, the instruction's execution is halted, and an error trap sequence begins. No result is presented to the destination.

**Table 9.** *Responses to Invalid Operations*

| Operation | Operands | Masked Response |
|---|---|---|
| Arithmetic | Unsupported operands | QNaN indefinite |
| Arithmetic | Signaling NaN | QNaN |
| Compare and test | One or both operands is a NaN | Set condition codes to unordered. |
| Addition | Opposite-signed infinities | QNaN indefinite |
| Subtraction | Like-signed infinities | QNaN indefinite |
| Multiplication | 0 and ∞ operands | QNaN indefinite |
| Division | ∞, ∞ operands or 0, 0 operands | QNaN indefinite |
| Remainder: FPREM, FPREM1 | Divisor = 0 or dividend = ∞ | QNaN indefinite |
| Trigonometric: FCOS, FPTAN, FSIN, FSINCOS | Argument = ∞ | QNaN indefinite |
| Squart root, log: FSQRT, FYL2X | Negative operand | QNaN indefinite |
| Integer, BCD store: FIST(P) | Empty register, NaN, ∞ | Store integer, BCD indefinite |
| Integer, BCD store: FBSTP | Empty register, NaN, ∞; or exceeds integer range | Store packed decimal indefinite |
| Register exchange: FXCH | Empty register | Set empty register to QNaN indefinite, then exchange. |

# Instruction Set

This section describes the complete set of floating-point instructions available on the SuperMathDX floating-point processor. Note that the SuperMathDX FPU instruction set is identical to that of the 80387.

## Notation

The following sections describe how each instruction is presented. Tables in the sections define symbols and their meanings as used in the descriptions of the instructions.

Each instruction includes the following information:

- Instruction name
- Instruction syntax
- Instruction format
- Instruction description
- Condition codes
- Zero/infinity
- Exception flags.

### Instruction Name

The name of an instruction (or related sets of instructions) is listed at the top of a page and is followed by a brief description of the instruction.

Table 10 defines the instruction mnemonics. Within the table, an asterisk (*) represents any number of missing characters.

*Table 10. Instruction Mnemonics*

| Mnemonic | Meaning |
|---|---|
| F* | All SuperMath FPU instructions begin with F. |
| FI* | Instructions that operate on integer data types |
| FB* | Instructions that operate with BCD data types |
| F*P | Instructions that cause the stack to be popped once |
| F*PP | Instructions that cause the stack to be popped twice |
| F*R | The reverse form of the instruction |

## Syntax and Format

The Syntax shows the usage of an instruction. The Format section lists all operand sources and destinations. All arguments that must be supplied are shown in italics. Optional arguments are surrounded by square brackets. Table 11 defines the argument symbols used in the Syntax and Format sections.

*Table 11. Argument Symbols*

| Symbol | Meaning |
|---|---|
| *dest* | Destination operand |
| *source* | Source operand |
| *stack* | Top of stack register represented as ST(0) |
| *register* | Data register represented as ST(*n*) where *n* = 0 to 7 |
| *memory* | Memory operand address in the 80386 |

## Description

The Description begins with a table showing the instruction, source operand, opcode, and clock cycles. The clock cycles are specified in terms of CPU clock cycles. The opcode is specified for each format of the instruction. Table 12 lists the types of opcodes and their location in memory.

**Table 12.** *Opcode Locations*

| Type | Location |
|---|---|
| ST(0) | Top of Stack |
| ST(1) | Next to Top of Stack |
| ST(n) | 80-bit data register, where n = 2-7 (referenced from the Top of Stack) |
| mem16i | Pointer to 16-bit integer in system memory |
| mem32i | Pointer to 32-bit integer in system memory |
| mem64i | Pointer to 64-bit integer in system memory |
| mem80b | Pointer to 80-bit BCD in system memory |
| mem32r | Pointer to 32-bit real number in system memory |
| mem64r | Pointer to 64-bit real number in system memory |
| mem80r | Pointer to 80-bit real number in system memory |
| mem2byte | Pointer to a 2-byte field in system memory |
| mem14/28byte | Pointer to either a 14-byte or 28-byte field in system memory |
| mem94/108byte | Pointer to either a 94-byte or 108-byte field in system memory |
| 80386 AX Reg | The AX register in an 80386 compatible CPU |
| const | The specified 80-bit real number constant in SuperMath |

The opcode includes the field abbreviations described in Table 13.

**Table 13.** *Field Abbreviations*

| Abbreviation | Meaning |
|---|---|
| xxh | Hexadecimal value |
| SIB | Stack Index Base (from the 80386-compatible CPU) |
| displ | Displacement: an 8-bit or 32-bit value following the instruction (from the 80386-compatible CPU) |
| REG2:0 | SuperMathDX FPU data register; 3 bits |
| RM2:0 | Register/Memory sits on the 80386-compatible CPU; 3 bits |
| MD1:0 | Mod bit field on the 80386-compatible CPU; 2 bits |

## Condition Codes

The Condition Codes section shows how executing the instruction affects the status of the SuperMathDX coprocessor condition codes (C0, C1, C2, and C3).

## Zero/Infinity

This section includes a table that provides information about the results produced when an instruction is executed with large operands. This section is omitted if it does not apply to an instruction. Table 14 describes the symbols used in the zero/infinity tables.

***Table 14.*** *Zero /Infinity Symbols*

| Symbol | Meaning |
|---|---|
| x,y | Positive integers |
| NaN | Not a number |
| R(0) | Zero as produced by current RC mode |

## Exception Flags

Table 15 lists the classes of numeric exception conditions that are recognized by the SuperMathDX coprocessor while executing numeric instructions.

*Table 15. Exception Flag Symbols*

| Symbol | | Meaning |
|---|---|---|
| QNaN | | Quiet NaN; has 1 as the most significant bit of the significand. |
| Exception status: | S | Stack fault or invalid register operation |
| | P | Precision error |
| | U | Underflow error |
| | O | Overflow error |
| | Z | Divide by zero error |
| | D | Denormalized operand error |
| | I | Invalid operation |
| | | Blank means unaffected due to execution of instruction. |
| MEM | | The value is loaded from memory. |

A table within the Exception Flags section describes the flags that are set for the exceptions that can occur for the current instruction. Otherwise, the section indicates "no exceptions."

## F2XM1 — Compute (2x–1)

Syntax: **F2XM1**

Format: **F2XM1**

Description:

| Instruction | Operand(s) (dest, source) | 1st Byte | Opcode/Instruction Encoding | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| F2XM1 | ST(0) | D9h | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | | 8-141 |

F2XM1 replaces the value at the top of the stack with $2^x-1$, where x is the original value in the Top of Stack (where $-1 \leq x \leq 1$). The result is rounded, depending upon the mode in effect.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 except after a Precision exception, where it defines whether or not rounding is away from 0.

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | +0 | $-\infty$ | −1 |
| −0 | −0 | $+\infty$ | $+\infty$ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denorm,0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Operand used | | | | | | 1 | |
| | Unmasked | Trap, unchanged | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Trap, unchanged | | | | | | | 1 |

## FABS— Absolute value

Syntax: **FABS**

Format: **FABS**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FABS | ST(0) | D9h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | 4 |

The FABS instruction replaces the value at the top of the stack with its absolute value. FABS sets the sign of the Top of Stack to 0 (i.e., positive).

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0.

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | +0 | $-\infty$ | $+\infty$ |
| -0 | +0 | $+\infty$ | $+\infty$ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register | Masked | QNaN | 1 | | | | | | 1 |
| Error | Unmasked | Trap/Abort | 1 | | | | | | 1 |

## FADD — Floating-point add

Syntax: **FADD(P)** [[*dest*,]*source*]
**FIADD** [[*dest*,]*source*]

Format: **FADD** *stack, memory*
**FADD** *stack, register*
**FADD(P)** *register, stack*
**FIADD** *stack, memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FIADD | ST(0),mem16i | DEh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 12 |
| FIADD | ST(0),mem32i | DAh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 12 |
| FADD | ST(0),mem32r | D8h | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 10 |
| FADD | ST(0),mem64r | DCh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 10 |
| FADD | ST(0),ST(n) | D8h | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | 7 |
| FADD | ST(n),ST(0) | DCh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | 7 |
| FADDP | ST(n),ST0 | DEh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | 7 |

The FADD instruction fetches, then adds the operands. The result is stored in the destination. The source operand is changed to extended precision format, if required. Depending on the mode in effect, the result is rounded to the precision indicated by the mode bits. FADDP causes the stack to be popped once.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 except after a Precision exception, where it defines whether or not rounding is away from 0.

Zero/Infinity:

| Operand 1 | Operand 2 | Result | Operand 1 | Operand 2 | Result |
|---|---|---|---|---|---|
| +0 | +0 | +0 | +∞ | +∞ | +∞ |
| -0 | -0 | -0 | -∞ | -∞ | -∞ |
| +0 | -0 | R(0) | +∞ | -∞ | Invalid |
| -0 | +0 | R(0) | -∞ | +∞ | Invalid |
| -x | +x | R(0) | +∞ | x | +∞ |
| +x | -x | R(0) | -∞ | x | -∞ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Overflow | Masked | Infinity | | | | 1 | | | |
| | Unmasked | Round, scale | | | | 1 | | | |
| Denormal | Masked | Denorm | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FCHS — Change sign

Syntax: **FCHS**

Format: **FCHS**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FCHS | ST(0) | D9h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | 4 |

The FCHS instruction replaces the value at the Top of Stack with its opposite.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0.

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | -0 | $+\infty$ | $-\infty$ |
| -0 | +0 | $-\infty$ | $+\infty$ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register | Masked | QNaN | 1 | | | | | | 1 |
| Error | Unmasked | Trap, Abort | 1 | | | | | | 1 |

## FCLEX— Clear exceptions

Syntax: **FCLEX**

Format: **FCLEX**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FCLEX | ST(n),ST(0) | DBh | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | | 4 |

The FCLEX instruction resets all exception flags, the Exception Status flag, and the Busy flag to 0.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags:

| Exception | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|
| All | Reset to zero | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## FCOM — Floating-point compare

Syntax: **FCOM(P) (P)** [[*dest*,]*source*]

Format:
**FICOM(P)** *stack, memory*
**FCOM(P)** *stack, memory*
**FCOM** *stack, register*
**FCOMP** *stack, register*
**FCOM (P)(P)** *stack, register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FICOM | ST(0),mem16i | DEh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 13 |
| FICOMP | ST(0),mem16i | DEh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 13 |
| FICOM | ST(0),mem32i | DAh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 10 |
| FICOMP | ST(0),mem32i | DAh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 13 |
| FCOM | ST(0),mem32r | D8h | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 10 |
| FCOMP | ST(0),mem32r | D8h | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 10 |
| FCOM | ST(0),mem64r | DCh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 12 |
| FCOMP | ST(0),mem64r | DCh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 12 |
| FCOM | ST(n),ST(0) | D8h | 1 | 1 | 0 | 1 | 0 | REG2 | REG1 | REG0 | | 8 |
| FCOMP | ST(n),ST(0) | D8h | 1 | 1 | 0 | 1 | 1 | REG2 | REG1 | REG0 | | 5 |
| FCOMPP | ST(1),ST(0) | DEh | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | 8 |

The Compare instructions compare the top of the stack to the source operand. The source operand can be a register, a single or double real, or a word or long integer, and it is converted to extended precision, if necessary. Exception flags are used to show the result of the comparison.

If either operand is a NaN or is an unsupported number, or if a stack fault occurs, the Invalid Operation exception flag is set and the result is set to Unordered. If the operands are QNaNs, the result is an invalid operation.

Condition Codes: The result of the comparison is determined by the condition code bits C3, C2, C1, and C0, as follows:

| dest/source Status | C3 | C2 | C1 | C0 |
|---|---|---|---|---|
| dest > source | 0 | 0 | 0 | 0 |
| dest < source | 0 | 0 | 0 | 1 |
| dest = source | 1 | 0 | 0 | 0 |
| Unordered | 1 | 1 | 0 | 1 |

If the source operand is either NaN or an undefined value, or if a stack fault occurs, an Invalid Operation exception occurs and the condition bits are undefined.

If the source register is empty, C3 = C2 = C0 = 1; C1 = 0.

Zero/Infinity:

| Destination | Source | Result | Destination | Source | Result |
|---|---|---|---|---|---|
| +0 | +0 | equal | +∞ | +∞ | equal |
| -0 | -0 | equal | -∞ | -∞ | equal |
| +0 | -0 | equal | +∞ | -∞ | dest>source |
| -0 | +0 | equal | -∞ | +∞ | dest<source |
| +0 | +x | dest<source | +∞ | x | dest>source |
| -0 | +x | dest<source | -∞ | x | dest<source |
| +0 | -x | dest>source | x | -∞ | dest>source |
| -0 | -x | dest>source | x | +∞ | dest<source |
| NaN | x | Unordered | x | NaN | Unordered |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | Unordered | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |
| Denormal | Masked | QNaN | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | Unordered | | | | | | | 1 |
| | Unmasked | Trap, abort | | | | | | | 1 |

## FCOS — Compute cosine(x)

Syntax: **FCOS**

Format: **FCOS**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FCOS | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 5-129 |

The FCOS instruction performs the function evaluation: $y = \cos(x)$. The source operand, x, is taken from the Top of Stack and must be in the range $0 < |x| < 2^{63}$. The result, y, is rounded according to the mode in effect and returned to the Top of Stack.

Condition Codes: C0 and C3 are undefined. C2 is used to specify reduction; it is set to 1 and is incomplete if the operand at the top of the stack is out of range. C1 is set to 1 after a Precision exception if the rounding done by the instruction was upward.

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | +1 | +∞ | Invalid operation |
| -0 | +1 | -∞ | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Denormal | Masked | Operand used | | | | | | 1 | |
| | Unmasked | Trap, unchanged | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FDECSTP— Decrement stack pointer

Syntax: **FDECSTP**

Format: **FDECSTP**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FDECSTP | None | D9h | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | 4 |

The FDECSTP instruction subtracts one (modulo 8) from the data register Top of Stack pointer (IS2:0 of the Status Register).

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0.

Exception Flags: No exceptions.

## FDIV — Floating-point divide

Syntax: **FDIV(R)(P)** [[*dest*,]*source*]

Format:
**FIDIV(R)** *stack, memory*
**FDIV(R)** *stack, memory*
**FDIV(R)** *stack, register*
**FDIV(R)(P)** *register, stack*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FIDIV | ST(0),mem16i | DEh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 29-51 |
| FIDIVR | ST(0),mem16i | DEh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 29-51 |
| FIDIV | ST(0),mem32i | DAh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 29-51 |
| FIDIVR | ST(0),mem32i | DAh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 29-51 |
| FDIV | ST(0),mem32r | D8h | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 28-48 |
| FDIVR | ST(0),mem32r | D8h | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 28-48 |
| FDIV | ST(0),mem64r | DCh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 29-50 |
| FDIVR | ST(0),mem64r | DCh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 29-50 |
| FDIV | ST(0),ST(n) | D8h | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | 24-45 |
| FDIVR | ST(0),ST(0) | D8h | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | 24-45 |
| FDIV | ST(n),ST(0) | DCh | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | 24-45 |
| FDIVR | ST(n),ST(0) | DCh | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | 24-45 |
| FDIVP | ST(n),ST(0) | DEh | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | 24-45 |
| FDIVRP | ST(n),ST(0) | DEh | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | 24-45 |

The FDIV instructions divide the Top of Stack by the other operand. The quotient is rounded to the precision according to the mode in effect and then placed in the destination.

Divide instructions ending in R (reverse) divide the Top of Stack into the other operand. These instructions each contain one additional clock cycle.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 except after a Precision exception, where it defines whether or not rounding is away from 0.

Zero/Infinity:

| Operand 1 | Operand 2 | Result | Operand 1 | Operand 2 | Result |
|---|---|---|---|---|---|
| +0 | +0 | Invalid | +∞ | +∞ | Invalid |
| +0 | -0 | Invalid | +∞ | +∞ | Invalid |
| -0 | +0 | Invalid | -∞ | +∞ | Invalid |
| -0 | -0 | Invalid | -∞ | -∞ | Invalid |
| +0 | +x | +0 | +∞ | +x | +∞ |
| -0 | -x | +0 | +∞ | -x | -∞ |
| +0 | -x | -0 | -∞ | +x | -∞ |
| -0 | +x | -0 | -∞ | -x | +∞ |
| +x | +y | +0* | +∞ | +0 | +∞ |
| -x | -y | +0* | +∞ | -0 | -∞ |
| +x | -y | -0* | -∞ | +0 | -∞ |
| -x | +y | -0* | -∞ | -0 | +∞ |

* These cases apply when $\left(\frac{+x}{+y}\right)$ produces an extreme denormalization or underflow (when the Underflow exception is masked).

Note: $\frac{Dividend}{Divisor}$ = Result

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Overflow | Masked | Infinity | | | | 1 | | | |
| | Unmasked | Round, scale | | | | 1 | | | |
| Divide by Zero | Masked | Infinity* | | | | | 1 | | |
| | Unmasked | Trap, unchanged | | | | | 1 | | |
| Denormal | Masked | Operand used | | | | | | 1 | |
| | Unmasked | Trap, unchanged | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Trap, unchanged | | | | | | | 1 |

* When the Divide by Zero exception is masked, the results in the above Zero/Infinity table are produced.

## FFREE — Free floating-point register

Syntax: **FFREE** [*dest* ]

Format: **FFREE** *register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FFREE | ST(n) | DDh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | 3 |

The FFREE instruction changes the destination register tag to empty, but it does not alter the contents of the register.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

## FINCSTP— Increment stack pointer

Syntax: **FINCSTP**

Format: **FINCSTP**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FINCSTP | None | D9h | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | 3 |

The FINCSTP instruction adds one (modulo 8) to the data register Top of Stack pointer.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0.

Exception Flags: No exceptions.

## FINIT — Initialize SuperMathDX floating-point unit

Syntax: **FINIT**

Format: **FINIT**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FINIT | None | DBh | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | | 3 |

The FINIT instruction resets the SuperMathDX floating-point coprocessor as shown in Table 16.

***Table 16.*** *FINIT Instruction Reassignments*

| Item | Reset Value |
|---|---|
| Mode control register | 037Fh |
| Status register | 0000h |
| Data registers Tag Word | FFFFh (Data Registers Empty) |
| Rounding control | RC1:0 = 00 (round to nearest) |
| Precision | PC1:0 = 11 (64-bit extended) |
| Top of stack | TS2:0 = 000 |
| Exception flags | Cleared (0) |
| Condition codes (C0-C3) | Cleared (C0 = C1 = C2 = C3 = 0) |

Condition Codes: C0, C1, C2, and C3 are set to 0.

Exception Flags: No exceptions.

## FLD — Load Top of Stack

Syntax: **FLD** [[*dest*,]*source*]

Format: **FLD** *stack, memory*
**FLD** *stack, register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FILD | ST(0),mem16i | DFh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 4 |
| FILD | ST(0),mem32i | DBh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 4 |
| FILD | ST(0),mem64i | DFh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 4 |
| FBLD | ST(0),mem80b | DFh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 63 |
| FLD | ST(0),mem32r | D9h | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 3 |
| FLD | ST(0),mem64r | DDh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 3 |
| FLD | ST(0),mem80r | DBh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 2 |
| FLD | ST(0),ST(n) | D9h | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | 3 |

The FLD instruction fetches the source operand. If the source operand is in single or double precision format, word or long integer format, or packed decimal format, the FLD instruction converts it into extended precision format. The result is stored in the TS2:0 bits field of the Status Word.

The Top of Stack is decremented.

ST7 must be empty or else an invalid operation exception will occur. Precision, Underflow, Overflow, and Divide by Zero exceptions cannot occur as a result of FLD. Denormal and Invalid Operation exceptions cannot occur as a result of 80-bit real or FLD register instructions.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 after normal execution. In the case of a register error, C1 is set to 1 if the destination register is full and to 0 if the source register is empty.

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | +0 | +∞ | +∞ |
| -0 | -0 | -∞ | -∞ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Denormal | Masked | Denorm | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FLD1 — Load constant 1.0

Syntax: **FLD1**

Format: **FLD1**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLD1 | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 2 |

FLD1 pushes 1.0 onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |

## FLDCW — Load mode control word

Syntax: **FLDCW** *source*

Format: **FLDCW** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FLDCW | mem2byte | D9h | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | | 5 |

The FLDCW instruction loads the specified bytes into the Mode Control Word and is typically used to establish the mode of operation.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

## FLDENV — Load SuperMathDX coprocessor environment

Syntax: **FLDENV** *source*

Format: **FLDENV** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FLDENV | mem14/28byte | D9h | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | | 92 |

The FLDENV instruction loads the SuperMathDX FPU environment from the specified memory location. The mode of the 80386-compatible CPU and the operand size determine the format of the environment as shown in Figure 17. The environment is made up of the Mode Control Word, the Status Word, the Data Register Tag Word, the Instruction Pointer, and the Data Pointer.

Condition Codes: C0, C1, C2, and C3 are loaded from memory.

Exception Flags: Exceptions may be loaded with the environment. If the Status Word loaded with the environment contains an exception that is enabled, an error trap will occur after the next wait or after the execution of an Exception Status instruction.

*Figure 17. SuperMathDX Coprocessor Environment*

**16-Bit Protected Mode**

| Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Status Word | | | | | | | | | | | | | | | | 02h |
| Tag Word | | | | | | | | | | | | | | | | 04h |
| Instruction Pointer Offset | | | | | | | | | | | | | | | | 06h |
| Code Segment Selector | | | | | | | | | | | | | | | | 08h |
| Operand Offset | | | | | | | | | | | | | | | | 0Ah |
| Operand Segment Selector | | | | | | | | | | | | | | | | 0Ch |

**16-Bit Real Mode/Virtual 8086 Mode**

| Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Status Word | | | | | | | | | | | | | | | | 02h |
| Tag Word | | | | | | | | | | | | | | | | 04h |
| Instruction Pointer 15:00 | | | | | | | | | | | | | | | | 06h |
| IP 19:16 | | | | 0 | Opcode 10:0 | | | | | | | | | | | 08h |
| Operand Pointer 15:00 | | | | | | | | | | | | | | | | 0Ah |
| OP 19:16 | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0Bh |

**32-Bit Protected Mode**

| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Instruction Pointer Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | 0 | Opcode 10:0 | | | | | | | | | | | Code Segment Selector | | | | | | | | | | | | | | | | 10h |
| Operand Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14h |
| Reserved | | | | | | | | | | | | | | | | Operand Segment Selector | | | | | | | | | | | | | | | | 18h |

**32-Bit Real Mode**

| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Reserved | | | | | | | | | | | | | | | | Instruction Pointer 15:00 | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | Instruction Pointer 31:16 | | | | | | | | | | | | 0 | Opcode 10:00 | | | | | | | | | | | | | | | | 10h |
| Reserved | | | | | | | | | | | | | | | | Operand Pointer 15:00 | | | | | | | | | | | | | | | | 14h |
| 0 | 0 | 0 | 0 | Operand Pointer 31:16 | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 18h |

## FLDL2E — Load constant log2(e)

Syntax: **FLDL2E**

Format: **FLDL2E**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FLDL2E | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | | 2 |

FLDL2E rounds the extended precision constant log2(e) according to the RC mode in effect and pushes it onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FLDL2T — Load constant log2(10)

Syntax: **FLDL2T**

Format: **FLDL2T**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLDL2T | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | | 2 |

FLDL2T rounds the extended precision constant log2(10) according to the RC mode in effect and pushes it onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FLDLG2 — Load constant log10(2)

Syntax: **FLDLG2**

Format: **FLDLG2**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLDLG2 | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | | 2 |

FLDLG2 rounds the extended precision constant $\log_{10}(2)$ according to the RC mode in effect and pushes it onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FLDLN2 — Load constant ln(2)

Syntax: **FLDLN2**

Format: **FLDLN2**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLDLN2 | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | | 2 |

FLDLN2 rounds the extended precision constant ln(2) according to the RC mode in effect and pushes it onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FLDPI — Load constant p

Syntax: **FLDPI**

Format: **FLDPI**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLDPI | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | | 2 |

FLDPI rounds the extended precision constant that approximates Pi ($\pi$) according to the RC mode in effect and pushes it onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FLDZ — Load constant 0

Syntax: **FLDZ**

Format: **FLDZ**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FLDZ | ST(0),const | D9h | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | | 4 |

FLDZ pushes zero onto the stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a register error, indicating that the destination register is full.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register | Masked | QNaN | 1 | | | | | | 1 |
| Error | Unmasked | Trap, abort | 1 | | | | | | 1 |

## FMUL — Floating-point multiply

Syntax: **FMUL(P)** [[*dest*]*source*]
**FIMUL** [[*dest*,]*source*]

Format: **FMUL** *stack, memory*
**FMUL** *stack, register*
**FMUL(P)** *register, stack*
**FIMUL** *stack, memory*
**FMUL** *stack, register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FIMUL | ST(0),mem16i | DEh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 12-15 |
| FIMUL | ST(0),mem32i | DAh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 12 |
| FMUL | ST(0),mem32r | D8h | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 10-12 |
| FMUL | ST(0),mem64r | DCh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 15 |
| FMUL | ST(0),ST(n) | D8h | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | 11 |
| FMUL | ST(n),ST(0) | DCh | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | 11 |
| FMULP | ST(n),ST(0) | DEh | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | 11 |

The FMUL instruction fetches, then multiplies the operands. It changes the source into extended precision format, if necessary. The mode in effect determines how the result is rounded. The result is stored in the destination register.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 1 after a Precision exception if the rounding done by the instruction was upward.

Zero/Infinity:

| Operand 1 | Operand 2 | Result | Operand 1 | Operand 2 | Result |
|---|---|---|---|---|---|
| +0 | +0 | +0 | +∞ | +∞ | +∞ |
| +0 | -0 | -0 | +∞ | -∞ | -∞ |
| -0 | +0 | -0 | -∞ | +∞ | -∞ |
| -0 | -0 | +0 | -∞ | -∞ | +∞ |
| +x | +0 | +0 | +∞ | +x | +∞ |
| +x | -0 | -0 | +∞ | -x | -∞ |
| -x | +0 | -0 | -∞ | +x | -∞ |
| -x | -0 | +0 | -∞ | -x | +∞ |
| +x | +y | +0* | +∞ | +0 | Invalid |
| +x | -y | -0* | +∞ | -0 | Invalid |
| -x | +y | -0* | -∞ | +0 | Invalid |
| -x | -y | +0* | -∞ | -0 | Invalid |

* For cases in which the product of xy produces an extremely small number (i.e., underflow) and the Underflow exception is masked, the result is denormalized to 0.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Overflow | Masked | Infinity | | | | 1 | | | |
| | Unmasked | Round, scale | | | | 1 | | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Trap, unchanged | | | | | | | 1 |

## FNOP— No operation

Syntax: **FNOP**

Format: **FNOP**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FNOP | None | D9h | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | | 4 |

FNOP performs no operation and affects only instruction pointers.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

## FPATAN— Arctangent

Syntax: **FPATAN**

Format: **FPATAN**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FPATAN | None | D9h | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 20-261 |

FPATAN computes the arctangent function evaluation: $z = \tan^{-1}\left(\frac{y}{x}\right)$, where z is in radians. The first source operand, x, is the Top of Stack, ST(0); the second source operand, y, is the next to Top of Stack, ST(1). The result is rounded according to the mode in effect. The result, z, falls into the range $-\pi \leq z \leq \pi$. The instruction pops the Top of Stack and returns z to the new Top of Stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is 0 after normal execution. C1 is set to 1 after a Precision exception if the rounding done by the instruction was upward.

Zero/Infinity:

| y | x | Result |
|---|---|---|
| $y=+0$ | $+\infty \geq x \geq +0$ | $z=+0$ |
| $y=-0$ | $+\infty \geq x \geq +0$ | $z=-0$ |
| $y=+0$ | $-\infty \leq x \leq -0$ | $z=+\pi$ |
| $y=-0$ | $-\infty \leq x \leq -0$ | $z=-\pi$ |
| $y>+0$ | $x=0$ | $z=+\pi/2$ |
| $y>+0$ | $x=+\infty$ | $z=+0$ |
| $y>+0$ | $x=-\infty$ | $z=+\pi$ |
| $y<-0$ | $x=0$ | $z=-\pi/2$ |
| $y<-0$ | $x=+\infty$ | $z=-0$ |
| $y<-0$ | $x=-\infty$ | $z=-\pi$ |
| $y=+\infty$ | $-\infty<x<+\infty$ | $z=+\pi/2$ |
| $y=+\infty$ | $x=+\infty$ | $z=+\pi/4$ |
| $y=+\infty$ | $x=-\infty$ | $z=+3\pi/4$ |
| $y=-\infty$ | $-\infty<x<+\infty$ | $z=-\pi/2$ |
| $y=-\infty$ | $x=+\infty$ | $z=-\pi/4$ |
| $y=-\infty$ | $x=-\infty$ | $z=-3\pi/4$ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FPREM — Remainder

Syntax: **FPREM**

Format: **FPREM**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FPREM | None | D9h | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | 42-124 |

The FPREM instruction computes the remainder resulting from dividing source operand x, by the next to Top of Stack operand, y. The remainder, r, replaces the value in the Top of Stack. If the numbers divide evenly, the remainder is 0. FPREM can reduce the exponent of x by 63 or more each pass.

The remainder is determined by multiplying y by the quotient and subtracting the result from x. The quotient is the result of dividing x by y, then using the chopping method to truncate the exact value toward zero. The sign of the remainder is the same as that of the original value of x in the stack.

FPREM is provided for compatibility with industry standard 8087 and 80287 numeric coprocessors. However, FPREM does not adhere to the remainder operation specified in IEEE Standard 754. FPREM1 is compatible with the IEEE standard.

Condition Codes: When the quotient is completely reduced, its value can be read from the condition code bits C3, C1, and C0, as shown in the following table.

| Result of Instruction's Execution | | C0 | C1 | C2 | C3 |
|---|---|---|---|---|---|
| Reduction not completed | | 0 | 1 | 0 | 0 |
| Reduction completed: q(mod 8) | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 1 | 0 |
| | 2 | 1 | 0 | 0 | 0 |
| | 3 | 1 | 0 | 1 | 0 |
| | 4 | 0 | 0 | 0 | 1 |
| | 5 | 0 | 0 | 1 | 1 |
| | 6 | 1 | 0 | 0 | 1 |
| | 7 | 1 | 0 | 1 | 1 |

If the source register is empty, then C0, C2, and C3 are undefined; C1 is 0.

Zero/Infinity:

| x | y | Result |
|---|---|---|
| x=0 | y=0 | Invalid operation |
| x≠0 | y=0 | Invalid operation |
| x=-0 | y≠0 | -0 |
| x=+0 | y≠0 | +0 |
| x=∞ | | Invalid operation |
| -∞<x<+∞ | y=∞ | x=q=0 |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Underflow | Masked | Denormal | | | | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FPREM1 — IEEE remainder

Syntax: **FPREM1**

Format: **FPREM1**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FPREM1 | ST(0) | D9h | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | | 50-128 |

The FPREM1 instruction computes the remainder resulting from dividing source operand, x, by the next to ST(0) operand, y. The remainder, r, replaces the value in the Top of Stack. If the numbers divide evenly, the remainder is 0. FPREM1 can reduce the exponent of x by 63 or more on each pass.

The remainder is determined by multiplying y by the quotient and subtracting the result from x. The quotient is the result of dividing x by y, then rounding to the nearest number (or nearest even number in case of a tie).

The FPREM1 instruction is compatible with IEEE Standard 754.

Condition Codes: When the quotient is completely reduced, its value can be read from the condition code bits C3, C1, and C0, as shown in the following table.

| Result of Instruction's Execution | | C0 | C1 | C2 | C3 |
|---|---|---|---|---|---|
| Reduction not completed | | 0 | 1 | 0 | 0 |
| Reduction completed: q(mod 8) | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 1 | 0 |
| | 2 | 1 | 0 | 0 | 0 |
| | 3 | 1 | 0 | 1 | 0 |
| | 4 | 0 | 0 | 0 | 1 |
| | 5 | 0 | 0 | 1 | 1 |
| | 6 | 1 | 0 | 0 | 1 |
| | 7 | 1 | 0 | 1 | 1 |

If the source register is empty, then C0, C2, and C3 are undefined; C1 is 0.

Zero/Infinity:

| x | y | Result |
|---|---|---|
| x=0 | y=0 | Invalid operation |
| x≠0 | y=0 | Invalid operation |
| x=-0 | y≠0 | -0 |
| x=+0 | y≠0 | +0 |
| x=∞ | | Invalid operation |
| -∞<x<+∞ | y=∞ | x=q=0 |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Underflow | Masked | Denormal | | | | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FPTAN— Tangent

Syntax: **FPTAN**

Format: **FPTAN**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FPTAN | ST(0) | D9h | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | | 5-166 |

The FPTAN instruction calculates the tangent of x, the source operand taken from the Top of Stack. The source operand must be expressed in radians and must be in the range $| x | < 2^{63}$. FPTAN rounds the result (according to the mode in effect), places the value onto the Top of Stack, then pushes 1.0 onto the Top of Stack.

Condition Codes: C0 and C3 are undefined. C2 specifies reduction, where C2 = 1 means the reduction is incomplete. In case of a register error, C1 indicates the type of error, where C1 = 1 means the destination register is full and C1 = 0 means the source register is empty. C1 also indicates the type of rounding (away from 0) after a Precision exception.

Zero/Infinity:

| Operand 1 (x) | Result | Operand 1 (x) | Result |
|---|---|---|---|
| +0 | y=+0 | +∞ | Invalid operation |
| -0 | y=-0 | -∞ | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FRNDINT — Round to integer

Syntax: **FRNDINT**

Format: **FRNDINT**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FRNDINT | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | | 5-110 |

The FRNDINT instruction rounds the value in the Top of Stack to an integer. The rounding operation performed depends on the value of the RC field in the Control Word.

Condition Codes: C0, C2, and C3 are undefined. C1 indicates whether the source operand was rounded up (C1 = 1) or rounded down (C1 = 0). C1 is also set to 0 if the source register is empty.

Zero/Infinity:

| Operand 1 | Result | Operand 1 | Result |
|---|---|---|---|
| +0 | +0 | +∞ | +∞ |
| -0 | -0 | -∞ | -∞ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Denormal | Masked | Denormal used | 0 | | | | | 1 | |
| | Unmasked | Trap, abort | 0 | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FRSTOR — Restore FPU state

Syntax: **FRSTOR** *dest*

Format: **FRSTOR** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FRSTOR | mem94/108byte | DDh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 106 |

The FRSTOR instruction loads the SuperMathDX FPU environment and registers from the memory location specified by the source operand. The data structure of the environment depends on the operand size and the current operating mode. This instruction is normally executed in conjunction with FSAVE, which saves the environment and registers to a specific memory location.

The SuperMathDX FPU environment is determined by the Mode Control Word, the Status Word, the Tag Word, the Instruction Pointer, and the Data Pointer.

FRSTOR loads the registers after loading the environment. The registers are in the 80 bytes following the environment data.

Condition Codes: C0, C1, C2, and C3 are loaded from memory.

Exception Flags: Exceptions may be loaded with the environment. If the Status Word loaded with the environment contains an exception that is enabled, an error trap will occur after the next wait or after the execution of an Exception Status instruction.

## FSAVE — Save FPU state

Syntax: **FSAVE** *dest*

Format: **FSAVE** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSAVE | mem94/108byte | DDh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 106 |

The FSAVE instruction saves the SuperMathDX FPU environment and registers to the memory location specified by the source operand.

The SuperMathDX FPU environment is made up of the Mode Control Word, the Status Word, the Data Register Tag Word, the Instruction Pointer, and the Data Pointer.

FSAVE saves the registers in memory after the storing the environment. The registers are stored in the 80 bytes following the environment data.

The current 80386 operating mode and operand size determine the format of the environment, as shown in Figures 18, 19, 20, and 21.

*Figure 18. 16-Bit Protected Mode*

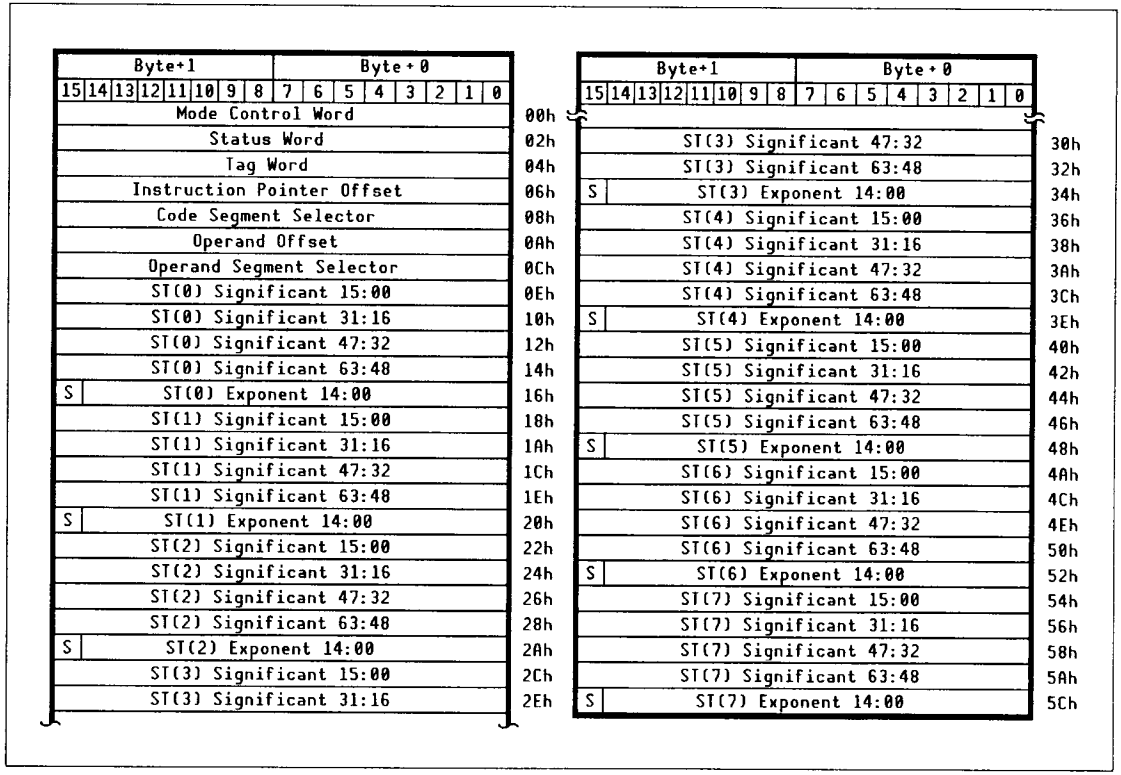


***Figure 19.*** *16-Bit Real Mode*

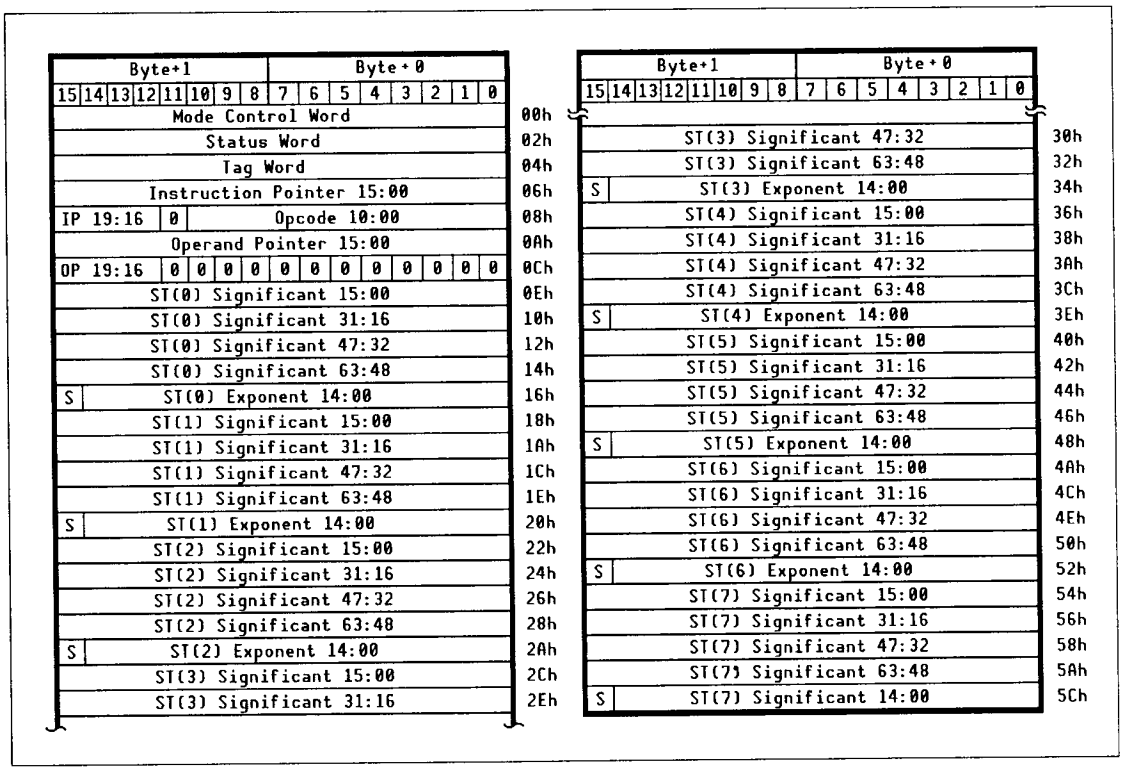
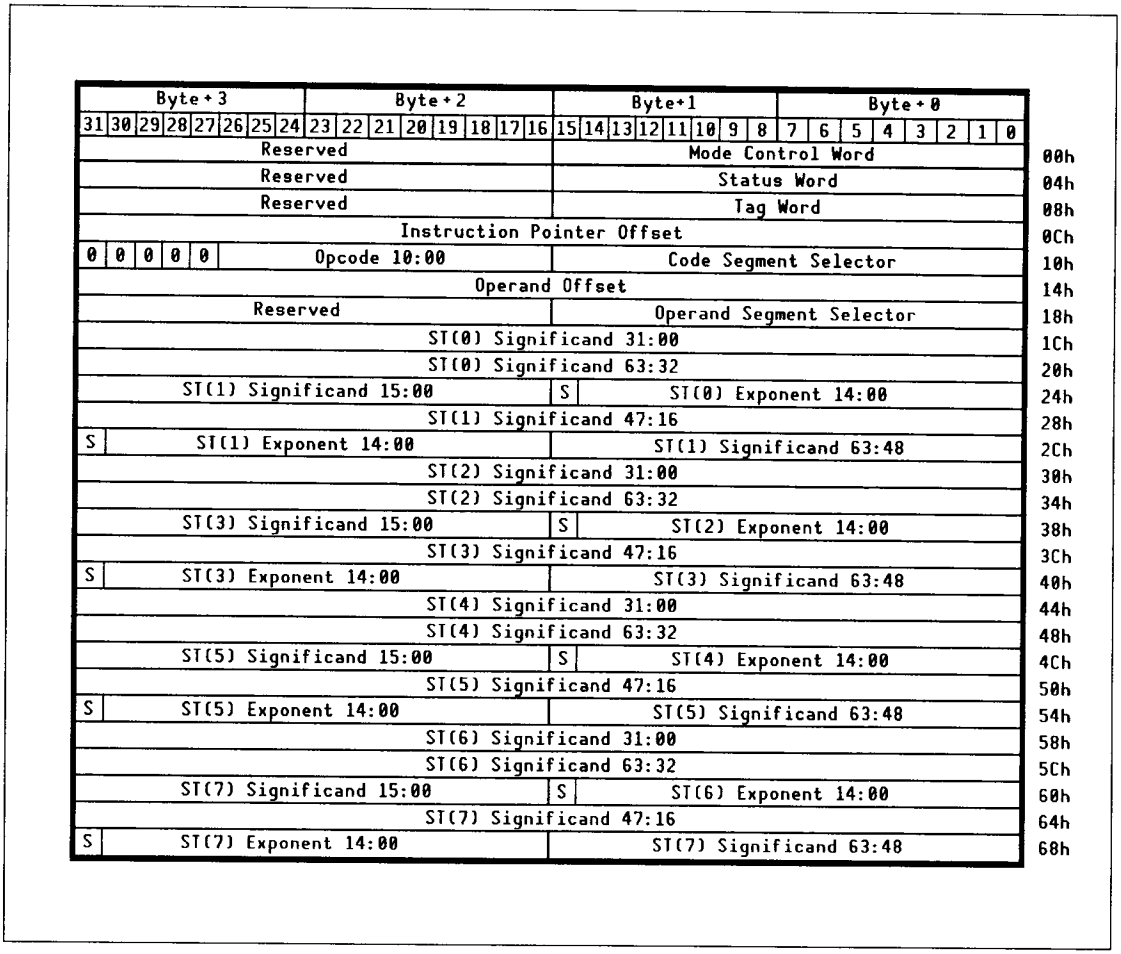


*Figure 20. 32-Bit Protected Mode*



| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Instruction Pointer Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | 0 | Opcode 10:00 | | | | | | | | | | | Code Segment Selector | | | | | | | | | | | | | | | | 10h |
| Operand Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14h |
| Reserved | | | | | | | | | | | | | | | | Operand Segment Selector | | | | | | | | | | | | | | | | 18h |
| ST(0) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1Ch |
| ST(0) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 20h |
| ST(1) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(0) Exponent 14:00 | | | | | | | | | | | | | | | 24h |
| ST(1) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 28h |
| S | ST(1) Exponent 14:00 | | | | | | | | | | | | | | | ST(1) Significand 63:48 | | | | | | | | | | | | | | | | 2Ch |
| ST(2) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 30h |
| ST(2) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 34h |
| ST(3) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(2) Exponent 14:00 | | | | | | | | | | | | | | | 38h |
| ST(3) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3Ch |
| S | ST(3) Exponent 14:00 | | | | | | | | | | | | | | | ST(3) Significand 63:48 | | | | | | | | | | | | | | | | 40h |
| ST(4) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 44h |
| ST(4) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 48h |
| ST(5) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(4) Exponent 14:00 | | | | | | | | | | | | | | | 4Ch |
| ST(5) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 50h |
| S | ST(5) Exponent 14:00 | | | | | | | | | | | | | | | ST(5) Significand 63:48 | | | | | | | | | | | | | | | | 54h |
| ST(6) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 58h |
| ST(6) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 5Ch |
| ST(7) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(6) Exponent 14:00 | | | | | | | | | | | | | | | 60h |
| ST(7) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 64h |
| S | ST(7) Exponent 14:00 | | | | | | | | | | | | | | | ST(7) Significand 63:48 | | | | | | | | | | | | | | | | 68h |

*Figure 21. 32-Bit Real Mode*

| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Reserved | | | | | | | | | | | | | | | | Instruction Pointer 15:00 | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | Instruction Pointer 31:16 | | | | | | | | | | | | 0 | Opcode 10:00 | | | | | | | | | | | | | | | | 10h |
| Reserved | | | | | | | | | | | | | | | | Operand Pointer 15:00 | | | | | | | | | | | | | | | | 14h |
| 0 | 0 | 0 | 0 | Operand Pointer 31:16 | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 18h |
| ST(0) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1Ch |
| ST(0) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 20h |
| ST(1) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(0) Exponent 14:00 | | | | | | | | | | | | | | | 24h |
| ST(1) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 28h |
| S | ST(1) Exponent 14:00 | | | | | | | | | | | | | | | ST(1) Significand 63:48 | | | | | | | | | | | | | | | | 2Ch |
| ST(2) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 30h |
| ST(2) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 34h |
| ST(3) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(2) Exponent 14:00 | | | | | | | | | | | | | | | 38h |
| ST(3) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3Ch |
| S | ST(3) Exponent 14:00 | | | | | | | | | | | | | | | ST(3) Significand 63:48 | | | | | | | | | | | | | | | | 40h |
| ST(4) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 44h |
| ST(4) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 48h |
| ST(5) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(4) Exponent 14:00 | | | | | | | | | | | | | | | 4Ch |
| ST(5) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 50h |
| S | ST(5) Exponent 14:00 | | | | | | | | | | | | | | | ST(5) Significand 63:48 | | | | | | | | | | | | | | | | 54h |
| ST(6) Significand 31:00 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 58h |
| ST(6) Significand 63:32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 5Ch |
| ST(7) Significand 15:00 | | | | | | | | | | | | | | | | S | ST(6) Exponent 14:00 | | | | | | | | | | | | | | | 60h |
| ST(7) Significand 47:16 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 64h |
| S | ST(7) Exponent 14:00 | | | | | | | | | | | | | | | ST(7) Significand 63:48 | | | | | | | | | | | | | | | | 68h |

Condition Codes: C0, C1, C2, and C3 are all set to 0.

Exception Flags: No exceptions.

## FSCALE— Multiply by 2n

Syntax: **FSCALE**

Format: **FSCALE**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSCALE | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | | 11 |

The FSCALE instruction multiplies the contents of the Top of Stack by $2^n$. (The value for n is the next to Top of Stack value truncated toward 0 to convert it into an integer.) The result is rounded according to the mode in effect and placed in the Top of Stack.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 except after a Precision exception, when it signifies whether rounding was towards or away from 0.

Zero/Infinity:

| x | n | Result |
|---|---|---|
| +0 | $-\infty$ | +0 |
| -0 | $-\infty$ | -0 |
| 0 | $+\infty$ | Invalid operation |
| $+\infty$ | $-\infty$ | Invalid operation |
| $-\infty$ | $-\infty$ | Invalid operation |
| $+\infty$ | $-\infty < n \leq +\infty$ | $+\infty$ |
| $-\infty$ | $-\infty < n \leq -\infty$ | $-\infty$ |
| $x \neq 0$ | $+\infty$ | sign(x)*$\infty$ |
| $x \neq 0$ | $-\infty$ | sign(x)*0 |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked<br>Unmasked | QNaN<br>Unchanged | 1<br>1 | | | | | | 1<br>1 |
| Precision | Masked<br>Unmasked | Rounded<br>Rounded | | 1<br>1 | | | | | |
| Underflow | Masked<br>Unmasked | Denormal/0<br>Round, scale | | 1 | 1<br>1 | | | | |
| Overflow | Masked<br>Unmasked | Infinity<br>Round, scale | | | | 1<br>1 | | | |
| Denormal | Masked<br>Unmasked | Denorm<br>Trap, abort | | | | | | 1<br>1 | |
| Invalid Operation | Masked<br>Unmasked | QNaN<br>Unchanged | | | | | | | 1<br>1 |

## FSIN — Function evaluation of sine(x)

Syntax: **FSIN**

Format: **FSIN**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSIN | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | | 5-126 |

The FSIN instruction calculates the sine of x, which is the source operand taken from the Top of Stack. The source operand must be expressed in radians and must be in the range $|x| < 2^{63}$. FSIN rounds the result (according to the mode in effect) and places the value onto the Top of Stack.

Condition Codes: C0 and C3 are undefined. C2 specifies reduction where C2 = 1 means the reduction is incomplete. In case of a Precision exception, C1 indicates the type of rounding (away from 0).

Zero/Infinity:

| Operand | Result | Operand | Result |
|---|---|---|---|
| +0 | +0 | +∞ | Invalid operation |
| -0 | -0 | -∞ | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FSINCOS— Function evaluation of sine(x) and cosine(y)

Syntax: **FSINCOS**

Format: **FSINCOS**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSINCOS | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | | 5-237 |

The FSINCOS instruction calculates the sine of x and the cosine of x. The source operand, x, is taken from the Top of Stack. The source operand must be expressed in radians and must be in the range $|x| < 2^{63}$. FSINCOS rounds the results (according to the mode in effect); places y, the result of sin(x), onto the top of stack; then pushes z, the result of cos(x), onto the stack.

Condition Codes: C0 and C3 are undefined. C2 specifies reduction, where C2 = 1 means the reduction is incomplete. In case of a register error, C1 indicates the type of error, where C1 = 1 means the destination register is full and C1 = 0 means the source register is empty. C1 also indicates the type of rounding (away from 0) after a Precision exception.

Zero/Infinity:

| Operand (x) | Result | Operand (x) | Result |
|---|---|---|---|
| +0 | y=+0, z=+1 | +∞ | Invalid operation |
| -0 | y=-0, z=+1 | -∞ | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FSQRT — Calculate the square root of x

Syntax: **FSQRT**

Format: **FSQRT**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSQRT | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | | 19-39 |

The FSQRT instruction calculates the square root of x. The source operand, x, is taken from the Top of Stack. FSQRT rounds the results to the appropriate precision (according to the mode in effect) and puts the result onto the Top of Stack.

Condition Codes: C0, C2, and C3 are undefined. C1 indicates the type of rounding (away from 0) after a Precision exception but is otherwise 0.

Zero/Infinity:

| Operand (x) | Result | Operand (x) | Result |
|---|---|---|---|
| +0 | +0 | $-\infty \le x < -0$ | Invalid operation |
| -0 | -0 | $+\infty$ | $+\infty$ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FST — Store register

Syntax: **FST(P)** *dest*

Format: **FIST(P)** *memory*
**FBSTP** *memory*
**FST(P)** *memory*
**FST(P)** *register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FIST | mem16i,ST(0) | DFh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FISTP | mem16i,ST(0) | DFh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FIST | mem32i,ST(0) | DBh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FISTP | mem32i,ST(0) | DBh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FISTP | mem64i,ST(0) | DFh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FBSTP | mem80b,ST(0) | DFh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 67 |
| FST | mem32r,ST(0) | D9h | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FSTP | mem32r,ST(0) | D9h | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 8 |
| FST | mem64r,ST(0) | DDh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 9 |
| FSTP | mem64r,ST(0) | DDh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 9 |
| FSTP | mem80r,ST(0) | DBh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,displ | 4 |
| FST | ST(n),ST(0) | DDh | 1 | 1 | 0 | 1 | 0 | REG2 | REG1 | REG0 | | 3 |
| FSTP | ST(n),ST(0) | DDh | 1 | 1 | 0 | 1 | 1 | REG2 | REG1 | REG0 | | 3 |

The FST instruction takes the value from the Top of Stack, converts the data format and rounds it, if necessary, and puts it into the destination register or memory location. The forms FISTP, FSTP, and FBSTP copy the value in the Top of Stack, then pop the stack.

The source operand is rounded to fit the size of the destination according to the mode in effect. If the source operand is a 0, ∞, or NaN, it is chopped on the right (not rounded) to fit the destination.

Condition Codes: C0, C2, and C3 are undefined. C1 indicates the type of rounding (away from 0) after a Precision exception but is otherwise 0.

Zero/Infinity:

| Operand (x) | Result | Operand (x) | Result |
|---|---|---|---|
| Empty | Invalid operation | ∞ → Integer | Invalid operation |
| NaN → Integer | Invalid operation | \| x \| > Integer range | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Rounded | | 1 | 1 | | | | |
| | Unmasked | Trap, abort | | | 1 | | | | |
| Overflow | Masked | Rounded | | | | 1 | | | |
| | Unmasked | Trap, abort | | | | 1 | | | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Trap, abort | | | | | | | 1 |

## FSTCW — Store control word

Syntax: **FSTCW** *dest*

Format: **FSTCW** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSTCW | mem2byte | D9h | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | | 5 |

The FSTCW instruction stores the contents of the Mode Control Word in the specified destination in memory.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

## FSTENV — Store environment

Syntax: **FSTENV** *dest*

Format: **FSTENV** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FSTENV | mem14/28byte | D9h | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,displ | 56 |

The FSTENV instruction saves the SuperMathDX FPU environment to the specified memory location. The 80386 mode and operand size determine the format of the environment as shown in Figure 22. The environment is made up of the Mode Control Word, the Status Word, the Data Register Tag Word, the Instruction Pointer, and the Data Pointer.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

*Figure 22. SuperMathDX Coprocessor Environment*

**16-Bit Protected Mode**

| Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Status Word | | | | | | | | | | | | | | | | 02h |
| Tag Word | | | | | | | | | | | | | | | | 04h |
| Instruction Pointer Offset | | | | | | | | | | | | | | | | 06h |
| Code Segment Selector | | | | | | | | | | | | | | | | 08h |
| Operand Offset | | | | | | | | | | | | | | | | 0Ah |
| Operand Segment Selector | | | | | | | | | | | | | | | | 0Ch |

**16-Bit Real Mode**

| Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Status Word | | | | | | | | | | | | | | | | 02h |
| Tag Word | | | | | | | | | | | | | | | | 04h |
| Instruction Pointer 15:00 | | | | | | | | | | | | | | | | 06h |
| IP 19:16 | | | | 0 | Opcode 10:0 | | | | | | | | | | | 08h |
| Operand Pointer 15:00 | | | | | | | | | | | | | | | | 0Ah |
| OP 19:16 | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0Bh |

**32-Bit Protected Mode**

| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Instruction Pointer Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | 0 | Opcode 10:0 | | | | | | | | | | | Code Segment Selector | | | | | | | | | | | | | | | | 10h |
| Operand Offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14h |
| Reserved | | | | | | | | | | | | | | | | Operand Segment Selector | | | | | | | | | | | | | | | | 18h |

**32-Bit Real Mode**

| Byte+3 | | | | | | | | Byte+2 | | | | | | | | Byte+1 | | | | | | | | Byte+0 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Reserved | | | | | | | | | | | | | | | | Mode Control Word | | | | | | | | | | | | | | | | 00h |
| Reserved | | | | | | | | | | | | | | | | Status Word | | | | | | | | | | | | | | | | 04h |
| Reserved | | | | | | | | | | | | | | | | Tag Word | | | | | | | | | | | | | | | | 08h |
| Reserved | | | | | | | | | | | | | | | | Instruction Pointer 15:00 | | | | | | | | | | | | | | | | 0Ch |
| 0 | 0 | 0 | 0 | Instruction Pointer 31:16 | | | | | | | | | | | | 0 | Opcode 10:00 | | | | | | | | | | | | | | | | 10h |
| Reserved | | | | | | | | | | | | | | | | Operand Pointer 15:00 | | | | | | | | | | | | | | | | 14h |
| 0 | 0 | 0 | 0 | Operand Pointer 31:16 | | | | | | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | 18h |

## FSTSW — Store status word

Syntax: **FSTSW** *dest*

Format: **FSTSW** *memory*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FSTSW | mem2byte | DDh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | | 5 |
| FSTSW | 80386 AX Reg | DFh | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | 5 |

The FSTSW instruction stores the contents of the Status Word in the specified destination in memory or in the AX register of the 80386 compatible processor.

Condition Codes: C0, C1, C2, and C3 are undefined.

Exception Flags: No exceptions.

## FSUB — Floating-point subtraction

Syntax: **FSUB(R)(P)** [[*dest*,] *source*]

Format:
**FISUB(R)** *stack, memory*
**FSUB(R)** *stack, memory*
**FSUB(R)** *stack, register*
**FSUB(R)(P)** *register, stack*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FISUB | ST(0),mem16i | DEh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 13-15 |
| FISUBR | ST(0),mem16i | DEh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 13-15 |
| FISUB | ST(0),mem32i | DAh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 17 |
| FISUBR | ST(0),mem32i | DAh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 17 |
| FSUB | ST(0),mem32r | D8h | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 10-15 |
| FSUBR | ST(0),mem32r | D8h | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 10-15 |
| FSUB | ST(0),mem64r | DCh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,displ | 14-17 |
| FSUBR | ST(0),mem64r | DCh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,displ | 14-17 |
| FSUB | ST(n),ST(0) | D8h | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | 7-9 |
| FSUBR | ST(n),ST(0) | D8h | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | 7-9 |
| FSUB | ST(n),ST(0) | DCh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | 7-9 |
| FSUBR | ST(n),ST(0) | DCh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | 6-7 |
| FSUBP | ST(n),ST(0) | DEh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | 6-7 |
| FSUBRP | ST(n),ST(0) | DEh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | 6 |

The source operand is subtracted from the destination. The result is rounded (according to the mode in effect) and returned to the destination.

When using either of the pop forms (FSUBP or FSUBRP), the instruction pops the ST(0). The reverse forms (FISUBR, FSUBR, or FSUBRP) subtract the destination from the source.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 except after a Precision exception, where it defines whether or not rounding is away from 0.

Zero/Infinity:

| Operand 1 | Operand 2 | Result | Operand 1 | Operand 2 | Result |
|---|---|---|---|---|---|
| +0 | +0 | R(0) | +∞ | +∞ | +∞ |
| -0 | -0 | R(0) | -∞ | -∞ | -∞ |
| +0 | -0 | +0 | +∞ | x | +∞ |
| -0 | +0 | -0 | -∞ | x | -∞ |
| +x | +x | R(0) | x | -∞ | +∞ |
| -x | -x | R(0) | x | +∞ | -∞ |
| +∞ | +∞ | Invalid | | | |
| -∞ | -∞ | Invalid | | | |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Overflow | Masked | Infinity | | | | 1 | | | |
| | Unmasked | Round, scale | | | | 1 | | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FTST — Test top of stack

Syntax: **FTST**

Format: **FTST**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FTST | ST(0) | D9h | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | 3 |

The FTST instruction compares the value in the Top of Stack to 0. The condition codes reflect the result of the comparison.

Condition Codes: The result of the comparison is determined by the condition code bits C3, C2, C1, and C0, as follows:

| ST(0) Status | C3 | C2 | C1 | C0 |
|---|---|---|---|---|
| ST(0) >0 | 0 | 0 | 0 | 0 |
| ST(0) <0 | 0 | 0 | 0 | 1 |
| ST(0)=+0 | 1 | 0 | 0 | 0 |
| ST(0)=-0 | 1 | 0 | 1 | 0 |
| Unordered | 1 | 1 | 0 | 1 |

If the source operand is either a NaN or an unsupported number, or if a stack fault occurs, an Invalid exception occurs and the condition code bits are undefined.

If the source register is empty, then C0, C2, and C3 are undefined; C1=0.

Zero/Infinity:

| ST(0) | Result | ST(0) | Result |
|---|---|---|---|
| +0 | =0 | +∞ | >0 |
| -0 | =0 | -∞ | <0 |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register error | Masked | Unchanged | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Denorm used | | | | | | 1 | |
| Invalid operation | Masked | Unchanged | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

QNaNs cause an invalid operation exception to occur.

## FUCOM — Unordered compare

Syntax: **FUCOM(P)** [[*dest*,] *source*]

Format: **FUCOM(P)(P)** *stack,register*

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FUCOM | ST(n),ST(0) | DDh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | 4 |
| FUCOMP | ST(n),ST(0) | DDh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | 5 |
| FUCOMPP | ST(1),ST(0) | DAh | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | 7 |

The FUCOM instruction compares the source to the Top of Stack. The condition codes reflect the result of the comparison. FUCOMP pops the Top of Stack. FUCOMPP compares the Top of Stack to the next to Top of Stack and then pops the Top of Stack twice.

Condition Codes: The result of the comparison is determined by the condition code bits C3, C2, C1, C0, as follows:

| dest/source Status | C3 | C2 | C1 | C0 |
|---|---|---|---|---|
| dest>source | 0 | 0 | 0 | 0 |
| dest<source | 0 | 0 | 0 | 1 |
| dest=source | 1 | 0 | 0 | 0 |
| Unordered | 1 | 1 | 0 | 1 |

If the source operand is either a NaN or an unsupported number, or if a stack fault occurs, an Invalid exception occurs and the condition code bits are undefined.

If the source register is empty, C3 = C2 = C0 = 1 and C1 = 0.

Zero/Infinity:

| dest | source | Result | dest | source | Result |
|---|---|---|---|---|---|
| +0 | +0 | = | +∞ | +∞ | = |
| -0 | -0 | = | -∞ | -∞ | = |
| +0 | -0 | = | +∞ | -∞ | dest>src |
| -0 | +0 | = | -∞ | +∞ | dest<src |
| +0 | +x | dest<src | +∞ | x | dest>src |
| -0 | +x | dest<src | -∞ | x | dest<src |
| +0 | -x | dest>src | x | -∞ | dest>src |
| -0 | -x | dest>src | x | +∞ | dest<src |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | Unordered | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Denorm used | | | | | | 1 | |
| Invalid Operation | Masked | Unordered | | | | | | 1 | |
| | Unmasked | Unchanged | | | | | | 1 | |

## FXAM — Examine operand

Syntax: **FXAM**

Format: **FXAM**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FXAM | ST(0) | D9h | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | 3 |

The FXAM instruction examines the source operand (Top of Stack) and reports its type by setting the condition codes.

Condition Codes: The condition codes are set to report the result of the examination.

| ST(0) Contents | C3 | C2 | C0 |
|---|---|---|---|
| Unnormal | 0 | 0 | 0 |
| NaN | 0 | 0 | 1 |
| Normal | 0 | 1 | 0 |
| Infinity | 0 | 1 | 1 |
| Zero | 1 | 0 | 0 |
| Empty | 1 | 0 | 1 |
| Denormal | 1 | 1 | 0 |

C1 reflects the sign of the source operand, where 0 = positive and 1 = negative.

Exception Flags: No exceptions.

## FXCH — Exchange register contents

Syntax: **FXCH**

Format: **FXCH**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | Bytes 3-7 | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | |
| FXCH | ST(n),ST(0) | D9h | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | 4 |

The FXCH instruction swaps the contents of the Top of Stack and the source register.

Condition Codes: C1 is always 0. C0, C2, and C3 are undefined after normal execution and are set to 1 as a result of a register error.

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register | Masked | Unordered | 1 | | | | | | 1 |
| Error | Unmasked | Unchanged | 1 | | | | | | 1 |

## FXTRACT — Extract exponent

Syntax: **FXTRACT**

Format: **FXTRACT**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FXTRACT | ST(0) | D9h | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | | 6 |

The FXTRACT instruction operates on the value contained in ST(0). The exponent portion of ST(0) is converted into 80-bit extended precision format and is pushed onto the stack (y). The exponent of x is set to 0 (i.e., 3FFFh biased) but its sign is preserved.

Condition Codes: C0, C2, and C3 are undefined. C1 is set to 0 after normal execution. In case of a register error, C1 indicates the type of error, where C1 = 1 means the destination register is full and C1 = 0 means the source register is empty.

Zero/Infinity:

| Operand 1 | Result |
|---|---|
| +0 | y=+0, x=−∞, Divide by zero exception |
| −0 | y=−0, x=−∞, Divide by zero exception |
| +∞ | y=−∞, x=+∞ |
| −∞ | y=+∞, x=+∞ |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Divide by Zero | Masked | ST(0)=0, ST(1)=$-\infty$; | | | | | 1 | | |
| | Unmasked | Trap, abort | | | | | 1 | | |
| Denormal | Masked | Denormal used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Trap, unchanged | | | | | | | 1 |

## FYL2X — Compute y * log2(x)

Syntax: **FYL2X**

Format: **FYL2X**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FYL2X | ST(0) | D9h | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | | 12-236 |

The FYL2X instruction computes the base 2 logarithm of x, which is the value in the Top of Stack. It then multiplies the result by the next to Top of Stack value, y, rounds the result according to the mode in effect, places the result in the next to Top of Stack, then pops the stack.

If x is negative, the invalid operation exception occurs.

Condition Codes: C0, C2, and C3 are undefined. C1 indicates the type of rounding (away from 0) after a Precision exception but is otherwise 0.

Zero/Infinity:

| x | y | Result |
|---|---|---|
| x<0 | | Invalid operation |
| x=0 | y≠0 | Divide by 0 exception |
| x=0 | y=0 | Invalid operation |
| x=1 | y=∞ | Invalid operation |
| x>1 | y=∞ | y |
| 0<x<1 | y=∞ | -y |
| x=∞ | y>+0 | +∞ |
| x=∞ | y<-0 | -∞ |
| x=∞ | y=0 | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Overflow | Masked | Infinity | | | | 1 | | | |
| | Unmasked | Round, scale | | | | 1 | | | |
| Divide by Zero | Masked | ST(0)=-∞ | | | | | 1 | | |
| | Unmasked | Trap, abort | | | | | 1 | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## FYL2XP1 — Compute y * $\log_2$ (x+1)

Syntax: **FYL2XP1**

Format: **FYL2XP1**

Description:

| Instruction | Operand(s) (dest, source) | Opcode/Instruction Encoding | | | | | | | | | Clock Cycles |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1st Byte | 2nd Byte | | | | | | | Bytes 3-7 | |
| | | | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | |
| FYL2XP1 | ST(0) | D9h | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | | 5-227 |

The FYL2XP1 instruction computes the base 2 logarithm of x (the Top of Stack) plus 1.0. It then multiplies the result by the next to Top of Stack value, y, rounds the result according to the mode in effect, places the result in the next to Top of Stack, then pops the stack.

FYL2XP1 is used to compute the logarithm of numbers whose absolute value is close to 1. FYL2XP1 is more accurate than FYL2X in this case. The operand, x, (the Top of Stack) must be in the range -(1-SQRT(2)/2)<x<1-SQRT(2)/2.

Condition Codes: C0, C2, and C3 are undefined. C1 indicates the type of rounding (away from 0) after a Precision exception but is otherwise 0.

Zero/Infinity:

| x | y | Result |
|---|---|---|
| x=-0 | y≥+0 | -0 |
| x=+0 | y≥+0 | +0 |
| x=-0 | y≤-0 | +0 |
| x=+0 | y<-0 | -0 |
| x=0 | y=∞ | Invalid operation |
| x>0 | y=∞ | y |
| -1<x<0 | y=∞ | -y |
| x=∞ | y>+0 | +∞ |
| x=∞ | y<-0 | -∞ |
| x=∞ | y=0 | Invalid operation |

Exception Flags:

| Exception | Mode | Result | S | P | U | O | Z | D | I |
|---|---|---|---|---|---|---|---|---|---|
| Register Error | Masked | QNaN | 1 | | | | | | 1 |
| | Unmasked | Unchanged | 1 | | | | | | 1 |
| Precision | Masked | Rounded | | 1 | | | | | |
| | Unmasked | Rounded | | 1 | | | | | |
| Underflow | Masked | Denormal/0 | | 1 | 1 | | | | |
| | Unmasked | Round, scale | | | 1 | | | | |
| Denormal | Masked | Denorm used | | | | | | 1 | |
| | Unmasked | Trap, abort | | | | | | 1 | |
| Invalid Operation | Masked | QNaN | | | | | | | 1 |
| | Unmasked | Unchanged | | | | | | | 1 |

## SuperMathDX Operands

Table 17 summarizes the operands and shows the encoding for all SuperMath FPU instructions.

*Table 17. SuperMathDX Operands*

| Opcode/Instruction Encoding | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | Instruction | Operand(s) (dest,source) |
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | | |
| D8h | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | FADD | ST(0), ST(n) |
| D8h | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | FMUL | ST(0), ST(n) |
| D8h | 1 | 1 | 0 | 1 | 0 | REG2 | REG1 | REG0 | | FCOM | ST(n), ST(0) |
| D8h | 1 | 1 | 0 | 1 | 1 | REG2 | REG1 | REG0 | | FCOMP | ST(n), ST(0) |
| D8h | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | FSUB | ST(n), ST(0) |
| D8h | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | FSUBR | ST(n), ST(0) |
| D8h | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | FDIV | ST(0), ST(n) |
| D8h | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | FDIVR | ST(0), ST(n) |
| D8h | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FADD | ST(0), mem32r |
| D8h | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FMUL | ST(0), mem32r |
| D8h | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FCOM | ST(0), mem32r |
| D8h | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FCOMP | ST(0), mem32r |
| D8h | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FSUB | ST(0), mem32r |
| D8h | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FSUBR | ST(0), mem32r |
| D8h | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FDIV | ST(0), mem32r |
| D8h | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FDIVR | ST(0), mem32r |
| D9h | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | FLD | ST(0), ST(n) |
| D9h | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | FXCH | ST(n), ST(0) |
| D9h | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | | FNOP | None |
| D9h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | FCHS | ST(0) |
| D9h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | FABS | ST(0) |
| D9h | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | FTST | ST(0) |
| D9h | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | | FXAM | ST(0) |
| D9h | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | | FLD1 | ST(0), const |
| D9h | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | | FLDL2T | ST(0), const |
| D9h | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | | FLDL2E | ST(0), const |
| D9h | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | | FLDPI | ST(0), const |
| D9h | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | | FLDLG2 | ST(0), const |

**Table 17.** *SuperMathDX Operands (continued)*

| Opcode/Instruction Encoding | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | Instruction | Operand(s) (dest,source) |
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | | |
| D9h | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | | FLDLN2 | ST(0), const |
| D9h | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | | FLDZ | ST(0), const |
| D9h | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | | F2XMI | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | | FYL2X | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | | FPTAN | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | | FPATAN | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | | FXTRACT | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | | FPREM1 | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | FDECSTP | None |
| D9h | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | | FINCSTP | None |
| D9h | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | FPREM | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | | FYL2XP1 | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | | FSQRT | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | | FSINCOS | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | | FRNDINT | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | | FSCALE | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | | FSIN | ST(0) |
| D9h | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | FCOS | ST(0) |
| D9h | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FLD | ST(0), mem32r |
| D9h | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FST | mem32r, ST(0) |
| D9h | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FSTP | mem32r, ST(0) |
| D9h | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FLDENV | mem14/28byte |
| D9h | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FLDCW | mem2byte |
| D9h | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FSTENV | mem14/28byte |
| D9h | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | | FSTCW | mem2byte |
| D9Ah | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | FUCOMPP | ST(1), ST(0) |
| DAh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIADD | ST(0), mem32i |
| DAh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FIMUL | ST(0), mem32i |
| DAh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FICOM | ST(0), mem32i |
| DAh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FICOMP | ST(0), mem32i |
| DAh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FISUB | ST(0), mem32i |
| DAh | MD1 | MD0 | 1 | 0 | 1 | RM;2 | RM1 | RM0 | SIB,DISPL | FISUBR | ST(0), mem32i |
| DAh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIDIV | ST(0), mem32i |

*Table 17. SuperMathDX Operands (continued)*

| Opcode/Instruction Encoding | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | Instruction | Operand(s) (dest,source) |
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | | |
| DAh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FIDIVR | ST(0), mem32i |
| D8h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | See Note 1 | |
| D8h | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | | See Note 2 | |
| DBh | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | | FCLEX | ST(n), ST(0) |
| DBh | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | | FINIT | None |
| DBh | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | | See Note 3 | |
| DBh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FILD | ST(0), mem32i |
| DBh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIST | mem32i, ST(0) |
| DBh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FISTP | mem32i, ST(0) |
| DBh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FLD | ST(0), mem80r |
| DBh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FSTP | ST(n), ST(0) |
| DCh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | FADD | ST(n), ST(0) |
| DCh | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | FMUL | ST(n), ST(0) |
| DCh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | FSUBR | ST(n), ST(0) |
| DCh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | FSUB | ST(n), ST(0) |
| DCh | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | FDIVR | ST(n), ST(0) |
| DCh | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | FDIV | ST(n), ST(0) |
| DCh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FADD | ST(0), mem64r |
| DCh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FMUL | ST(0), mem64r |
| DCh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FCOM | ST(0), mem64r |
| DCh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FCOMP | ST(0), mem64r |
| DCh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FSUB | ST(0), mem64r |
| DCh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FSUBR | ST(0), mem64r |
| DCh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FDIV | ST(0), mem64r |

1 This encoding may be generated by some assemblers. However, the SuperMathDX coprocessor executes an FNOP instruction when it is presented this encoding, maintaining industry standard compatibility. This encoding corresponds to the 8087/287 FENI instruction.

2 This encoding may be generated by some assemblers. However, the SuperMathDX coprocessor executes a FNOP instruction when it is presented this encoding, maintaining industry standard compatability. This encoding corresponds to the 8087/287 FDISI instruction.

3 This encoding may be generated by some assemblers. However, the SuperMathDX coprocessor executes a FNOP instruction when it is presented this encoding, maintaining industry standard compatability. This encoding corresponds to the 8087/287 FSETPM instruction.

*Table 17. SuperMathDX Operands (continued)*

| Opcode/Instruction Encoding | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | Instruction | Operand(s) (dest,source) |
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | | |
| DCh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FDIVR | ST(0), mem64r |
| DDh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | FFREE | ST(n) |
| DDh | 1 | 1 | 0 | 1 | 0 | REG2 | REG1 | REG0 | | FST | ST(n), ST(0) |
| DDh | 1 | 1 | 0 | 1 | 1 | REG2 | REG1 | REG0 | | FSTP | mem80r, ST(0) |
| DDh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | FUCOM | ST(n), ST(0) |
| DDh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | FUCOMP | ST(n), ST(0) |
| DDh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FLD | ST(0), mem64r |
| DDh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FST | mem64r, ST(0) |
| DDh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FSTP | mem64r, ST(0) |
| DDh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FRSTOR | mem94/108byte |
| DDh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FSAV | mem94/108byte |
| DDh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | | FSTSW | mem2byte |
| DEh | 1 | 1 | 0 | 0 | 0 | REG2 | REG1 | REG0 | | FADDP | ST(n), ST(0) |
| DEh | 1 | 1 | 0 | 0 | 1 | REG2 | REG1 | REG0 | | FMULP | ST(n), ST(0) |
| DEh | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | | FCOMPP | ST(1), ST(0) |
| DEh | 1 | 1 | 1 | 0 | 0 | REG2 | REG1 | REG0 | | FSUBR | ST(n), ST(0) |
| DEh | 1 | 1 | 1 | 0 | 1 | REG2 | REG1 | REG0 | | FSUBP | ST(n), ST(0) |
| DEh | 1 | 1 | 1 | 1 | 0 | REG2 | REG1 | REG0 | | FDIVRP | ST(n), ST(0) |
| DEh | 1 | 1 | 1 | 1 | 1 | REG2 | REG1 | REG0 | | FDIVP | ST(n), ST(0) |
| DEh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIADD | ST(0), mem16i |
| DEh | MD1 | MD0 | 0 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FIMUL | ST(0), mem16i |
| DEh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FICOM | ST(0), mem16i |
| DEh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FICOMP | ST(0), mem16i |
| DEh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FISUB | ST(0), mem16i |
| DEh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FISUBR | ST(0), mem16i |
| DEh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIDIV | ST(0), mem16i |
| DEh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | | FIDIVR | ST(0), mem16i |

*Table 17. SuperMathDX Operands (continued)*

| Opcode/Instruction Encoding | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1st Byte | 2nd Byte | | | | | | | | Bytes 3-7 | Instruction | Operand(s) (dest,source) |
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | | | |
| DFh | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | FSTSW | 80386 AX Register |
| DFh | MD1 | MD0 | 0 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FILD | ST(0), mem16i |
| DFh | MD1 | MD0 | 0 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FIST | mem16i, ST(0) |
| DFh | MD1 | MD0 | 0 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FISTP | mem16i, ST(0) |
| DFh | MD1 | MD0 | 1 | 0 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FBLD | ST(0), mem806 |
| DFh | MD1 | MD0 | 1 | 0 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FILD | ST(0), mem64i |
| DFh | MD1 | MD0 | 1 | 1 | 0 | RM2 | RM1 | RM0 | SIB,DISPL | FBSTP | mem80b, ST(0) |
| DFh | MD1 | MD0 | 1 | 1 | 1 | RM2 | RM1 | RM0 | SIB,DISPL | FISTP | mem64i, ST(0) |

# AC/DC Characteristics

The tables and figures in this section describe the operating environment and signal timings required by the SuperMathDX coprocessor. The signal timings correspond to those of 80386-compatible CPUs and provide complete interface compatibility.

## General Characteristics

Use of the SuperMathDX coprocessor within the ranges specified in Table 18 is mandated for guaranteed operation.

***Table 18.*** *Recommended Operating Conditions*

| Symbol | Parameter | Min. | Max. | Unit | Notes |
|---|---|---|---|---|---|
| $V_{cc}$ | Supply voltage | 4.75 | 5.25 | V | |
| $T_c$ | Case temperature | 0 | 70 | °C | |
| $I_{oh}$ | Output high current | — | -1.0 | mA | $V_{oh}$ = min $V_{oh}$ |
| $I_{ol}$ | Output low current | — | 4.0 | mA | $V_{ol}$ = max $V_{ol}$ |
| $I_{ik}$ | Input clamp current | — | ±10 | mA | $V_{in}$ $V_{ss}$ or $V_{in}$ $V_{cc}$ |
| $I_{ok}$ | Output clamp current | — | ±25 | mA | $V_{out}$ $V_{ss}$ or $V_{out}$ $V_{cc}$ |

Table 19 states the most extreme conditions that will be tolerated by the chip without permanent chip damage. Note that these are not continuous operating conditions; the chip will probably be damaged if operated near these limits for a prolonged period of time.

***Table 19.*** *Maximum Tolerated Conditions*

| Symbol | Parameter | Min. | Max. | Unit |
|---|---|---|---|---|
| $V_{cc}$ | Supply voltage | -0.5 | 6.0 | V |
| | Case temperature (powered) | 0 | 100 | °C |
| | Storage temperature (unpowered) | -65 | 150 | °C |
| | Voltage on any pin | -0.5 | $V_{cc}$ + 0.5 | V |
| | Power dissipation at 5.25V | — | 1.9 | W |

Table 20 shows input, output, and I/O capacitance values for 5V operation.

**Table 20.** *Capacitance*

| Symbol | Parameter | Max. | Unit | Test Condition |
|---|---|---|---|---|
| $C_{in}$ | Input capacitance | 10 | pF | $f_c$ = 1MHz |
| $C_{out}$ | I/O, output capacitance | 12 | pF | $f_c$ = 1MHz |
| $C_{clk}$ | Clock capacitance | 20 | pF | $f_c$ = 1MHz |

# DC Characteristics

Table 21 provides the DC operating characteristics of the SuperMathDX coprocessor.

**Table 21.** *DC Characteristics*

| Symbol | Parameter | Min. | Max. | Unit | Test Condition |
|---|---|---|---|---|---|
| $V_{il}$ | Input low voltage | 0 | 0.8 | V | |
| $V_{ih}$ | Input high voltage | 2.0 | $V_{cc}$ | V | |
| $V_{cl}$ | Clock input low voltage | 0 | 0.8 | V | |
| $V_{ch}$ | Clock input high voltage | 3.7 | $V_{cc}$ | V | |
| $V_{ol}$ | Output low voltage | — | 0.45 | V | $I_{ol}$ = 4.0mA |
| $V_{oh}$ | Output high voltage | 2.4 | — | V | $I_{oh}$ = 1.0mA |
| $I_{il}$ | Input leakage current | — | ±15 | µA | $V_{in}$ = 0V to $V_{cc}$ |
| $I_{oz}$ | Output leakage current | — | ±15 | µA | $V_{out}$ = 0V to $V_{cc}$ |
| $I_{cc}$ | $V_{cc}$ supply current | — | 380 | mA | @66.7MHz (100mA typical) |

# AC Characteristics

Table 22 shows the AC operating characteristics of a 33MHz SuperMathDX coprocessor. The timings listed are with respect to the CPUCLK2 input unless otherwise noted.

***Table 22.*** *AC Operating Characteristics*

| Symbol | Parameter | Min. | Max. | Unit | Figure Number |
|---|---|---|---|---|---|
| **Data Bus (D31:0) Timings** | | | | | |
| t1 | Data output delay ($C_L$ = 50pF) | 0 | 37 | ns | 21 |
| t2 | Data output float time ($C_L$ = 50pF) | 3 | 19 | ns | 21 |
| t3 | Data input setup time | 8 | | ns | 21 |
| t4 | Data input hold time | 8 | | ns | 21 |
| **Output Signal (READYO*, PEREQ, BUSY*, ERROR*) Timings** | | | | | |
| t5 | READY0* output delay ($C_L$ = 50pF) | 3 | 17 | ns | 23 |
| t5 | PEREQ output delay ($C_L$ = 50pF) | 4 | 25 | ns | 23 |
| t5 | BUSY* output delay ($C_L$ = 50pF) | 4 | 21 | ns | 23 |
| t5 | ERROR* output delay ($C_L$ = 50pF) | 4 | 25 | ns | 23 |
| t6 | Float time from ZSTEN* active/delay time from ZSTEN* inactive | 1 | 30 | ns | 23 |
| **Control Signal (CMD0*, NPCS1*, NPCS2, W/R*) Timings** | | | | | |
| t7 | Setup time | 13 | | ns | 21 |
| t8 | Hold time | 4 | | ns | 21 |
| **Other Signal Timings** | | | | | |
| t9 | ADS* setup time | 13 | | ns | 21 |
| t10 | ADS* hold time | 4 | | ns | 21 |
| t11 | READY* setup time | 7 | | ns | 21 |
| t12 | READY* hold time | 4 | | ns | 21 |
| t13 | ZSTEN* setup time | 13 | | ns | 21 |
| t14 | ZSTEN* hold time | 2 | | ns | 21 |
| t15 | RESETIN setup time | 5 | | ns | 22 |
| t16 | RESETIN hold time | 3 | | ns | 22 |
| t17 | RESETIN duration | 40 | | CPUCLK2s | 22 |
| t18 | RESETIN inactive before first opcode write | 50 | | CPUCLK2s | 22 |
| **Clock Input (CPUCLK2) Timings** | | | | | |
| t19 | Period | 15 | 125 | ns | 25 |
| t20 | Time above 2.0V | 6.25 | | ns | 25 |

**Table 22.** *AC Operating Characteristics (continued)*

| Symbol | Parameter | Min. | Max. | Unit | Figure Number |
|---|---|---|---|---|---|
| **Clock Input (CPUCLK2) Timings (continued)** | | | | | |
| t21 | Time above min $V_{ch}$ | 4.5 | | ns | 25 |
| t22 | Time below 2.0V | 6.25 | | ns | 25 |
| t23 | Time below 0.8V | 4.5 | | ns | 25 |
| t24 | Fall time from min $V_{ch}$ to 0.8V | | 6 | ns | 25 |
| t25 | Rise time from 0.8V to min $V_{ch}$ | | 6 | ns | 25 |
| **Control Interface Relative Timings** | | | | | |
| t26 | BUSY* duration[1] | 90.9 | | ns | 24 |
| t27 | ERROR* active to BUSY* inactive | 6 | | CPUCLK2s | 24 |
| t28 | PEREQ inactive to ERROR* active | 6 | | CPUCLK2s | 24 |
| t29 | READY* active to BUSY* active | 4 | 4 | CPUCLK2s | 24 |
| t30 | Time from opcode write to next write (opcode or operand) | 6 | | CPUCLK2s | 24 |
| t31 | Time from operand write to next operand write | 8 | | CPUCLK2s | 24 |

**Figure 23.** *Data I/O Timings*

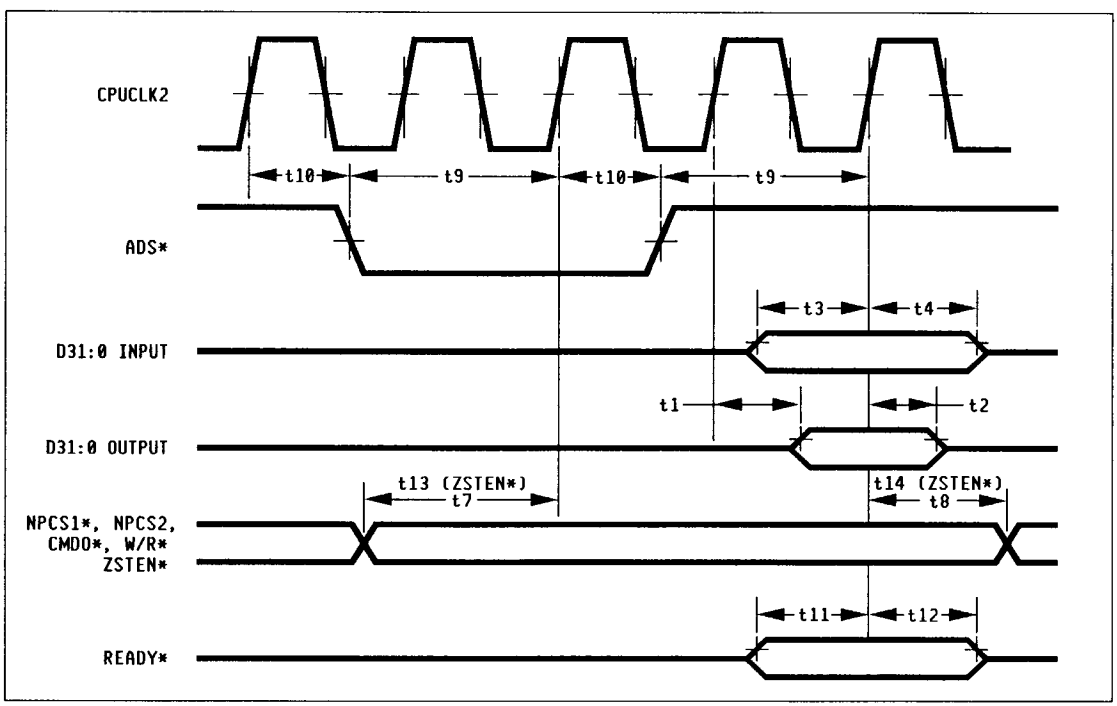


**Figure 24.** *Reset Timings*

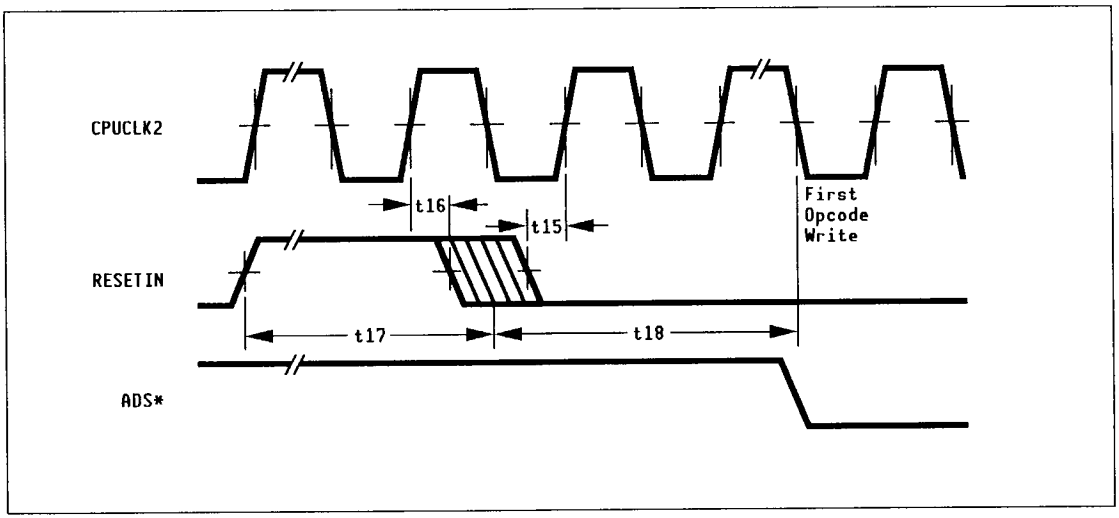



**Figure 25.** *Timing After Change of ZSTEN* State*

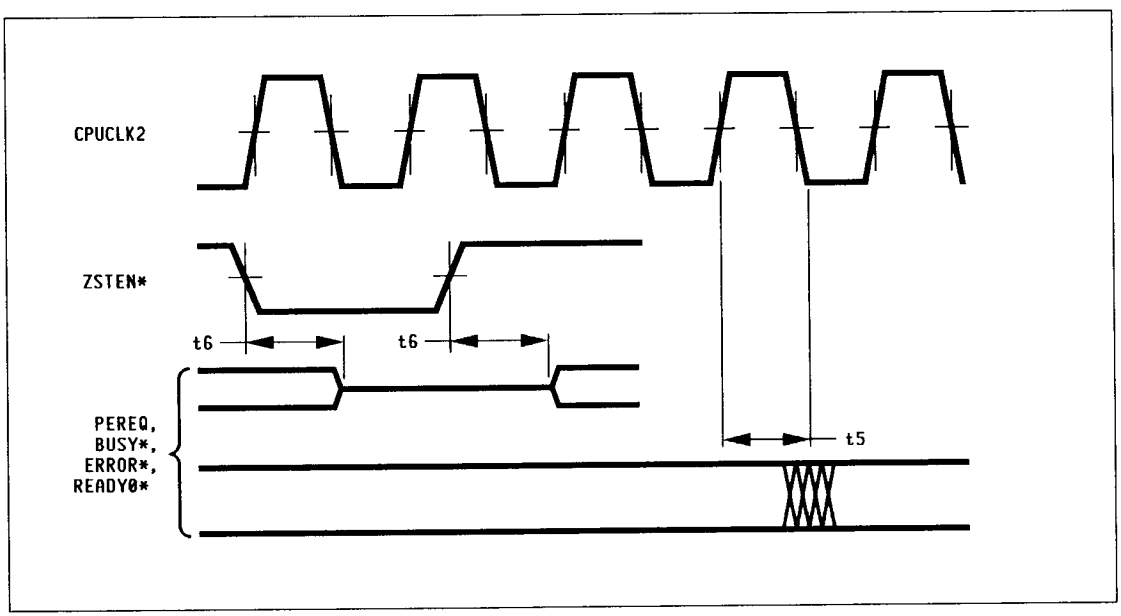


**Figure 26.** *Relative Control Signal Timings*

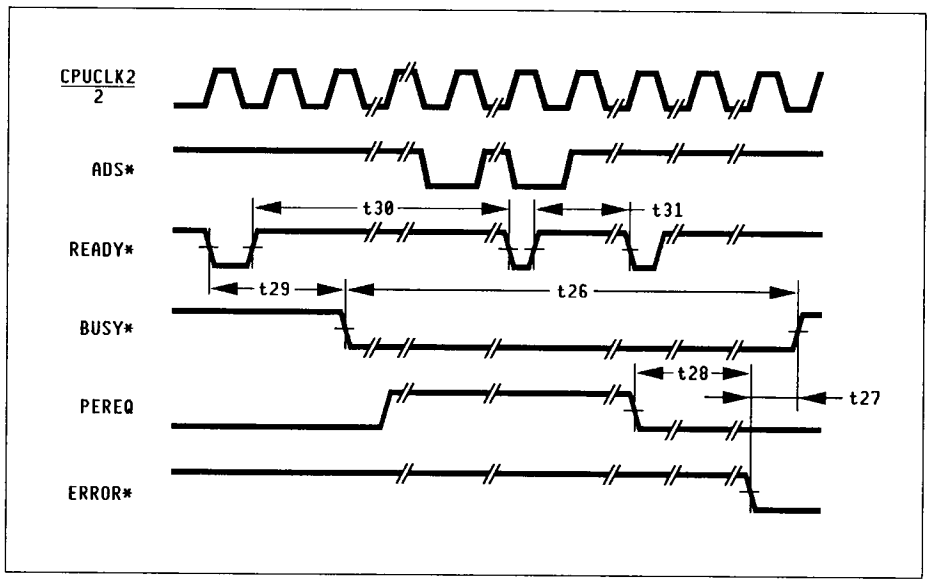
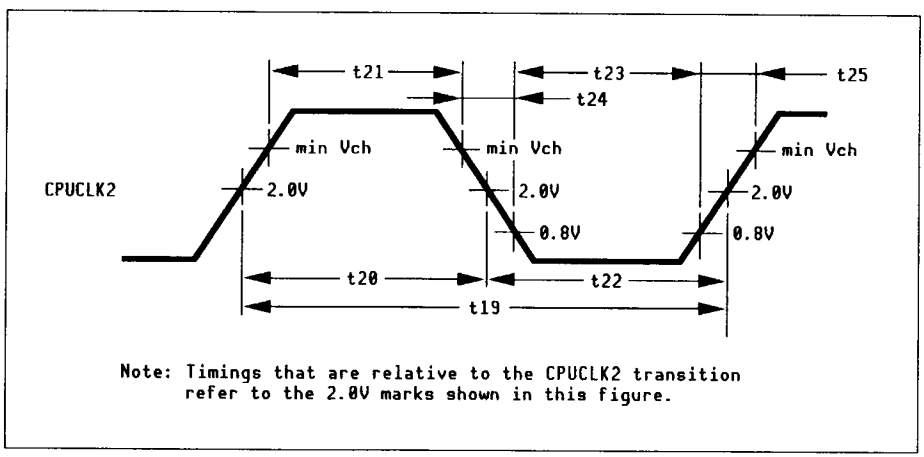



**Figure 27.** *CPUCLK2 Waveform Characteristics*



# Packaging Dimensions

Figure 28 shows the dimensions of the 68-pin PGA package.

**Figure 28.** *Ceramic Pin Grid Array Package*